Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA DE SEGOVIA

Grado en Ingeniería Informática de Servicios y Aplicaciones

CARGA Y VISUALIZACIÓN DE MEDIDAS RADIOMÉTRICAS

Autor: **Ángel Martín Bartolomé**

Tutores: **Luis Mª Fuentes García**
Ramiro González Cartón

Agradecimientos

Inicialmente tengo que agradecer a todos los profesores de la carrera por todos aquellos conocimientos que me han transmitido y me han ayudado a la realización de este proyecto. Por supuesto agradecer a mis tutores toda su colaboración, su tiempo y su paciencia a la hora de contestar todos y cada uno de mis correos con las dudas que me han ido surgiendo. Por otro lado, también mencionar a Fernando Díaz que, a pesar de que no es mi tutor del trabajo, me ofreció una tutoría que me ayudó bastante a la hora de saber cómo estructurar el proyecto.

Como es obvio, agradecer a toda aquella gente cercana que me ha apoyado desde mis inicios de la carrera hasta ahora, especialmente en mis momentos de agobio y nerviosismo por exámenes y entregas de trabajos más próximas de lo deseado… Muchas gracias es especial a mis padres, a mi hermano, mi cuñada y por otro lado a todos aquellos compañeros de la carrera que me han estado soportando hasta ahora y que muchos de ellos han pasado de ser compañeros a ser amigos. En especial a Borja Martín que, conviviendo con él en Madrid, me ha ayudado a ponerme las pilas en momentos de pereza y a desconectar en momentos de saturación con el proyecto.

¡Muchas gracias a todos!

Resumen

El objetivo de este proyecto es que, a partir de la red de instrumentos radiométricos instalados en estaciones ubicadas a lo largo del mundo, sea posible representar todos aquellos datos recogidos por todos y cada uno de los dispositivos con su previa carga y formalización de datos en su conveniente estructura de almacenamiento.

La principal aplicación que tiene la recogida y representación de todos estos datos, van desde el conocimiento de la microfísica de los radiómetros instalados en las estaciones, hasta temas tan importantes actuales como el estudio del cambio climático o la calidad del aire con sus implicaciones en la salud pública.

Para lograr esos objetivos, se decide automatizar los procesos relacionados con la carga y la representación de datos mediante el desarrollo de software:

- Gestión de la información generada por los dispositivos: Verificar que los datos generados por los radiómetros son correctos y entonces proceder a su carga en una base de datos.
- Apoyo a la tarea investigadora: A través de la representación de la información de la base de datos, se pretende facilitar a los usuarios de la red de radiómetros la realización de análisis sobre los datos.

En este proyecto se implementan estos objetivos proporcionando una potente herramienta a la comunidad científica.

Abstract

The aim of this project is to provide a frame for data representation. The data from all the radiometric instruments placed in station all over the world is loaded and organised for later representation.

The main application of this goes from the knowledge of the microphysics of the radiometers installed in the stations, to as important current issues as the study of climate change or air quality related with its implications in the public health.

It is decided to automate the processes related to the loading and representation of data using software development:

- Management of the information generated by the devices: The system checks the data generated by the radiometers are correct, then the system loads it into a database.
- Support for the research task: Through the graphic representations, the system makes easier to perform analysis of the data to the users of the radiometer network.

These objectives are implemented providing a powerful tool to the scientific community.

Índice de contenido

Índice de tablas

Índice de ilustraciones

Capítulo 1
Introducción

1.1 Motivación

El sistema climático se define como un sistema complejo e interactivo compuesto por la atmósfera, superficie terrestre, hielo y nieve. Este es impulsado por la radiación solar principalmente en el rango de onda corta hasta 3000nm. Sin embargo, no toda la radiación que llega a la atmósfera es absorbida por esta y para después alcanzar la superficie terrestre; aproximadamente el 30% es reflejado hacia el espacio por las nubes, aerosoles o la superficie de la Tierra, principalmente en aquellos lugares con un alto albedo (nieve, hielo y desiertos). El otro ~70% es absorbido por la superficie terrestre y la atmósfera. Para equilibrar la radiación entrante en la Tierra, ésta emite la misma cantidad de energía como radiación de onda larga. Esta energía se transporta alrededor de la Tierra y se usa en funciones tales como la evaporación del agua. La *Ilustración 1* muestra los principales procesos que influyen en el equilibrio energético, donde a largo plazo, la cantidad de radiación solar incidente absorbida por la Tierra y la atmósfera alcanza su equilibrio al liberarse por parte de la Tierra y la atmósfera la misma cantidad de radiación emitida de onda larga

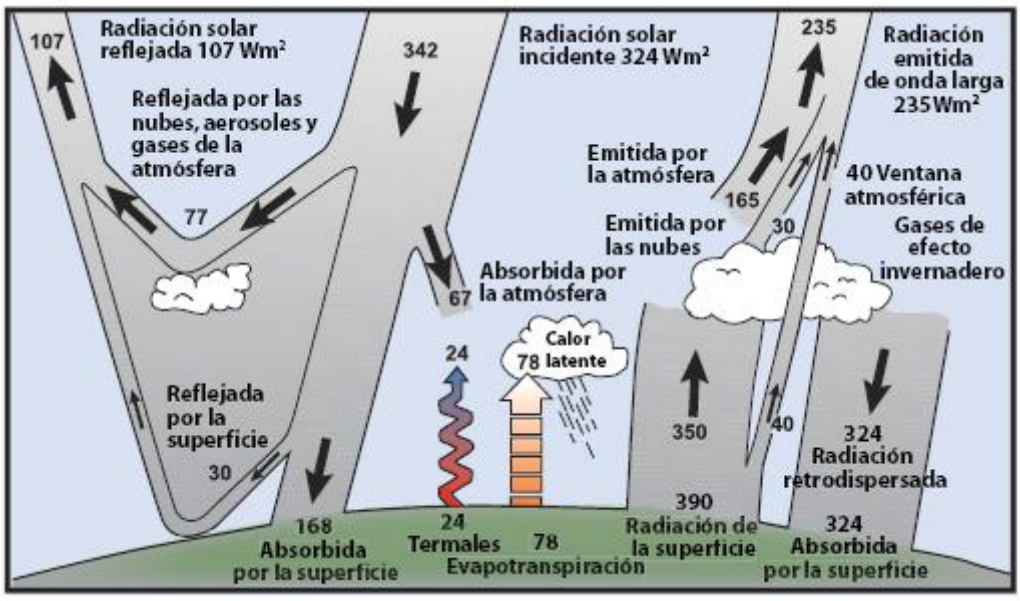


Ilustración 1: Cálculo del equilibrio anual y mundial de energía

Las nubes son el principal factor involucrado en mantener el equilibrio energético ya que pueden reflejar la radiación solar al espacio (efecto del albedo de las nubes) y atrapar la radiación infrarroja emitida por la superficie y la troposfera inferior (efecto invernadero de las nubes). La relación entre los dos efectos depende de muchos factores, incluyendo propiedades microfísicas de las nubes (IPCC, 2007).

El Sol es la estrella más cercana a la Tierra y es un cuerpo totalmente gaseoso compuesto principalmente de hidrógeno. Es la principal fuente de la energía que llega a la Tierra. La distribución espectral de la radiación al llegar al exterior de la atmósfera, se conoce como la radiación extraterrestre que cubre un intervalo de hasta 3000nm, dividido en tres zonas principales: ultravioleta (UV), visible (VIS) e infrarrojo (LW). El 95% de la energía del Sol cae en el rango de 300-2400nm.

Monitorizar la radiación solar en la superficie de la Tierra requiere una caracterización detallada del instrumento y una calibración precisa para proporcionar datos de radiación de alta calidad. En los últimos años, el uso de los instrumentos de a bordo en satélites también se han dirigido a la evaluación de los niveles de radiación solar en la superficie (e.g., Fioletov et al., 2004; Meloni et al., 2005; Ialongo et al., 2011).

RADIACIÓN SOLAR

La energía solar resulta del proceso de fusión nuclear que tiene lugar en el Sol. Esta energía es el motor que mueve nuestro medio ambiente, siendo la energía solar que llega a la superficie terrestre 10.000 veces mayor que la energía consumida actualmente por toda la humanidad.

Esta energía llega a la tierra en forma de radiación, ondas electromagnéticas caracterizadas por su longitud de onda (o frecuencia) que forman el espectro solar. Estas ondas electromagnéticas se desplazan por el vacío a la velocidad de la luz ($3 \cdot 10^8 m/s$) y su frecuencia determina su energía, visibilidad o penetración atmosférica.

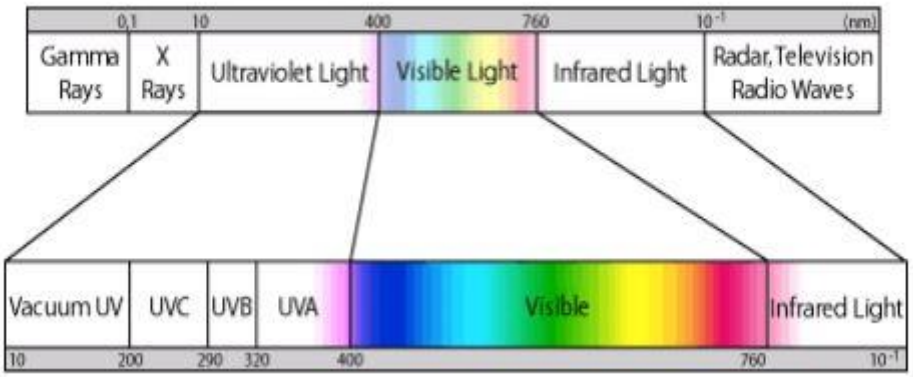


Ilustración 2: Espectro Solar

La proporción de radiación solar en las diferentes regiones del espectro es aproximadamente la siguiente:

Región	Proporción
Ultravioleta	7%
Luz visible	43%
Infrarrojo	49%
El resto	1%

Tabla 1: Espectro Solar

Es realmente importante conocer en detalle cada una de estas medidas de radiación ya que permite realizar estudios sobre las transformaciones de la energía en el sistema Tierra-Atmósfera, de analizar las propiedades y la distribución de la atmósfera, así como los elementos que la forman (aerosoles, vapor de agua, ozono…). También es relevante conocer estas medidas para estudiar la distribución y las variaciones de la radiación incidente, reflejada y la total. Por último, es de gran ayuda para satisfacer las necesidades derivadas de las actividades de la biología, la medicina, de la agricultura, de la arquitectura, de la ingeniería y de la industria relacionadas con la radiación.

A continuación, se describen los tipos de radiaciones solares dividiéndose en dos tipos, según su frecuencia (UV, PAR e LW) y su dirección (directa, difusa o global)

- ## UV (Radiación Ultravioleta)

La radiación ultravioleta es una radiación electromagnética no visible generada por el Sol. Aproximadamente el 7% de la energía que emite el Sol es radiación ultravioleta. Cubre un rango espectral desde los 100 a los 400 nm y se divide de la siguiente manera:

- Ultravioleta C de 100 a 280 nm. absorbida totalmente por el ozono.
- Ultravioleta B de 280 a 320 nm. absorbida parcialmente por el ozono.
- Ultravioleta A de 320 a 400 nm. apenas absorbida por el ozono.

Aunque aparentemente el 7% no resulta un porcentaje muy alto respecto a la radiación total, los efectos que provoca a los seres vivos y al medio ambiente hace que sea muy importante.

Este tipo de radiación tiene influencia sobre la salud (cáncer de piel, cataratas…), el clima (variación del balance energético, terrestre…), procesos biológicos (fotosíntesis), ecológicos (modificación de ecosistemas) y fotoquímicos (formación y descomposición de contaminantes). Todo esto unido a una reducción continuada de la capa de ozono provoca un auge en esta radiación UV con resultado muy dañino para el ser humano.

En este proyecto se tiene en cuenta la **radiación ultravioleta eritemática**, que expresa los efectos de enrojecimiento de la piel causada por el sol. Para llegar a ese valor se debe haber aplicado un factor de calibración sobre la radiación UV. A partir de esos resultados se puede determinar la peligrosidad de estas radiaciones.

- ## PAR (Radiación Fotosintéticamente Activa)

La radiación fotosintéticamente activa es la cantidad de radiación integrada del rango de longitudes de onda que son capaces de producir actividad fotosintética. Se encuentra en el umbral de los 400 a 700 nm dentro del espectro de radiación visible.

Conocer este tipo de longitudes de onda resulta útil para adecuar la iluminación en entornos artificiales o para elegir mallas de sombreo aplicadas en la obtención de la radiación solar difusa.

- ## LW (Radiación Infrarroja)

Se denomina la radiación de longitud de onda mayor que la visible. El espectro infrarrojo, que se extiende desde los 700 nm hasta las 100 micras se divide en tres regiones: el cercano (0,7 – 5 micras), mediano infrarrojo (5 – 30 micras) y el lejano infrarrojo (30 – 100 micras). En este caso, se partirá de las 4 micras.

- ## DNI (Radiación Solar Directa Normal)

Se basa en la radiación solar que se mide a través de un *pìrheliómetro*. Se trata de un sensor de radiación solar que mide el grado normal de incidencia de irradiación solar directa de onda corta. Únicamente mide la radiación procedente del sol y de una región anular del cielo muy próxima al sol.

Ilustración 3: Pirheliómetro de 1ª clase

- DIF (Radiación Solar Difusa)

Para realizar mediciones y obtener únicamente la componente difusa de la radiación solar, la componente de radiación directa se cubre por medio de un sistema de pantalla o sombreado.

- SW (Radiación Solar Global)

La radiación global se define como la radiación solar recibida de un ángulo solido de 2π estereorradianes sobre una superficie horizontal. La radiación global incluye la recibida directamente del disco solar y también la radiación celeste difusa dispersada al atravesar la atmósfera. El rango espectral va desde los 300 a los 3000 nm. Se debe tener en cuenta que, en este caso, la radiación directa incluida va en función de la posición del sol (no es directamente la medida de Radiación Directa Normal).

En este caso se utiliza un piranómetro de tipo *Kipp-Zonen*:

Ilustración 4: Piranómetro Kipp-Zonen

1.2 Objetivos

Debido a que el Grupo de Óptica Atmosférica de la Universidad de Valladolid (GOA) está trabajando con dispositivos radiométricos, el objetivo de este trabajo es proporcionar soporte en la gestión de las medidas y su representación. Para ello, el proyecto se divide en dos sistemas:

- **Sistema de Carga**

El principal objetivo de este sistema es liberar carga de trabajo al grupo a la hora de procesar y almacenar cada uno de los ficheros que llegan al servidor desde los dispositivos radiométricos. Con la realización de este proyecto todo este proceso de carga se realiza de forma automática.

El almacenamiento de la información en la base de datos centralizada del GOA (CAELIS) que permita al grupo llevar una buena gestión y tener una mejor accesibilidad a la información de las mediciones recogidas por los dispositivos. A partir de ella se pueden realizar filtrados sobre los datos y crear numerosas aplicaciones que traten este tipo de información.

El sistema también lleva un registro con los resultados de los procesados y las cargas a la base de datos. De esta manera se tiene una trazabilidad que permita llevar un control, tanto de los ficheros que han dado resultados fallidos, como los que han sido cargados satisfactoriamente.

- **Sistema de Visualización**

El objetivo de poder representar toda la información acerca de las mediciones radiométricas que hay en la base de datos, está orientado a la investigación científica. Que cualquier usuario tenga la posibilidad de realizar un análisis sobre los valores de radiación solar que han sido o están siendo recogidos en cualquiera de las estaciones radiométricas del GOA.

Por otro lado, la creación de un API REST que permita acceder a la información de la base de datos, da la posibilidad de realizar nuevas aplicaciones que quieran utilizar este tipo de datos.

1.3 Contenido del DVD

El DVD que se adjunta en la entrega del proyecto contiene los siguientes recursos:

- **Memoria_AngelMartin.pdf**
  - Se trata del documento pdf correspondiente a la memoria del proyecto.

- **Programas_EntornoDesarrollo**
  - **Base de Datos**: incluye los recursos necesarios para crear las tablas en la BD.
    - Incluye *DDL_radiometros.sql*
  - **Caelis_radiometros**: formado por los programas del sistema de carga. Están configurados para desplegarse en mi localhost tras su compilación
    - **Programs**: código fuente *.c*, *.h*,*.sh y .cnf*.
  - **Radiometros_Web**: formado por los programas del sistema de visualización.
    - **Radiometros_back**: código de la aplicación NodeJS del servidor.
    - **RadiometrosAngular**: código de la aplicación web.

- **Programas_EntornoProduccion**
  - **Caelis_radiometros**: sistema de carga configurado para desplegarse sobre el servidor real.
    - **Programs**: scripts *.sh* y programas compilados.
    - **Files**: distribución de los ficheros *.rad* con los que trabajan los programas.
  - **Radiometros_Web**: recursos del sistema de visualización.
    - **Radiometros_back:** Servidor NodeJS configurado para acceder a la base de datos real. En él está la aplicación web Angular desplegada en el directorio */public/radiometer*.

- **FicherosPrueba**
  - Incluye los ficheros con los que se ha trabajados para hacer pruebas.

Capítulo 2
Gestión del proyecto

2.1. Metodología de trabajo

Desde un planteamiento inicial, el proyecto de divide en 3 fases con funciones bien diferenciadas:

- **1ª Fase**: Desarrollo de los programas encargados de la asimilación y la carga de datos.
- **2ª Fase**: Desarrollo del programa encargado de gestionar la ejecución del software de la primera fase, así como la gestión su trazabilidad.
- **3ª Fase**: Desarrollo del módulo web que de visibilidad a los datos almacenados y tratados para su posterior análisis.

A partir de este esbozado del proyecto, se plantean tres subsistemas y para el correcto resultado funcional de cada uno de ellos, se ha utilizado un desarrollo ágil basado en el **modelo iterativo e incremental**. Esto quiere decir que cada una de las tres fases planteadas va a estar formada por un conjunto iteraciones o etapas en las que se repite un proceso de trabajo similar y de esta manera proporcionar un resultado completo sobre el producto final.

Las actividades que tienen lugar dentro de cada iteración son la elicitación, el análisis, el diseño, la implementación, las pruebas y finalmente la documentación:

- **ELICITACIÓN**

En esta fase se lleva a cabo la captura de los requisitos del producto que los clientes han solicitado. Se descubren todas aquellas piezas de información relevantes para el desarrollo del software y así obtener un resultado lo más satisfactorio posible para el cliente.

Las técnicas que se utilizan son las reuniones periódicas, pudiendo ser ésas de forma virtual por videollamada o de forma personal. Por otro lado, también se resolverán dudas y se aclararán requisitos y funcionalidades mediante la comunicación vía email.

- **ANÁLISIS**

Esta fase plasma toda la información relevante recogida de la fase anterior ofreciendo así un conocimiento más preciso del funcionamiento del sistema a construir. Básicamente ofrece respuesta a la pregunta ¿Qué se quiere desarrollar?

Esto se realiza mediante diferentes tipos de representaciones como diagramas Entidad-Relación, OCL, árbol de características… Por otro lado, descripciones de las fuentes de datos, identificación de los actores que interactúan con el sistema…

- **DISEÑO**

A partir del análisis se produce una descripción de la estructura interna del software que sirva como base para su construcción. Se representa la arquitectura a nivel lógico y físico de sus componentes y de esta manera se responde a la pregunta ¿Cómo se va a desarrollar?

En esta fase los datos se representan a nivel lógico con un modelo relacional. Por otro lado, también se requieren diagramas de actividades o de estados que proporcionen una visión más detallada del flujo de trabajo del sistema y su comportamiento.

- **IMPLEMENTACIÓN**

Se desarrolla el código fuente que tendrá el software partiendo del diseño que previamente se ha establecido.

- **PRUEBAS**

Una vez que el código permite su ejecución, es el momento de detectar posibles fallos o comportamientos anómalos en el sistema.

- **DOCUMENTACIÓN**

Se redacta cada una de las fases del proyecto y su información relevante permitiendo así conservar su historia, facilitar la utilización del software y mejorar el entendimiento por parte del usuario.

La secuencia de actividades de cada iteración se puede ver representada en la *Ilustración 5*. Como se puede observar, este modelo está basado en un modelo en cascada en el que el comienzo de una actividad debe esperar a la finalización de la inmediata anterior. Por otro lado, se tiene en cuenta que ante la detección de alguna anomalía en alguna de las actividades o ante alguna petición de cambio por parte del cliente, se puede retomar la actividad anterior para realizar modificaciones.

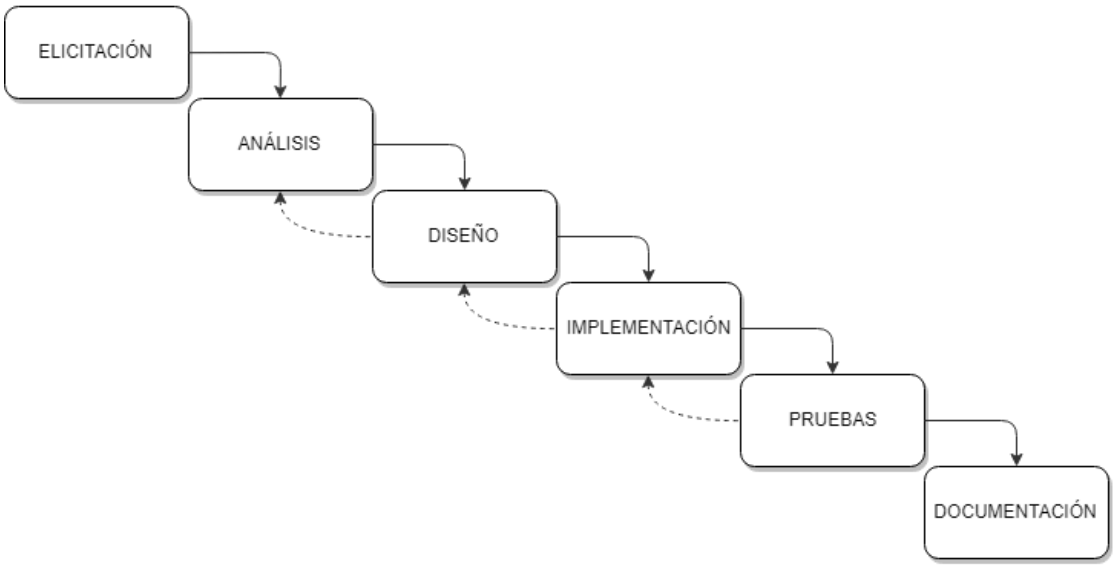


Ilustración 5: Flujo de actividades de una iteración

El esquema general de desarrollo del proyecto basado en el modelo iterativo incremental se puede ver reflejado en la *Ilustración 6*. En este caso, el modelo permite al desarrollador sacar ventaja, en una determinada iteración, de lo que se ha aprendido a lo largo del desarrollo de la iteración anterior. Una vez más, ante la presencia de una situación excepcional, se puede retomar el desarrollo de una iteración anterior para realizar las modificaciones pertinentes.

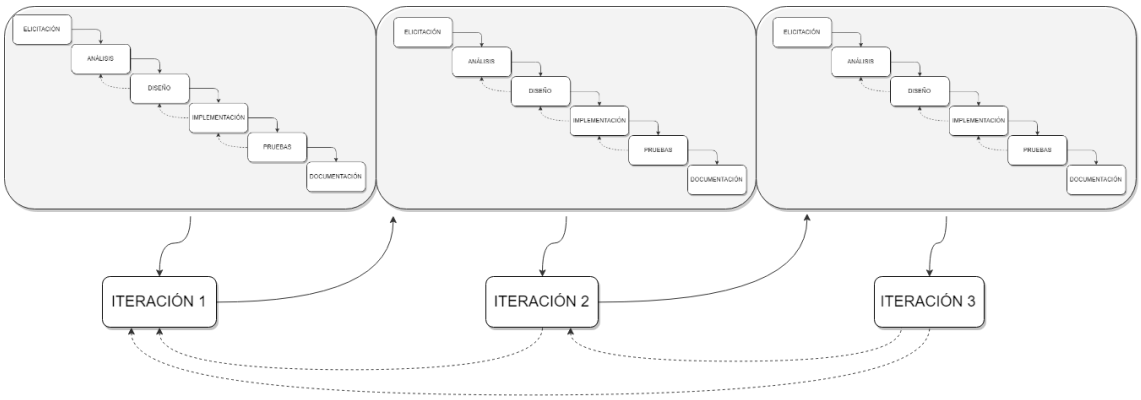


Ilustración 6: Modelo iterativo incremental

2.2. Planificación temporal estimada

Partiendo de finales de enero como fecha de inicio del proyecto, interesa estimar cual será la fecha aproximada en la que finalizará el mismo. A priori, el objetivo es presentar el proyecto antes del inicio del verano, ofreciendo un producto funcional y bien documentado.

En base a las 3 fases principales del proyecto, la *Ilustración 7* muestra los plazos temporales que han sido considerados, teniendo en cuenta la metodología previamente descrita. Como se puede apreciar, las tareas que se repiten en cada una de las fases tienen una duración más o menos similar. Hay que tener en cuenta que el tiempo estimado para cada una de las fases se ha hecho de manera empírica.

En la fase inicial, es natural que la actividad de elicitación tenga una duración mayor que en el resto de las fases debido a que, aparte de que no se conoce el proyecto que el cliente solicita, es de vital importancia tener claro, desde el inicio, cuáles son las funcionalidades y resultados requeridos del sistema.

En el caso de la última fase, a pesar de que se solicitan funcionalidades tan básicas como representar datos de forma gráfica, se estima una duración mayor que para el resto por dos motivos. En primer lugar, la persona encargada del desarrollo no cuenta con experiencia en la tecnología *Symfony2* por lo que será necesario un periodo de formación que, a su vez, implica una duración mayor en la actividad de implementación de dicha tecnología. La otra razón que alarga esta última fase del proyecto es la actividad de documentación, ya que está próxima a la conclusión del proyecto.

A continuación, en la *Ilustración 7* se representa una tabla con las tres fases y sus respectivas actividades a las que se las ha asignado una duración en días estimados. Se ha tenido en cuenta que el calendario de trabajo es de 5 horas diarias durante los 7 días de la semana.

Nombre	Duracion	Inicio	Terminado
RADIOMETROS	123 days	**25/01/18 8:00**	**11/05/18 16:00**
1ª FASE	30 days	**25/01/18 8:00**	**20/02/18 15:00**
Elicitación	4 days	25/01/18 8:00	29/01/18 13:00
Análisis	5 days	29/01/18 13:00	1/02/18 14:00
Diseño	6 days	1/02/18 14:00	7/02/18 11:00
Implementación	7 days	7/02/18 11:00	13/02/18 15:00
Pruebas	3 days	13/02/18 15:00	15/02/18 14:00
Documentación	5 days	15/02/18 14:00	20/02/18 15:00
2ª FASE	22 days	**20/02/18 15:00**	**12/03/18 13:00**
Elicitación	3 days	20/02/18 15:00	22/02/18 14:00
Análisis	3 days	22/02/18 14:00	26/02/18 13:00
Diseño	4 days	26/02/18 13:00	28/02/18 17:00
Implementación	5 days	1/03/18 8:00	6/03/18 9:00
Pruebas	3 days	6/03/18 9:00	7/03/18 17:00
Documentación	4 days	8/03/18 8:00	12/03/18 13:00
3ª FASE	71 days	**12/03/18 13:00**	**11/05/18 16:00**
Elicitación	3 days	12/03/18 13:00	14/03/18 11:00
Formación	30 days	14/03/18 11:00	10/04/18 9:00
Análisis	4 days	10/04/18 9:00	12/04/18 14:00
Diseño	4 days	12/04/18 14:00	17/04/18 9:00
Implementación	15 days	17/04/18 9:00	30/04/18 13:00
Pruebas	3 days	30/04/18 13:00	2/05/18 11:00
Documentación	12 days	2/05/18 11:00	11/05/18 16:00

Ilustración 7: Actividades y duración estimada

A partir de la figura anterior, se pueden representar las tareas secuencialmente en forma de diagrama de barras o diagrama de Gantt. El resultado se ve a continuación en la *Ilustración 8*.

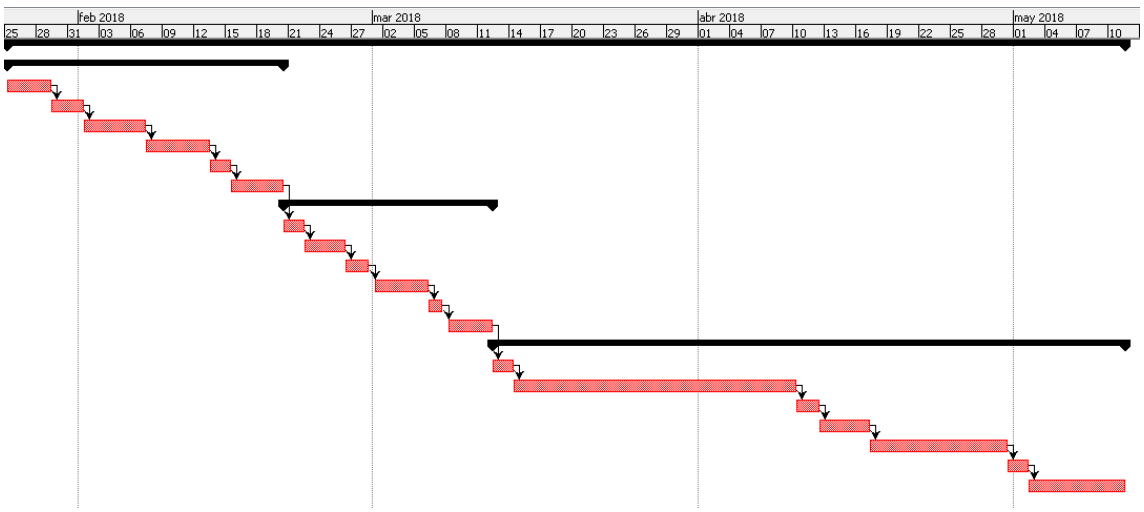


Ilustración 8: Diagrama de Gantt

La meta era presentar el proyecto antes del inicio del verano y según la estimación temporal obtenida, el proyecto estaría acabado a finales de mayo.

A continuación, se calculan las horas totales que se esperan invertir en el desarrollo del proyecto. Se tiene en cuenta que en cada una de las fases se invertirán **5 horas diarias** durante todos los días de la semana. De esta manera, la siguiente tabla refleja este número total de horas:

FASES	DÍAS INVERTIDOS	HORAS INVERTIDAS
1ª FASE	30	150
2ª FASE	22	110
3ª FASE	71	355
	123 días	**615 horas**

Tabla 2: Estimación de horas totales invertidas

2.3. Presupuesto estimado

En este momento se intenta tener una idea aproximada del presupuesto necesario para la elaboración del proyecto. Para ello se tienen en cuenta todos aquellos costes **hardware**, **software** y de **personal** que ha intervenido en el proceso de desarrollo.

Para calcular el coste software y hardware se tiene únicamente en cuenta, dentro del periodo de vida útil del mismo, el desgaste que ha sufrido durante el desarrollo del proyecto y para ello se ha tenido en cuenta la estimación temporal de **123 días**.

a) Coste hardware

- Ordenador personal → *HP Pavilion Sleekbook 15 PC*, procesador *x64 Intel Core i3-3217* 1.8GHz de dos núcleos, disco duro HDD 500 GB, memoria RAM 6 GB, sistema operativo *Windows 10*.
    - Uso: Estimación de 4 años de vida útil, por lo tanto, un % de uso del 8,4 respecto a los 123 días de duración estimada del proyecto.

$$4\ años - - - - - 100\%$$
$$0{,}336\ años - - - \ \mathbf{8,4}\%$$

- Conexión a internet: Conexión de fibra óptica de 100Mb, compañía *Vodafone*
    - Uso: Estimado 30% durante aproximadamente los 4 meses la duración estimada del proyecto (123 días).

- Otros: Hace referencia a material de oficina y al coste de la impresión y encuadernación de la documentación.

Elemento	Coste	Uso (%)	Coste final
Ordenador personal	500 €	8,4	42,6€
Conexión Internet	55 €/mes (x4 meses = 220€)	30	66€
Servidor GOA	0 €		0€
Otros	50 €	100	50€
			Total: 168,6€

Tabla 3: Presupuesto en hardware

b) Coste software

Todo el software que se va a utilizar para el desarrollo del proyecto es gratuito, a excepción de Microsoft Office 365 y GitHub.

- Microsoft Office 365 Personal: adquisición de licencia de un año por 199€. Si la duración del proyecto se estima en 123 días equivalentes a 0,336 años, el % de uso es del 8,4.
- GitHub: adquisición de plan de desarrollador para poder llevar un control de versiones del código del proyecto en repositorios privados. Se ha contratado este plan únicamente para el desarrollo del proyecto por lo que su % de uso es el 100.

Elemento	Coste	Uso (%)	Coste final
Microsoft Office 365	199 €	8.4	16,71€
Draw.io	0€		0€
ProjectLibre	0€		0€
CodeBlocks	0€		0€
Visual Studio Code	0€		0€
Notepad++	0€		0€
MySQLWorkBench	0€		0€
VirtualBox	0€		0€
GitHub	7€/mes (X4 meses = 28€)	100	28€
Putty	0€		0€
Filezilla	0€		0€
			Total: 44,71€

Tabla 4: Presupuesto en software

c) Coste de personal

En este momento se quiere obtener un presupuesto teniendo en cuenta el personal involucrado en la elaboración del proyecto. Por un lado, existe un perfil de analista que ha sido el encargado de llevar el proceso de elicitación de requisitos, de análisis, diseño y documentación. Por otro lado, existe el perfil del programador que se ha encargado de la implementación del código, las pruebas y los periodos de formación propia.

Para el sueldo por horas de cada uno de los roles se ha tenido en cuenta el salario medio establecido en España.

Rol	Tiempo	Coste	Total
Analista	1ª Fase: 20días 2ª Fase: 14días 3ª Fase: 23días Total: 57 días 57 * 5h/día = **285h**	13,9 €/h	3961,5 €
Desarrollador	1ª Fase: 10días 2ª Fase: 8días 3ª Fase: 48días Total: 66 días 66 * 5h/día = **330h**	11,4 €/h	3762 €
			Total: 7.723,5€

Tabla 5: Presupuesto en personal

d) Coste total

El presupuesto final es la suma de todos los presupuestos. Se realiza la suma del presupuesto hardware, del coste software y del coste de personal:

Presupuesto Total	Total
Presupuesto hardware	168,6€
Presupuesto software	44,71€
Presupuesto personal	7.723,5€
	7.936,81€

Tabla 6: Presupuesto total

2.4. Planificación temporal real

Han surgido diferentes cambios respecto a la planificación estimada:

- La metodología ha cambiado respecto a la que se planteaba inicialmente. Tras unas jornadas de elicitación con el cliente, se decidió plantear el proyecto como la subdivisión de dos sistemas, uno de carga y otro de visualización. Entonces, ahora en vez de 3 fases, existen dos sistemas y para el desarrollo de cada uno de ellos han intervenido varias iteraciones.
- Existe un cambio a la hora de implementar el sistema de visualización y, en vez de desarrollar la aplicación web con *Symfony2*, se ha utilizado un framework basado en JavaScript: *Angular5* para el fron-end y *NodeJS* y *ExpressJS* para el back-end.
- En los primeros periodos del proyecto, se pudieron emplear 5 horas diarias para desarrollarlo, pero más adelante esa jornada se ve reducida notablemente por la realización de las prácticas en empresa, pasando de 5 horas a 3 horas diarias a partir del 18 de febrero.

	Duración	Horas / Día	Fecha inicio	Fecha fin
Sistema de Carga				
Iteración 1 (Base de datos)	**19 días**	**95 horas**	**25/01/2018**	**28/02/2018**
Elicitación inicial • Entendimiento de los objetivos y el alcance del proyecto • Obtención de primeros requisitos	1 día	5	25/01/2018	26/01/2018
Formación MySQL • Refrescar conocimientos	1 día	5	26/01/2018	27/01/2018
Análisis de la base de datos	4 días	20	27/01/2018	01/02/2018
Diseño de la base de datos	3 días	15	01/02/2018	04/02/2018
Implementación de la base de datos	2 días	10	04/02/2018	06/02/2018
Pruebas de la base de datos	4 días	20	06/02/2018	10/02/2018
Documentación	4 días	20	10/02/2018	14/02/2018
Iteración 2 (Programa Filtrado + Librerías)	**13.5 días**	**47.5 horas**	**14/02/2018**	**28/02/2018**
Elicitación de requisitos para el filtrado	0.5 días	2.5	14/02/2018	14/02/2018

Formación C	1 días	5	15/02/2018	16/02/2018
Análisis	2 días	10	16/02/2018	18/02/2018
Diseño	2 días	6	18/02/2018	20/02/2018
Implementación	2 días	6	20/02/2018	22/02/2018
Pruebas	3 días	9	22/02/2018	25/02/2018
Documentación	3 días	9	25/02/2018	28/02/2018
Iteración 3 (Programa Cargar + Librerías)	**12.5 días**	**37.5 horas**	**28/02/18**	**13/02/2018**
Elicitación	0.5 días	1.5	28/02/2018	01/02/2018
Análisis	2 días	6	01/02/2018	03/02/2018
Diseño	2 días	6	03/02/2018	05/02/2018
Implementación	2 días	6	05/02/2018	07/02/2018
Pruebas	3 días	9	07/02/2018	10/02/2018
Documentación	3 días	9	10/02/2018	13/02/2018
Iteración 4 (Programa gestor)	**19.5 días**	**58.5 horas**	**13/02/2018**	**02/03/2018**
Elicitación	0.5 días	1.5	13/02/2018	13/02/2018
Formación Bash Script	2 días	6	13/02/2018	15/02/2018
Análisis	3 días	9	15/02/2018	18/03/2018
Diseño	3 días	9	18/03/2018	21/03/2018
Implementación	3 días	9	21/03/2018	24/03/2018
Pruebas	5 días	15	24/03/2018	29/03/2018
Documentación	3 días	9	29/03/2018	02/04/2018

Sistema de Visualización				
Iteración 1 (Front-end)	**66 días**	**198 horas**	**02/04/2018**	**06/06/2018**
Elicitación	1 día	3	02/04/2017	03/04/2018
Formación Angular5 – NodeJS – ExpressJS	40 días	120	03/04/2018	12/05/2018
Análisis	2 días	6	12/05/2018	14/05/2018
Diseño	2 días	6	14/05/2018	16/05/2018

Implementación	15 días	45	16/05/2018	31/05/2018
Pruebas	2 días	6	31/05/2018	02/06/2018
Documentación	4 días	12	02/06/2018	06/06/2018
Iteración 2 (Back-end)	**26 días**	**66 horas**	**06/06/2018**	**28/06/2018**
Análisis	2 días	6	06/06/2018	08/06/2018
Diseño	2 días	6	08/06/2018	10/06/2018
Implementación	10 días	30	10/06/2018	20/06/2018
Pruebas	2 días	6	20/06/2018	22/06/2018
Documentación	5 días	18	22/06/2018	27/06/2018

Tabla 7: Planificación temporal real

La siguiente ilustración representa en diagrama de Gantt todo este plan temporal que se ha seguido a lo largo de todo el proceso de desarrollo del proyecto. Para ello se basa en la tabla que se muestra previamente, donde cada una de las barras corresponde a una tarea de cada una de las iteraciones.

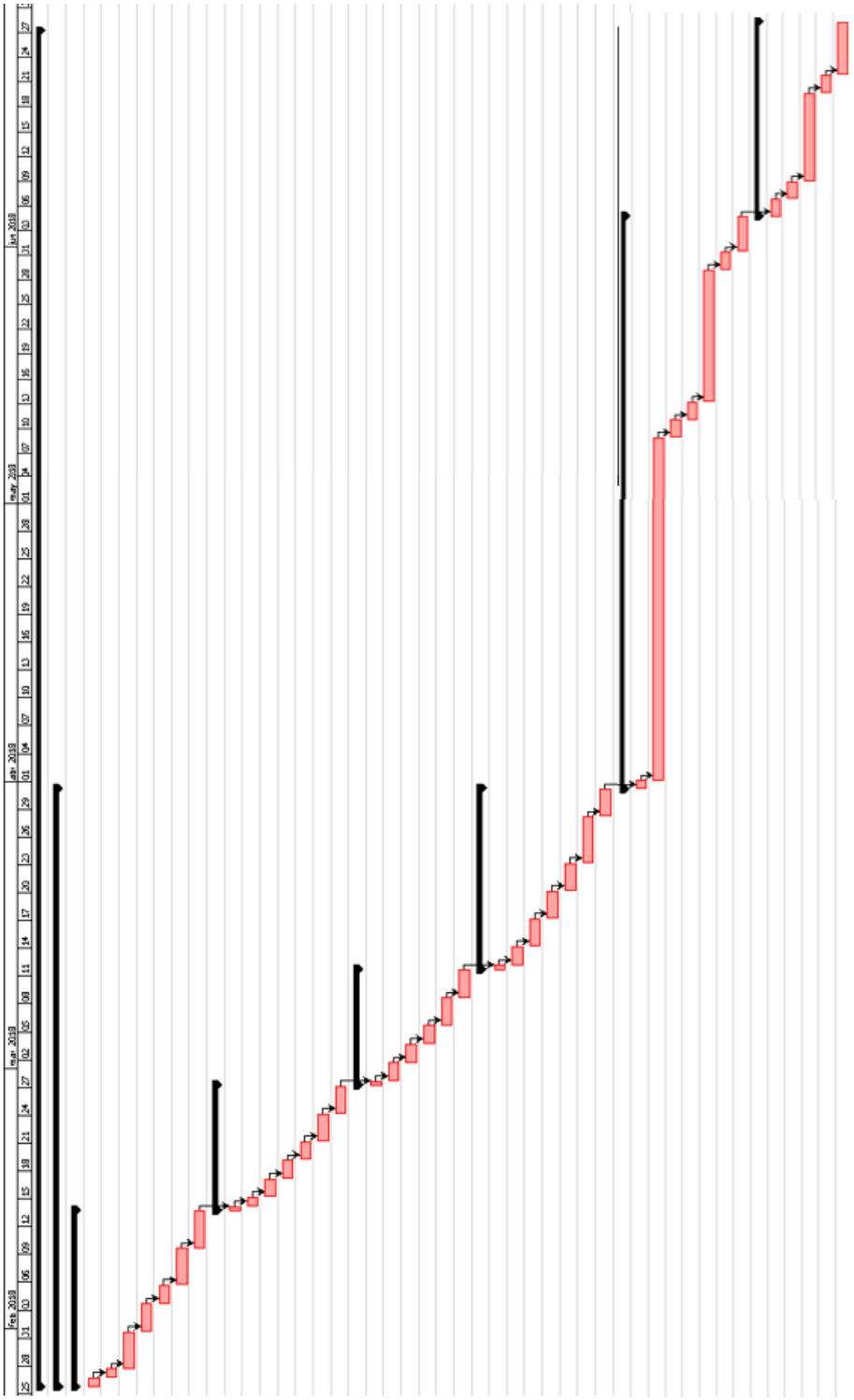


Ilustración 9: Diagrama de Gantt de la planificación real

A continuación, se muestra un resumen general de los datos de interés en cuanto al coste temporal real:

Fechas	
Comienzo del proyecto	25/01/2018
Fin del proyecto	27/06/2018

Tabla 8: Fechas reales del proyecto

Duración		
Sistema de Carga	1ª Iteración: **19 días** 2ª Iteración: **13.5 días** 3ª Iteración: **12.5 días** 4ª Iteración: **19.5 días**	Total: **64.5 días**
Sistema de Visualización	1ª Iteración: **66 días** 2ª Iteración: **26 días**	Total: **92 días**
		Total: 152,5 días

Tabla 9: Duración real del proyecto en días

Horas invertidas		
Sistema de Carga	1ª Iteración: **95 horas** 2ª Iteración: **47.5 horas** 3ª Iteración: **37.5 horas** 4ª Iteración: **58.5 horas**	Total: **238.5 horas**
Sistema de Visualización	1ª Iteración: **198 horas** 2ª Iteración: **66 horas**	Total: **264 horas**
		Total: 502,5 horas

Tabla 10: Duración real del proyecto en horas

2.5. Coste real

Se han utilizado los mismos elementos tanto software como hardware respecto al presupuesto inicial. Ahora para obtener el coste hay que incrementar el % de uso de estos ya que se ha aumentado el número de días.

a) Coste hardware

- Ordenador personal
  - Uso: Ahora son 152,5 días que suponen un % de uso del 10.44.
- Conexión a internet
  - Uso: Ahora ha pasado de 4 a 5 meses.

Elemento	Coste	Uso (%)	Coste final
Ordenador personal	500 €	10,44	52,2€
Conexión Internet	55 €/mes (x5 meses = 275€)	30	82,5€
Servidor GOA	0 €		0€
Otros	50 €	100	50€
			Total: 184,7 €

Tabla 11: Coste real del hardware

b) Coste software

Al igual que en el coste hardware, se tiene en cuenta el incremento de tiempo del proyecto:

Elemento	Coste	Uso (%)	Coste final
Microsoft Office 365	199 €	10,44	20,77€
Draw.io	0€		0€
ProjectLibre	0€		0€
CodeBlocks	0€		0€
Visual Studio Code	0€		0€
Notepad++	0€		0€
MySQLWorkBench	0€		0€
VirtualBox	0€		0€
GitHub	7€/mes (x5 meses = 35€)	100	35€
Putty	0€		0€
Filezilla	0€		0€
			Total: 55,77 €

Tabla 12: Coste real de software

c) Coste de personal

Donde sí ha variado el coste real del presupuesto inicial ha sido en el gasto de personal. En este caso, el número de horas que han sido necesarias para la finalización del proyecto han sido menos de las que se estimaron inicialmente. Por otro lado, ha intervenido un docente que ha incrementado el coste real debido a su intervención en los periodos de formación.

Rol	Tiempo	Coste	Total
Analista	Sistema de Carga 1ª Iteración: 13días -> 65h 2ª Iteración: 8.5días -> 32,5h 3ª Iteración: 7.5días -> 22.5h 4ª Iteración: 11,5 días -> 34,5h Sistema de Visualización 1ª Iteración: 49días -> 147h 2ª Iteración: 9días -> 27h Total: 57 días **Horas: 328,5h**	13,9 €/h	4.566,15 €
Desarrollador	Sistema de Carga 1ª Iteración: 6días -> 33h 2ª Iteración: 5días -> 15h 3ª Iteración: 5días -> 15h 4ª Iteración: 8días -> 24h Sistema de Visualización 1ª Iteración: 17días -> 51h 2ª Iteración: 12días -> 36h Total: 66 días **Horas: 174h**	11,4 €/h	1.983,6 €
Docente	Sistema de Visualización 1ª Iteración: 40días -> 120h Total: 40 días **Horas: 120h**	11,5 €/h	1.380 €
			Total: 7.929,75€

Tabla 13: Coste real de personal

d) Coste total

Finalmente se obtiene el coste final total, es decir, lo que realmente ha costado el desarrollo del proyecto:

Coste Total	Total
Coste hardware	184,7€
Coste software	55,77€
Coste personal	7.929,75€
	8.170,22€

Tabla 14: Coste real total

2.6. Comparativa

En este apartado se procede a hacer una comparativa entre los planes estimados y los planes reales que se han obtenido tras la finalización del proceso de desarrollo de los sistemas.

Por un lado, se comparan los plazos estimados respecto a los reales y, por otro lado, el presupuesto generado inicialmente respecto al coste real. De esta manera se aprecian las desviaciones dadas y se argumenta a qué se han podido deber estas:

Costes temporales

	Estimado	Real	Diferencia
Meses	123 jordanas ≈ 4 meses	152,5 jornadas ≈ 5 meses	**1 mes**
Horas	615 horas	502,5 horas	**112,5 horas**

Tabla 15: Comparativa de coste temporal

Inicialmente se estimó una jornada de trabajo de 5 horas diarias. Entonces, no se consideraba que la realización de prácticas curriculares en empresa iba a limitar esa jornada a 3 horas diarias. Esa es la principal razón que ha hecho que el proyecto se desvíe un mes sobre el plazo estimado. Por otro lado, como se puede apreciar en la tabla, el número de horas reales necesarias ha sido menor al número de horas estimadas. Esto se debe a dos razones:

- En un principio se decidió asignar los días necesarios teniendo en cuenta un colchón de tiempo que diera soporte ante posibles problemas que pudieran surgir en el desarrollo, es decir, se tuvo un punto de vista pesimista.
- El cambio de tecnología de *Symfony2* a *Angular5* + *NodeJS*. En la fase de creación de la parte web, se ha reducido notablemente el tiempo requerido, pasando de 555 horas (71 días x 5 horas/día) utilizando *Symfony2* a 198 horas (66 días x 3 horas/día) utilizando *Anular5* + *NodeJS*. Las causas de esto son las siguientes:
    - La presencia de un docente en el aprendizaje de *Angular5* y *NodeJS* a reducido el tiempo requerido en la formación respecto a *Symfony2*.
    - La realización de una aplicación web independiente ha supuesto un coste menor de análisis, diseño e implementación respecto a lo planteado con *Symfony2*, donde se requería crear un módulo para una pagina web existente. Esto implicaba un coste adicional para el entendimiento de la página web previa (arquitectura lógica, conexión de componentes…).

A modo resumen, si la jornada hubiera continuado siendo de 5 horas diarias, se habrían cumplido los plazos estimados inicialmente.

Costes económicos

	Presupuesto inicial	Coste real	Diferencia
Total	7.936,81€	8.170,22€	**233,41€**

Tabla 16: Comparativa de coste económico

En este caso el coste real ha sido mayor al presupuesto inicial. Aunque en el plan estimado hay más horas de trabajo que en el real, hay que tener en cuenta que se ha añadido el coste extra del docente encargado de dar la formación relativa a *Angular5* y *NodeJS*.

Capítulo 3
Documentación técnica
Sistema de Carga

3.1. Análisis

El objetivo de esta fase es ofrecer una visión del funcionamiento del sistema a nivel conceptual. Para ello se hace uso de diferentes diagramas y esquemas que faciliten su entendimiento. Inicialmente se presenta un **diagrama de contexto** que ofrece una visión del dominio del sistema de carga. También se hace referencia a la estructura de los ficheros de datos como a los detalles de los propios datos de los ficheros. Seguidamente el análisis se centra en la visión dinámica teniendo en cuenta los **actores del sistema**, **requisitos de usuario**, los **requisitos funcionales** y los **no funcionales**. Por último, la visión estática del sistema que recoge los **requisitos de información**, el **diagrama entidad relación** y el **diccionario de datos**.

3.1.1. Características del sistema

Las características del sistema son todas aquellas funcionalidades principales con las que el sistema debe contar para el cliente obtenga los resultados que espera del producto a desarrollar.

Id	Nombre	Descripción
Caract-01	**Filtrado**	
Caract-01.1	Validar Cabecera	El sistema debe ser capaz de filtrar la cabecera de los ficheros y verificar que el formato y los datos de la misma son válidos.

Tabla 17: Característica-01 Filtrado

Id	Nombre	Descripción
Caract-02	**Inserción**	
Caract-02.1	Formalizar datos	El sistema debe ser capaz de leer los datos del cuerpo del fichero y posteriormente debe insertarlos en las tablas correspondientes de la base de datos del sistema.

Tabla 18: Característica-02 Inserción

Id	Nombre	Descripción
Caract-03	**Gestión**	
Caract-03.1	Sincronía de ejecución	El sistema debe controlar la ejecución síncrona y secuencial de los programas que se encargan del procesado de ficheros de datos.
Caract-03.2	Trazabilidad	El sistema debe llevar un registro de los resultados obtenidos a procesar cada uno de los ficheros
Caract-03.2.1	Despliegue de directorios	El sistema almacena en el servidor los ficheros organizados en directorios en función de si los ficheros han sido procesados o no y, en caso de ser procesados, en función de su resultado.
Caract-03.2.2	Generar log	El sistema almacena en la base de datos un registro de los ficheros que han sido procesados de forma correcta.
Caract-03.3	Administrador	Todas aquellas funcionalidades que le permiten al administrador interactuar con el sistema.
Caract-03.3.1	Control de activación	El administrador tiene la posibilidad de activar o desactivar el estado del procesado de ficheros.
Caract-03.3.2	Automatización de tareas	El administrador establece el lapso que tiene que pasar para que se ejecute el script correspondiente al procesado de ficheros. Inicia el servicio del sistema de carga.
Caract-03.4	Adición manual de ficheros	El administrador puede añadir ficheros manualmente al directorio donde se encuentra la cola de ficheros a procesar.

Tabla 19: Característica-03 Gestión

3.1.2. Árbol de características

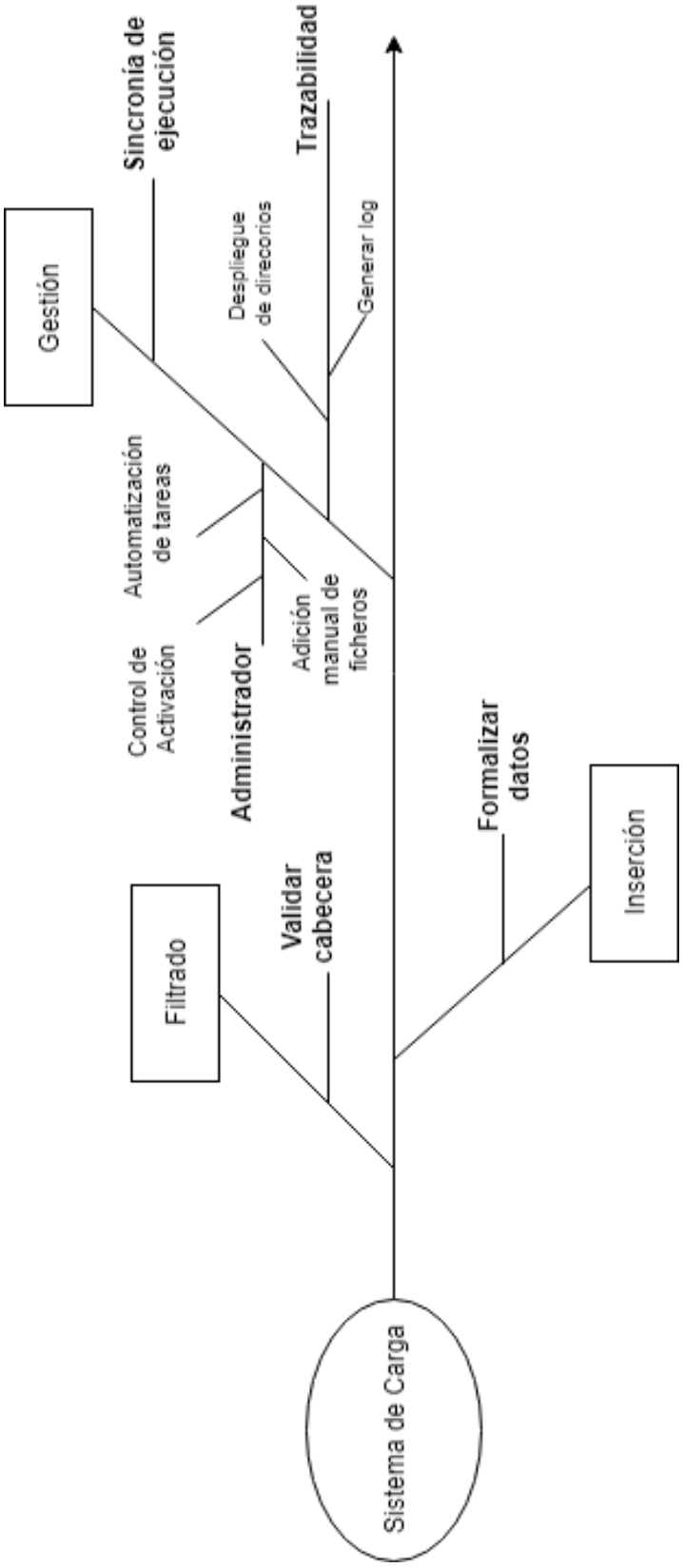


Ilustración 10: Árbol de características

3.1.3. Diagrama de contexto

El diagrama representado a continuación en la *Ilustración 11* ayuda a asimilar y entender cuáles son los límites del sistema, es decir, permite distinguir lo que es el sistema y su entorno.

En el centro del diagrama se ubica el sistema de carga de datos y como se puede observar, recibe información externa a través de centros de información, que en este caso son los radiómetros. Éstos, a través de sus registradores de datos (*dataloggers*), permiten recopilar mediciones y enviarlas al sistema en forma de fichero de texto.

Por otro lado, a la derecha aparece una entidad o sistema externo al que se entrega información una vez que nuestro sistema de carga de datos procesa los ficheros.

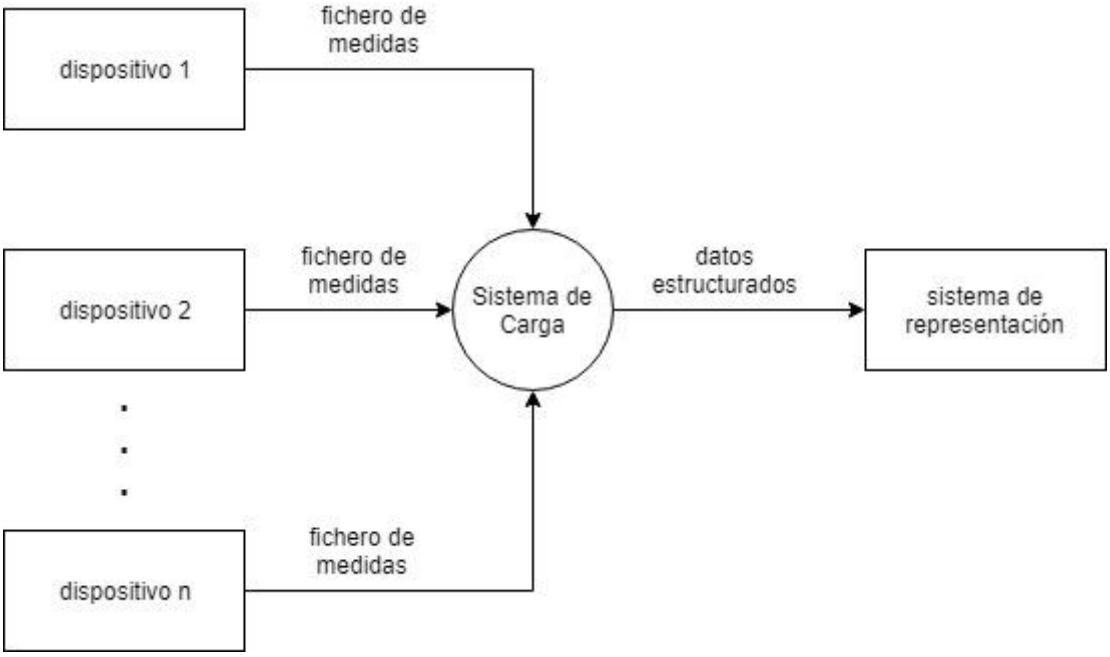


Ilustración 11: Diagrama de Contexto del sistema de carga

3.1.4. Estructura del fichero de datos

Es importante reflejar como es la estructura de los ficheros de datos que el sistema tendrá que procesar. El formato del fichero lo ha establecido previamente el *datalogger* del dispositivo, el cual ofrece los datos de la siguiente manera:

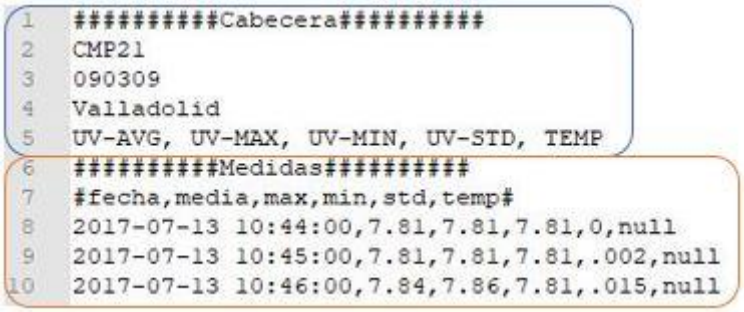


Ilustración 12: Ejemplo de fragmento de fichero de datos

A priori, se puede apreciar una primera división del contenido en **cabecera** y **medidas**. Dentro de la cabecera se pueden observar cuatro filas de la 2 a la 5: **referencia** del radiómetro, **identificador** del radiómetro, **estación** y **tipo de medición** realizada respectivamente.

En la parte inferior se sitúa el espacio que contiene las medidas. Cada una de las filas de este contenido es lo que se ha denominado **escenarios**, por lo tanto, el cuerpo del fichero está formado por escenarios y a su vez estos están formados por valores de **medidas**. El formato de cada fila/escenario será diferente en función del tipo de medición que se realice. En el caso del fragmento de ejemplo que se muestra (*Ilustración 12),* el tipo de medición es **UV** y este tipo de medición tiene el siguiente formato de escenario: **fecha**, **media**, **max**, **min**, **std** y **temp**. Cada uno de los parámetros que dan lugar al escenario tienen como el carácter "," como separador. El apartado _, se centra en detallar más en profundidad todos estos parámetros previamente nombrados.

A continuación, se presenta la estructura del fragmento del fichero anterior en esquema XML. Este tipo de esquemas se utiliza para describir la estructura y sus restricciones de una forma más precisa. De esta manera se consigue una percepción del fichero con un nivel alto de abstracción.

```
<cabecera>
    <estacion nombre="Valladolid">
        <dispositivo referencia="CMP21" id="090309"></dispositivo>
    </estacion>
</cabecera>
<mediciones tipo="UV">
    <escenario fecha="2017-07-13 10:44:00">
        <medida tipo="AVG">7.81</medida>
        <medida tipo="MAX">7.81</medida>
        <medida tipo="MIN">7.81</medida>
        <medida tipo="STD">0</medida>
        <medida tipo="TEMP"></medida>
    </escenario>
    <escenario fecha="2017-07-13 10:45:00">
        <medida tipo="AVG">7.81</medida>
        <medida tipo="MAX">7.81</medida>
        <medida tipo="MIN">7.81</medida>
        <medida tipo="STD">0.002</medida>
        <medida tipo="TEMP"></medida>
    </escenario>
    <escenario fecha="2017-07-13 10:46:00">
        <medida tipo="AVG">7.84</medida>
        <medida tipo="MAX">7.86</medida>
        <medida tipo="MIN">7.81</medida>
        <medida tipo="STD">0.015</medida>
        <medida tipo="TEMP"></medida>
    </escenario>
</mediciones>
```

Ilustración 13: Estructura XML del fragmento de fichero de datos

3.1.5. Descripción de los datos del fichero

Como se indica en la motivación del proyecto, este proyecto se centra en las radiaciones ultravioletas (**UV**), las radiaciones solares directas (**DNI**), solares globales (**SW**), Solares Difusas (**DIF**), fotosintéticamente activas (**PAR**) e Infrarrojas (**LW**). Por lo tanto, el sistema de carga se centra en estos tipos de mediciones.

Se tendrá en cuenta que de cada uno de los tipos se recogen de forma genérica los siguientes valores:

- **AVG:** Valor promedio.
- **MAX:** Valor máximo.
- **MIN:** Valor mínimo.
- **STD:** Desviación estándar.
- **TEMP:** Temperatura media.
- **TEMP2:** Temperatura adicional media.

Todos y cada uno de los valores anteriores se calculan y se obtienen por cada minuto de recogida, es decir, por cada minuto se obtiene un valor medio, un valor máximo, uno mínimo, una desviación estándar y una temperatura media. En el caso de la temperatura, existirá un valor de temperatura media adicional (TEMP2) para aquellas mediciones que recogen el tipo LW, es decir, aquellas que recogen la radiación infrarroja. Esto se debe a que, en este caso, es necesario saber la temperatura de la cúpula y del cuerpo del radiómetro para luego analizarlas y así realizar las correcciones pertinentes.

3.1.6. Perfil de los usuarios

En este apartado se describen los diferentes tipos de usuario que interactúan con el sistema y, en este caso, con este sistema de carga, sólo interactúa el administrador del propio sistema.

Usuario	Función
Administrador	Es el encargado de la activación o desactivación de la ejecución del proceso de carga de datos. Es el encargado de añadir de forma manual nuevos ficheros a la cola de procesado. Es el encargado de establecer el lapso que tiene que transcurrir para que se procesen ficheros.

Tabla 20: Usuarios del sistema de carga

3.1.7. Requisitos de usuario

A continuación, se listan todas aquellas acciones en las que el usuario interactúa con el sistema.

Id	Descripción
RU-01	El administrador puede activar la ejecución del proceso de carga de ficheros.
RU-02	El administrador puede desactivar la ejecución del proceso de carga de ficheros.
RU-03	El administrador puede poner un fichero en cola de procesado.
RU-04	El administrador puede establecer el tiempo a transcurrir para ejecutar el procesado de ficheros.
RU-05	El administrador puede consultar el valor del estado de procesado de ficheros.

Tabla 21: Requisitos de usuario del sistema de carga

3.1.8. Diagrama de casos de uso

Con el siguiente diagrama se representan las relaciones establecidas entre los diferentes casos de uso para el sistema de carga.

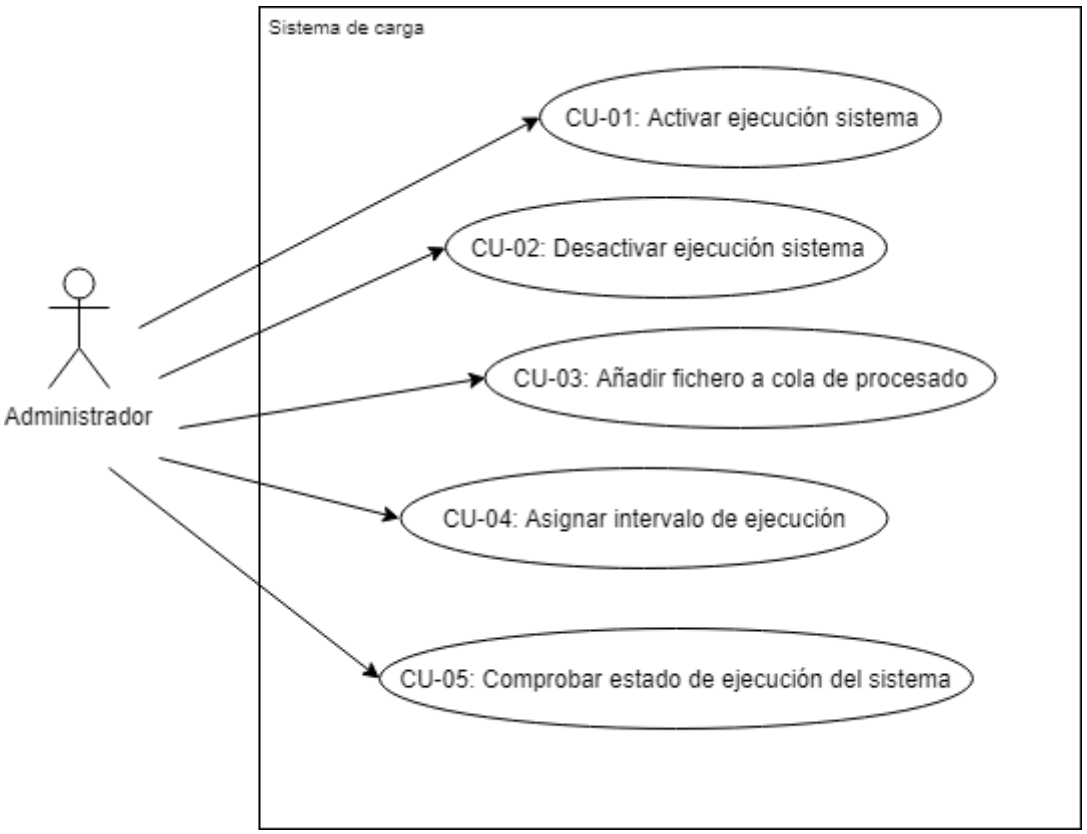


Ilustración 14: Diagrama de casos de uso del sistema de carga

El administrador tiene la posibilidad de activar o desactivar la ejecución que pone en marcha el procesado de ficheros para su posterior carga en la base de datos. Por otro lado, también puede añadir ficheros a la cola de procesado. El flujo normal de trabajo del sistema es que se vayan procesando los ficheros que van llegando a él, en concreto a un directorio donde se van encolando los estos. Pero también existe la posibilidad de que el administrador quiera añadir manualmente un fichero a esa cola. Esta situación puede darse en casos en los que un determinado fichero no pase un filtro del procesado y no se cargue en la base de datos, entonces el administrador puede corregir ese fichero y volver a ponerlo en cola de procesado.

3.1.9. Especificación de casos de uso

Id y Nombre	**CU-01: Activar ejecución sistema**
Actor	Administrador
Descripción	El administrador ejecuta el script que activa el proceso de carga de ficheros.
Precondiciones	PRE-01: El estado del proceso de carga debe estar desactivado. PRE-02: Haber iniciado sesión en el servidor como administrador.
Flujo normal	1. El administrador ejecuta el script de activación. 2. El sistema modifica el estado del proceso de carga de 0 a 1. 3. El sistema notifica al administrador de que el estado está activo.
Frecuencia	Baja: Únicamente se activará en un inicio, ya que el estado e inicializa por defecto a 0. También en situaciones excepcionales en las que el administrador haya tenido que desactivar la ejecución por algún motivo y entonces sea necesario activarlo de nuevo.

Tabla 22: Especificación del CU-01 S. Carga

Id y Nombre	**CU-02: Desactivar ejecución sistema**
Actor	Administrador
Descripción	El administrador ejecuta el script que desactiva el proceso de carga de ficheros.
Precondiciones	PRE-01: El estado del proceso de carga debe estar activado. PRE-02: Haber iniciado sesión en el servidor como administrador.
Flujo normal	1. El administrador ejecuta el script de desactivación. 2. El sistema modifica el estado del proceso de carga de 1 a 0. 3. El sistema notifica al administrador de que el estado está inactivo.
Frecuencia	Baja: Únicamente en casos excepcionales en los que el administrador vea necesaria la parada del proceso de carga de ficheros.

Tabla 23: Especificación del CU-02 S. Carga

Id y Nombre	**CU-03: Añadir fichero a la cola de procesado.**
Actor	Administrador
Descripción	El administrador mueve un fichero a la ruta o directorio donde se encolan los ficheros que se van a procesar.
Precondiciones	PRE-01: El fichero debe tener formato *.rad*. PRE-02: Haber iniciado sesión en el servidor como administrador.
Flujo normal	1. El administrador mueve un fichero al directorio del que se procesan los ficheros. 2. El sistema registra el fichero en esa ruta.
Frecuencia	Baja: Únicamente en casos excepcionales en los que el administrador vea necesario introducir un fichero manualmente al directorio correspondiente.

Tabla 24: Especificación del CU-03 S. Carga

Id y Nombre	**CU-04: Asignar intervalo de ejecución.**
Actor	Administrador
Descripción	El administrador asigna el intervalo de tiempo que tiene que transcurrir para que se vuelva a realizar la tarea de procesado de ficheros.
Precondiciones	PRE-01: Haber iniciado sesión en el servidor como administrador.
Flujo normal	1. El administrador indica el periodo en el que se ejecutará la tarea. 2. El sistema crea un servicio que se ejecuta en segundo plano de forma periódica.
Frecuencia	Baja: Se espera que el intervalo de ejecución se defina una única vez al crear el servicio.

Tabla 25: Especificación del CU-04 S. Carga

Id y Nombre	**CU-05: Comprobar estado de ejecución del sistema.**
Actor	Administrador
Descripción	El administrador consulta el estado de ejecución del procesamiento de ficheros. De esta manera sabe si está activo o inactivo.
Precondiciones	PRE-01: Haber iniciado sesión en el servidor como administrador.
Flujo normal	1. El administrador ejecuta el script que consulta el estado. 2. El sistema consulta a la base de datos. 3. El sistema devuelve un mensaje por pantalla indicando el estado del procesado.
Frecuencia	Baja: No se espera mucha interacción con la consulta del estado.

Tabla 26: Especificación del CU-05 S. Carga

3.1.10. Requisitos funcionales

En este apartado, el análisis se centra en los aspectos más dinámicos relacionados con los requisitos funcionales. Estos requisitos describen comportamientos y funciones que el sistema tiene que seguir para lograr los objetivos propuestos en este sistema de carga.

RF-ID	DESCRIPCIÓN
RF-01	El sistema debe abrir el fichero de datos en modo lectura.
RF-02	El sistema debe almacenar los valores de la cabecera en una estructura de datos.
RF-03	El sistema debe devolver el resultado de las comprobaciones de la cabecera del fichero teniendo en cuenta la base de datos.
RF-04	El sistema debe leer los valores del cuerpo del fichero
RF-05	El sistema debe insertar en la base de datos las medidas.
RF-06	El sistema debe devolver el resultado del proceso de inserción en la base de datos.
RF-07	El sistema debe consultar el valor que tiene el estado de procesado de ficheros.
RF-08	El sistema debe comprobar si hay ficheros sin procesar en el directorio.
RF-09	El sistema debe comprobar si el fichero que se quiere procesar ya existe en la base de datos.
RF-10	El sistema debe cambiar la ruta del fichero en función del resultado de su procesado.
RF-11	El sistema debe almacenar en la base de datos un registro de los resultados de los procesados de ficheros con resultados satisfactorios.
RF-12	El sistema debe ejecutar el procesado de ficheros cada 5 minutos.
RF-13	El sistema desactiva el procesado de carga modificando el estado del proceso de 1 a 0.
RF-14	El sistema activa el procesado de carga modificando el estado del proceso de 0 a 1.
RF-15	El sistema registra el fichero en la ruta.
RF-16	El sistema notifica al administrador de que el estado está activo.
RF-17	El sistema notifica al administrador de que el estado está inactivo.
RF-18	El sistema devuelve un mensaje por pantalla indicando el estado del procesado.

Tabla 27: Requisitos funcionales del S. Carga

Para hacer un seguimiento y validación de los requisitos funcionales en cuanto a los requisitos de usuario (*Apartado 3.1.6*) y a las características del sistema (*Apartado 3.1.1*), se ha elaborado la siguiente matriz de trazabilidad:

	RU-01	RU-02	RU-03	RU-04	RU-05	Carac-01 (Rama Filtrado)	Carac-02 (Rama Inserción)	Carac-03 (Rama Gestión)
RF-01						X	X	
RF-02						X	X	
RF-03						X		
RF-04							X	
RF-05							X	
RF-06							X	
RF-07					X			X
RF-08								X
RF-09								X
RF-10								X
RF-11								X
RF-12				X				X
RF-13		X						X
RF-14	X							X
RF-15			X					X
RF-16	X							
RF-17		X						
RF-18					X			

Tabla 28: Matriz de trazabilidad del S. Carga

3.1.11. Requisitos no funcionales

Los requisitos no funcionales **(RNF)** se encargan de describir las propiedades y las restricciones que el sistema debe cumplir. A un nivel inferior se pueden encontrar los Atributos de calidad **(AC)** que son aquellos que tienen en cuenta todo aquello relacionado con la disponibilidad, la usabilidad, seguridad y rendimiento del sistema.

RNF	DESCRIPCIÓN
RNF-01	El formato que debe tener el fichero de medidas estará formado por cabecera y cuerpo con las medidas. Todo ello separado por ######
RNF-02	El formato del fichero de mediciones de tipo LW tendrá un parámetro adicional (TEMP2).
RNF-03	El formato del fichero de datos debe ser *.rad*
RNF-04	El sistema procesará primero aquel fichero cuya antigüedad en el directorio de cola de procesado sea mayor.
RNF-05	Cuando se vaya a procesar un fichero con un nombre que ya se encuentra registrado en la base datos, éste sobrescribirá en la base de datos los valores del fichero previo.
RNF-06	El formato de aquellos datos que almacenen fechas será el siguiente: yyyy-mm-dd hh:mm:ss (y: año, m: mes, d: día, h: hora, m: minuto, s: segundo)

Tabla 29: Requisitos no funcionales del S. Carga

AC	DESCRIPCIÓN
AC-01	El sistema debe comparar si el valor de referencia de radiómetro de la cabecera del fichero existe en la base de datos.
AC-02	El sistema debe comparar si el valor de identificador de radiómetro de la cabecera del fichero existe en la base de datos.
AC-03	El sistema debe comparar si el nombre de la estación en la que está instalado el radiómetro, extraído de la cabecera del fichero, existe en la base de datos.
AC-04	El sistema debe comparar si el tipo de medición de la cabecera del fichero existe en la base de datos.
AC-05	El sistema debe comprobar si existe la ruta de directorios donde se almacenarán los ficheros, si no existe se crea.
AC-06	El sistema debe almacenar una copia inalterada de todos los ficheros que le llegan, ya sean o no correctos.
AC-07	El servidor estará operativo 24x7 en la recepción y procesado de ficheros
AC-08	Se garantiza la consistencia y la integridad de los datos a través del sistema gestor de la base de datos.
AC-09	La información para realizar la conexión con la base de datos (usuario, contraseña, host…) se almacena en un fichero con permisos protegidos en cuanto a la lectura de usuarios que no sean administradores o creadores de estos.

Tabla 30: Atributos de calidad del S. Carga

3.1.12. Requisitos de información

Los requisitos de información corresponden con todos aquellos datos o información que es relevante para lograr los objetivos que plantea el sistema. Se tendrá en cuenta que cada uno de los requisitos representa una entidad o una relación, de manera que cada uno de los elementos del diagrama entidad-relación concernirá con un requisito de información.

Como se puede ver en el diagrama de contexto representado en la figura_, el sistema actual depende de unas entidades externas que son los dispositivos, los cuales generan los ficheros de información con los que trabaja el sistema de carga. Puesto que todo ese contexto es adyacente al sistema, a continuación, se hace una introducción a los requisitos de información previos al sistema de carga, es decir, se presentan los pre-requisitos de información de los que se parte:

- Pre-requisitos de información

Identificador	Descripción
PRE-RI-01	El sistema almacenará la información sobre los radiómetros.
PRE-RI-02	El sistema almacenará la información sobre las instalaciones
PRE-RI-03	El sistema almacenará la información sobre los usuarios de CAELIS.
PRE-RI-04	El sistema almacenará la información sobre las estaciones.
PRE-RI-05	El sistema almacenará la información del tipo de instalaciones existentes.
PRE-RI-06	El sistema almacenará la información del coeficiente de calibración de los radiómetros.
PRE-RI-07	El sistema almacenará la información de las instalaciones asociadas a los radiómetros.
PRE-RI-08	El sistema almacenará la información de las instalaciones y su asociación a las estaciones.
PRE-RI-09	El sistema almacenará la información de las instalaciones y su asociación a los tipos de instalaciones existentes.
PRE-RI-10	El sistema almacenará la información de las actualizaciones que los usuarios hacen sobre los radiómetros.
PRE-RI-11	El sistema almacenará la información acerca de las instalaciones y el usuario que introduce la información de estas.
PRE-RI-12	El sistema almacenará la información acerca de los coeficientes de calibración en función de los radiómetros.

Tabla 31: Pre-requisitos de información del sistema

Como se puede comprobar, todos estos pre-requisitos contienen la información de los radiómetros en lo que respecta a su instalación y configuración.

- Requisitos de información del sistema

Identificador del pre-requisito	Descripción
RI-01	El sistema almacena la información de los ficheros de datos.
RI-02	El sistema almacena la información de los escenarios.
RI-03	El sistema almacena la información de las medidas.
RI-04	El sistema almacena la información de los tipos de mediciones existentes.
RI-05	El sistema almacena la información acerca de la actividad principal del sistema.
RI-06	El sistema almacena la información acerca de las sub-acciones del sistema.
RI-07	El sistema almacena el log de ejecuciones.
RI-08	El sistema almacena la información acerca de los ficheros de datos generados por los radiómetros.
RI-09	El sistema almacena la información acerca de los escenarios que forman parte de los ficheros.
RI-10	El sistema almacena la información acerca de las medidas que forman parte de los escenarios.
RI-11	El sistema almacena la información acerca de los escenarios y los posibles tipos de mediciones asociadas a ellos.
RI-12	El sistema almacena la información acerca de las sub-acciones que tienen asociadas las acciones.
RI-13	El sistema almacena la información acerca de las sub-acciones que han sido corridas en el log de ejecuciones.
RI-14	El sistema almacena la información acerca de los ficheros de datos que han sido procesados y registrados en el log de ejecuciones.

Tabla 32: Requisitos de información del sistema

3.1.13. Diagrama Entidad-Relación

Este diagrama ofrece un mejor entendimiento de la estructura, la semántica, las interrelaciones y restricciones de la base de datos, planteando una descripción estable de los contenidos de esta. Todos los requisitos de información anteriores se representan de forma expresiva y visual con el objetivo de que sean fácil de interpretar. El diagrama obtenido se ofrece a continuación:

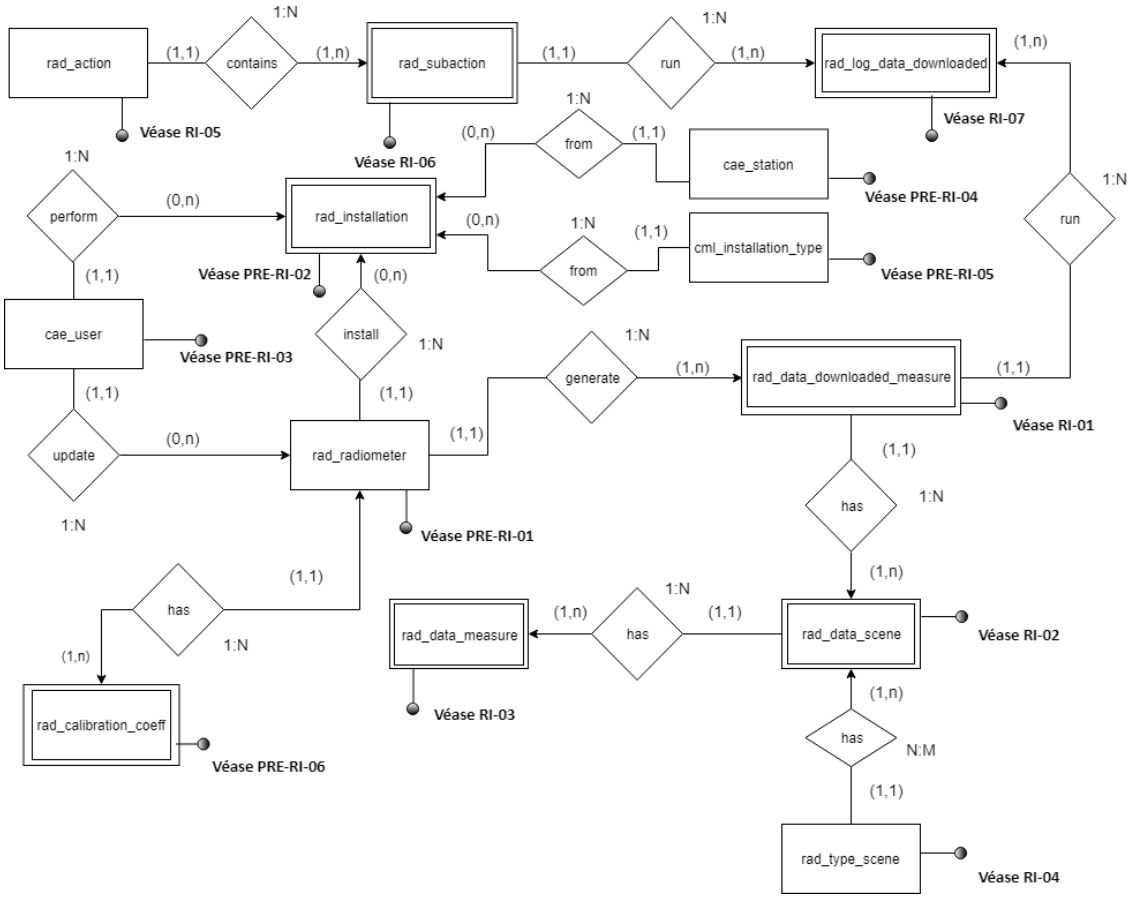


Ilustración 15: Diagrama Entidad-Relación del sistema

Existen numerosas entidades débiles (doble rectángulo) que dependen de su entidad maestra o entidad propietaria. Dicho de otra manera, una entidad débil no puede ser identificada sin su entidad fuerte. En este caso, no puede existir una instalación de radiómetro sin el propio radiómetro. Tampoco puede haber un coeficiente de calibración de un radiómetro sin el radiómetro… ídem para la entidad *rad_data_downloaded_measure* que tiene la información de los ficheros generados por los radiómetros, sin radiómetro no hay fichero de datos. A su vez, sin fichero de datos, no existen los escenarios y sin escenarios tampoco existirían las medidas. Tampoco se puede generar un log de procesado de ficheros sin ficheros o sin las propias acciones que hacen que se procesen los mismos para dar lugar al log.

3.1.14. Diccionario de datos

Los requisitos de información obtenidos previamente se pasarán a listar de forma más precisa y rigurosa para lograr un mejor rendimiento de estos y, de esta manera evitar posibles interpretaciones erróneas o ambigüedades. Se mostrarán las entidades, atributos y relaciones, haciendo hincapié en la especificación del dominio de cada atributo y en la especificación de los atributos derivados.

- Entidades

Requisito	**PRE-RI-01: El sistema almacenará la información sobre los radiómetros.**			
Entidad	rad_radiometer			
Atributos	**Nombre**	**Tipo**	**PK NULL UNIQUE**	**Descripción**
	id	String (15)	X X	Valor identificativo del radiómetro. Está formado por una secuencia de números.
	ref	String (40)		Valor de referencia del radiómetro.
	update	Timestamp		Fecha en la que se insertaron los datos del radiómetro.

Tabla 33: Diccionario de datos para PRE-RI-01

Requisito	**PRE-RI-02: El sistema almacenará la información sobre las instalaciones.**			
Entidad	rad_installation			
Atributos	**Nombre**	**Tipo**	**PK NULL UNIQUE**	**Descripción**
	date	datetime	X	Fecha de la instalación.
	rad	String (40)	X	Valor identificativo del radiómetro. Está formado por una secuencia de números
	visible	ENUM (yes, no)		Su valor determina si la instalación del dispositivo se realiza de manera que éste sea visible en la red.
	update	Timestamp		Fecha en la que se insertaron los datos del radiómetro.

Tabla 34: Diccionario de datos para PRE-RI-02

Requisito	**PRE-RI-03: El sistema almacenará la información sobre los usuarios de CAELIS.**			
Entidad	cae_user			
Atributos	**Nombre**	**Tipo**	**PK NULL UNIQUE**	**Descripción**
	nick	String(100)	X | | X	Nombre a través del cual el usuario se identifica en el sistema.

Tabla 35: Diccionario de datos para PRE-RI-03

Requisito	**PRE-RI-04: El sistema almacenará la información sobre las estaciones.**			
Entidad	cae_station			
Atributos	**Nombre**	**Tipo**	**PK NULL UNIQUE**	**Descripción**
	name	String (50)	X | | X	Nombre de la estación.
	latitude	Decimal(10,7)	| |	Latitud de la estación en grados.
	longitude	Decimal(10,7)	| |	Longitud de la estación en grados.
	elevation	mediumint (5)	| |	Longitud de la estación en grados.
	aeronet_name	String (50)	| X |	Nombre de la estación en AERONET si está dada de alta.
	rolo_name	String (50)	| X |	
	description	text	| X |	Descripción de la estación.
	stable	ENUM (yes,no)	| |	Determina si el estado de la estación es estable o no.
	last_connection	timestamp	| X |	Fecha de la última interacción con la estación.

Tabla 36: Diccionario de datos para PRE-RI-04

Requisito	**PRE-RI-05: El sistema almacenará la información del tipo de instalaciones existentes.**			
Entidad	cml_installation_type			
Atributos	**Nombre**	**Tipo**	**PK NULL UNIQUE**	**Descripción**
	name	String (50)	X | | X	Abreviatura del nombre del tipo.
	title	String (50)	| X |	Nombre del tipo de instalación.
	description	text	| X |	Descripción de lo que representan.

Tabla 37: Diccionario de datos para PRE-RI-05

Requisito	**RI-01: El sistema almacena la información de los ficheros de datos.**			
Entidad	**rad_data_downloaded_measure**			
Atributos	**Nombre**	**Tipo**	**PK NULL UNIQUE**	**Descripción**
	filename	String (200)	X | |	Nombre del fichero. Está incluida la ruta donde se encuentra almacenado.
	rad	String (50)	X | |	Valor identificativo del radiómetro. Está formado por una secuencia de números.
	fmt	Varchar(50)	| X |	Fecha del primer escenario del fichero. El formato es YYYY-mm-dd hh:mm:ss.
	lmt	Varchar(50)	| X |	Fecha del último escenario del fichero.
	total_scenes	Int (11)	| X |	Número de escenarios que tiene el fichero.

Tabla 38: Diccionario de datos para RI-01

Requisito	**RI-02: El sistema almacena la información de los escenarios.**			
Entidad	**rad_data_scene**			
Atributos	**Nombre**	**Tipo**	**PK NULL UNIQUE**	**Descripción**
	date	Varchar(50)	X	Fecha en la que el radiómetro recogió el escenario.
	filename	String (200)	X	Nombre del fichero. Está incluida la ruta donde se encuentra almacenado.
	rad	String (50)	X	Valor identificativo del radiómetro. Está formado por una secuencia de números.

Tabla 39: Diccionario de datos para RI-02

Requisito	**RI-03: El sistema almacena la información de las medidas.**			
Entidad	**rad_data_measure**			
Atributos	**Nombre**	**Tipo**	**PK NULL UNIQUE**	**Descripción**
	Key	ENUM (AVG, MAX, MIN, STD, TEMP, TEMP2)	X	Tipo de medida recogida por el radiómetro. Puede ser de tipo AVG, MAX, MIN, STD, TEMP, TEMP2.
	date	Varchar(50)	X	Fecha en la que el radiómetro recogió el escenario.
	filename	String (200)	X	Nombre del fichero. Está incluida la ruta donde se encuentra almacenado.
	rad	String (50)	X	Valor identificativo del radiómetro. Está formado por una secuencia de números.
	Value	float		Valor de la medición.

Tabla 40: Diccionario de datos para RI-03

Requisito	**RI-04: El sistema almacena la información de los tipos de mediciones existentes.**			
Entidad	**rad_type_scene**			
Atributos	**Nombre**	**Tipo**	**PK NULL UNIQUE**	**Descripción**
	name	String (10)	X	Abreviatura del tipo de medición.
	title	String (50)	X	Nombre completo del tipo de medición.
	description	String (500)	X	Descripción del tipo de medición.

Tabla 41: Diccionario de datos para RI-04

Requisito	**RI-05**: **El sistema almacena la información acerca de la actividad principal del sistema**.			
Entidad	**rad_action**			
Atributos	**Nombre**	**Tipo**	**PK NULL UNIQUE**	**Descripción**
	name	String (10)	X	Nombre de la acción.
	status	ENUM (0, 1)		Estado de la acción. Valor 1 activo, valor 0 inactivo.
	description	text	X	Descripción de la acción.

Tabla 42: Diccionario de datos para RI-05

Requisito	**RI-06: El sistema almacena la información acerca de las sub actividades del sistema.**			
Entidad	**rad_subaction**			
Atributos	**Nombre**	**Tipo**	**PK NULL UNIQUE**	**Descripción**
	Name	String (20)	X	Nombre de la sub-acción.
	Action_name	String (10)	X	Nombre de la acción.
	description	text	X	Descripción de la sub-acción.

Tabla 43: Diccionario de datos para RI-06

Requisito	**RI-07: El sistema almacena el log de ejecuciones.**					
Entidad	**rad_log_data_downloaded**					
Atributos	**Nombre**	**Tipo**	**PK**	**NULL**	**UNIQUE**	**Descripción**
	date	Varchar(50)	X			Nombre del tipo de medición.
	filename	String (200)	X			Nombre del fichero. Está incluida la ruta donde se encuentra almacenado.
	rad	String (50)	X			Valor identificativo del radiómetro. Está formado por una secuencia de números.
	acc	String (20)	X			Nombre de la acción.
	sub	String (10)	X			Nombre de la sub-acción.

Tabla 44: Diccionario de datos para RI-07

- Relaciones

Requisito	**PRE-RI-07: El sistema almacenará la información de las instalaciones asociadas a los radiómetros**.	
Descripción	Relación que enlaza los radiómetros y la instalación.	
Relación	install	
Cardinalidad	1:N	Ninguna o muchas instalaciones pueden asociarse a un radiómetro, sin embargo, una instalación concreta no puede tener asociada varios radiómetros.
Entidades relacionadas	• rad_radiometer • rad_installation	

Tabla 45: Diccionario de datos PRE-RI-07

Requisito	**PRE-RI-08: El sistema almacenará la información de las instalaciones y su asociación a las estaciones.**	
Descripción	Relación que enlaza las instalaciones con las estaciones en las que pueden hacerse las instalaciones.	
Relación	from	
Cardinalidad	1:N	Una misma instalación no puede asociarse a mas de una estación, sin embargo, a una estación se la pueden asociar ninguna o varias instalaciones.
Entidades relacionadas	• rad_installation • rad_station	

Tabla 46: Diccionario de datos PRE-RI-08

Requisito	**PRE-RI-09:** El sistema almacenará la información de las instalaciones y su asociación a los tipos de instalaciones existentes.	
Descripción	Relación que enlaza las instalaciones realizadas con el tipo de instalaciones que existe.	
Relación	from	
Cardinalidad	1:N	Una misma instalación no puede asociarse a más de un tipo de instalación, por otro lado, un mismo tipo de instalación puede estar asociado a ninguna o varias intalaciones.
Entidades relacionadas	• rad_installation • cml_installation_type	

Tabla 47: Diccionario de datos PRE-RI-09

Requisito	**PRE-RI-10: El sistema almacenará la información de las actualizaciones que los usuarios hacen sobre los radiómetros.**	
Descripción	Relación que enlaza un radiómetro a un determinado usuario que actualiza la información del mismo en la base de datos.	
Relación	update	
Cardinalidad	1:N	Ninguno o varios radiómetros pueden ser actualizados por un mismo usuario, sin embargo, no puede haber varios usuarios que hayan actualizado la información de un mismo radiómetro.
Entidades relacionadas	• rad_radiometer • cae_user	

Tabla 48: Diccionario de datos PRE-RI-10

Requisito	**PRE-RI-11: El sistema almacenará la información acerca de las instalaciones y el usuario que introduce la información de estas.**	
Descripción	Relación que enlaza una instalación con un usuario que la realiza y documenta.	
Relación	perform	
Cardinalidad	1:N	Ninguna o varias instalaciones pueden ser realizadas por un mismo usuario, sin embargo, no puede haber varios usuarios que hayan realizado una misma instalación.
Entidades relacionadas	• rad_installation • cae_user	

Tabla 49: Diccionario de datos PRE-RI-11

Requisito	**PRE-RI-12: El sistema almacenará la información acerca de los coeficientes de calibración en función de los radiómetros.**	
Relación	has	
Cardinalidad	1:N	Uno o varios coeficientes de calibración están relacionados con un radiómetro, sin embargo, no puede haber varios radiómetros con un mismo coeficiente de calibración.
Entidades relacionadas	• rad_radiometer • rad_calibration_coeff	

Tabla 50: Diccionario de datos PRE-RI-12

Requisito	**RI-08: El sistema almacena la información acerca de los ficheros de datos generados por los radiómetros.**	
Relación	generate	
Cardinalidad	1:N	Ninguno o varios ficheros de datos pueden ser generados por un mismo radiómetro, sin embargo, no puede haber varios radiómetros que hayan generado un mismo fichero de datos.
Entidades relacionadas	• rad_radiometer • rad_data_downloaded_measure	

Tabla 51: Diccionario de datos RI-08

Requisito	**RI-09: El sistema almacena la información acerca de los escenarios que forman parte de los ficheros.**	
Relación	has	
Cardinalidad	1:N	Uno o varios escenarios pueden estar dentro de un mismo fichero de datos, sin embargo, no puede haber varios ficheros que contengan un mismo escenario.
Entidades relacionadas	• rad_data_downloaded_measure • rad_data_scene	

Tabla 52: Diccionario de datos RI-09

Requisito	**RI-10: El sistema almacena la información acerca de las medidas que forman parte de los escenarios.**	
Relación	has	
Cardinalidad	1:N	Una o varias medidas pueden pertenecer a un mismo escenario, sin embargo, no puede haber varios escenarios que contengan una misma medida.
Entidades relacionadas	• rad_data_scene • rad_data_measure	

Tabla 53: Diccionario de datos RI-10

Requisito	**RI-11: El sistema almacena la información acerca de los escenarios y los posibles tipos de mediciones asociadas a ellos.**	
Relación	belong to	
Cardinalidad	1:N	Ninguno o varios escenarios pueden estar asociados a un tipo de medición, sin embargo, no puede haber varios tipos de mediciones asociados a un mismo escenario.
Entidades relacionadas	• rad_data_scene • type_scene	

Tabla 54: Diccionario de datos RI-11

Requisito	**RI-12: El sistema almacena la información acerca de las sub-acciones que tienen asociadas las acciones.**	
Relación	contains	
Cardinalidad	1:N	Una o varias sub-acciones están asociadas a una acción, sin embargo, no puede haber varias acciones que contengan una misma sub-acción concreta.
Entidades relacionadas	• action • subaction	

Tabla 55: Diccionario de datos RI-12

Requisito	**RI-13: El sistema almacena la información acerca de las sub-acciones que han sido corridas en el log de ejecuciones.**	
Relación	run	
Cardinalidad	1:N	Uno o varios logs recogen la ejecución de una sub-acción concreta, sin embargo, no puede haber varias sub-acciones para un mismo registro de log.
Entidades relacionadas	• subaction • log_data_downloaded	

Tabla 56: Diccionario de datos RI-13

Requisito	**RI-14: El sistema almacena la información acerca de los ficheros de datos que han sido procesados y registrados en el log de ejecuciones.**	
Relación	collected in	
Cardinalidad	1:N	Uno o varios registros de log pueden estar asociados a un mismo fichero, sin embargo, no puede haber varios ficheros de datos que tengan asociados un mismo registro log concreto.
Entidades relacionadas	• log_data_downloaded • rad_data_downloaded_measure	

Tabla 57: Diccionario de datos RI-14

3.2. Diseño

3.4.1. Elección del SGBD

En este caso no se utiliza una base de datos nueva, sino que se incorporan tablas a una ya existente (CAELIS). Como es lógico se va a utilizar el mismo Sistema Gestor de Bases de Datos que utiliza CAELIS, **MySQL**. Esta es una de las especificaciones que solicitó el cliente al inicio del proyecto. De esta manera, la base de datos queda organizada en tablas y contamos con integridad referencial entre ellas y disparadores que ejecutan acciones a partir de eventos (*Triggers*).

3.4.2. Diseño lógico de la base de datos

El diseño lógico de la base de datos adaptará el modelo conceptual, obtenido en el análisis, a las características del modelo soportado por el SGBD utilizado para su implementación. Por lo tanto, en esta fase, se transforma el modelo Entidad-Relación previo en un **modelo relacional** teniendo en cuenta unas **decisiones de transformación** para adaptar el modelo conceptual al SGBD seleccionado.

En este punto están presentes también las **restricciones de integridad**, haciendo referencia al diccionario de datos obtenido en el análisis, y al desarrollo de los **Triggers** necesarios que permiten controlar aquellos aspectos relacionados con el dominio y las reglas de negocio.

a) Modelo Relacional

La transformación del modelo E-R al relacional es una actividad que se lleva a cabo sin tener en cuenta el tipo de SGBD que se vaya a utilizar. La idea fundamental se basa en el uso de relaciones consideradas como tablas compuestas por filas y columnas.

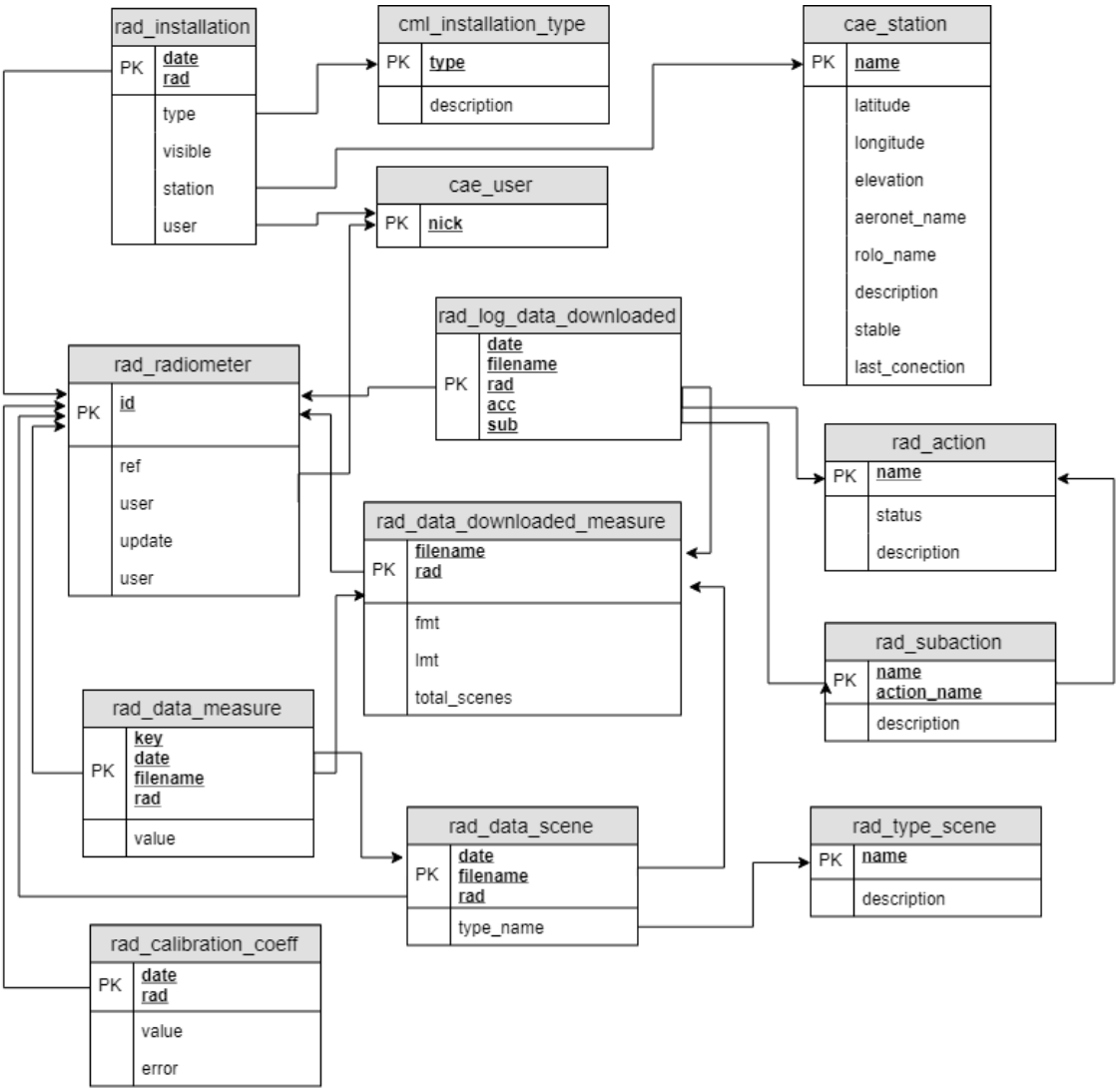


Ilustración 16: Modelo relacional de la base de datos

b) Decisiones de transformación

A continuación, se pasa a explicar cómo se ha ajustado el modelo relacional a las propiedades específicas del SGBD seleccionado, en este caso a MySQL. En la *Ilustración 16,* se puede observar el modelo resultante a partir de las siguientes decisiones de transformación partiendo del diseño conceptual representado por el modelo Entidad-Relación.

- Para cada Entidad se ha creado una nueva tabla y cada atributo dará lugar a las columnas de esta, teniendo en cuenta que, el que actuaba como identificador único en el modelo conceptual, pasa a ser clave primaria en el modelo lógico para cada tabla generada.
- Las relaciones de tipo 1:N pueden ser de dos tipos en función de la entidad que forme la misma:

- Relación 1:N entre entidades independientes (no débiles): dan lugar a la propagación de la clave primaria de la entidad correspondiente a la cardinalidad 1, a la tabla de la entidad correspondiente a la cardinalidad N. La tabla receptora lo amacena como clave foránea.
- Relación 1:N habiendo entidades débiles: es similar al caso anterior pero en esta situación, la tabla receptora almacena la clave primaria, relacionada con la entidad padre, como clave primaria en la propia tabla que está relacionada con la entidad débil.

Hay que tener en cuenta que no se ha seguido estrictamente esta normalización en todos los casos puesto que se ha contemplado una excepción:

- En el caso de la tabla *rad_data_measure*, la columna que almacena el valor 'date' tendría que hacer referencia (según la normalización indicada anteriormente) a la tabla *rad_data_scene* y en este caso no lo hace. Más adelante, al exponer cómo se consigue la integridad de los datos, se explica de qué manera se soluciona esta situación.

c) Integridad de los datos

La consistencia de los datos es uno de los principales requisitos que la base de datos debe cumplir, de tal manera que se garantice que todos los datos son congruentes y no tengan interpretaciones adversas. Para que esto sea posible, se contemplan diferentes restricciones de integridad.

Hasta ahora, el **diccionario de datos** ha establecido diferentes restricciones, así como la especificación obligatoria de los tipos de datos (`VARCHAR`, `FLOAT`, `ENUM`…), la acotación de los posibles valores válidos que un campo puede tener, es decir, las condiciones de dominio que se deben respetar. Por otro lado, el diccionario también recoge restricciones en cuanto a la integridad a nivel de entidad. Por ejemplo, se establece que para cada clave primaria (PK) solo se le pueden asociar valores únicos (`UNIQUE`) y no nulos (`NOT NULL`).

Es en el diseño lógico donde entran en juego las restricciones de integridad referencial. El Lenguaje de Definición de Datos (**DDL**), es donde se define la estructura de las tablas de la base de datos. Este DDL resultante a partir del modelo lógico, debe respetar las diferentes correspondencias con los valores permitidos de cada clave candidata en la tabla referenciada. Se presta especial atención a las referencias entre tablas, así como a las operaciones de borrado y modificación que guardan relación con claves foráneas. El recurso DDL se encuentra en el DVD proporcionado en la presentación del proyecto.

En la integridad referencial se puede observar que todas las operaciones de borrado o modificado, que afectan a alguna clave foránea, se realizan en `CASCADE` desencadenando modificaciones y borrados en cascada. En caso de que se dé una modificación del valor de un campo de la tabla referente, automáticamente cambiará el valor de la clave foránea de los registros relacionados en la tabla que hace referencia. Lo mismo en el caso del borrado. Cuando se elimina un registro de la tabla referente, de

forma automática se eliminan también los registros que la hacen referencia en otras tablas. De esta manera se verifica el cumplimiento de la integridad referencial.

Cabe destacar que, además de las restricciones de integridad mencionadas previamente, se recurre al uso de **Triggers** o disparadores que actuarán en el caso excepcional del borrado de un registro perteneciente a la tabla *rad_data_scene*. El código fuente también se encuentra en el DVD proporcionado.

A continuación, se detalla cual es la función que realiza el Trigger desarrollado y la necesidad del uso de este:

- **Trigger 1 - rad_delete_scene_measures**

Este disparador verifica que se cumpla la integridad referencial cuando se realiza una operación de borrado de un registro dentro de la tabla *rad_data_scene*. De esta manera, antes de borrar un determinado registro identificado por la fecha, el nombre del fichero al que pertenece y el radiómetro que lo generó, se borran todos los registros existentes en la tabla *rad_data_measure* cuyos identificadores sean igual a los del registro *rad_data_scene* que se quiere borrar.

A pesar de que la operación de borrado directo sobre la tabla *rad_data_scene* no está contemplada, como medida de prevención de la integridad, se ha decidido crear el Trigger, para el caso en el que el administrador de la base de datos decida borrar un determinado escenario de un determinado fichero en la base de datos. Dicha situación será excepcional puesto que el proceso normal en ese caso sería modificar el fichero real, con formato *.rad*, eliminando la línea correspondiente a dicho escenario y volver a procesarlo para que se actualice en la base de datos.

3.4.3. Disposición de directorios

Todos los recursos referentes a este sistema de carga se encuentran alojados en el servidor, en concreto en un directorio específico llamado **Caelis_radiometros**. En la *Ilustración 17* se puede observar cómo es la disposición general de los recursos en el servidor.

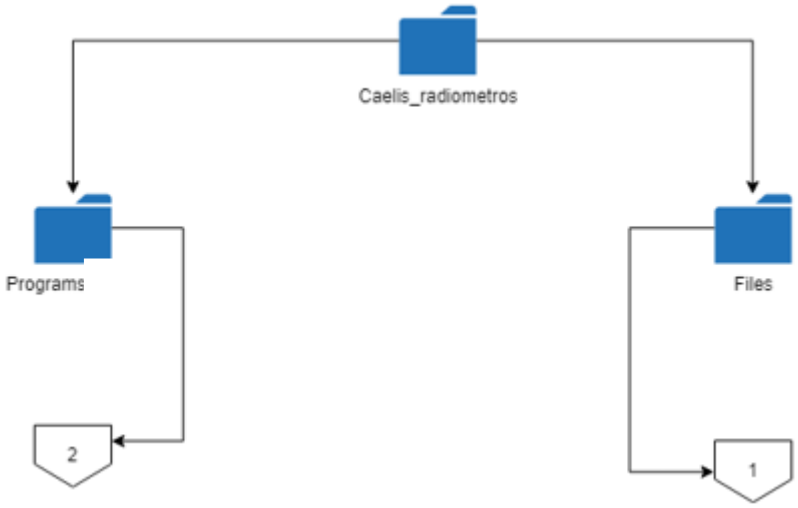


Ilustración 17: Estructura general de directorios

A continuación, se describe cada uno de los recursos que se aprecian en la *Ilustración 17*.

- **Caelis_radiometros**

Directorio raíz del proyecto. En él están contenidos los directorios ProgramsServer y Files.

- **Programs**

Directorio formado por los programas que dan la funcionalidad al sistema. Está formado por los directorios *filter*, *insert*, *sources_rad_manage* y el script *rad_manage.sh*. Este último se encarga de llevar el procesado de ficheros, gestionando la ejecución secuencial del filtrado y la inserción de ficheros en la base de datos. También se encarga de gestionar la ruta en la que se irán alojando los distintos ficheros que llegan al servidor en función del resultado de su procesado.

- **Files**

Directorio en el que se alojan los ficheros de datos formados por la información recogida por cada uno de los radiómetros. Está formada por los directorios *rad_achieve* y *rad_upload*.

A continuación, se representa en la *Ilustración 18* el directorio **Files** más en detalle:

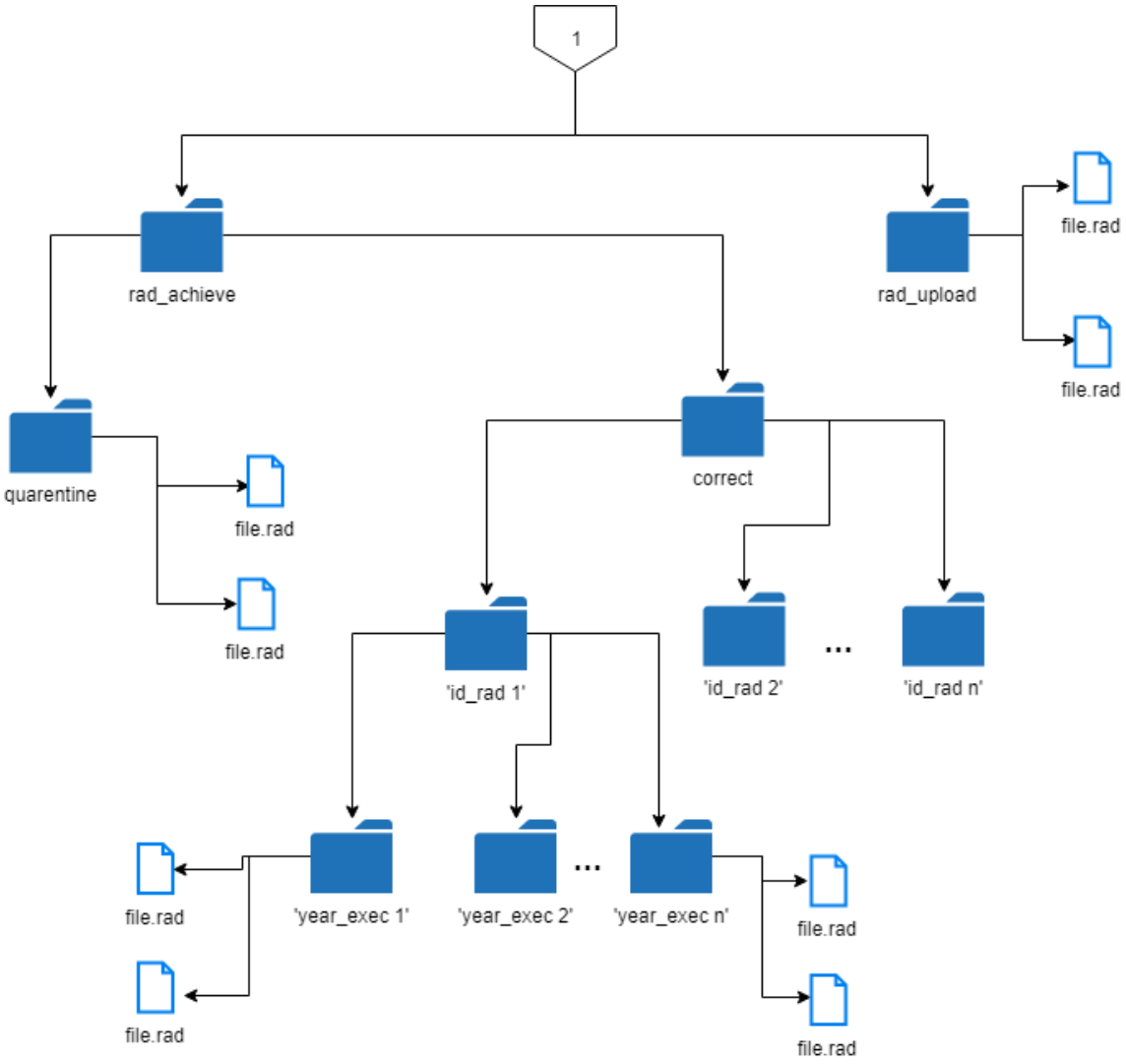


Ilustración 18: Estructura del directorio FIles

Seguidamente se describen los recursos que lo forman:

- **rad_upload**

Directorio en el que se alojan los ficheros que llegan desde los radiómetros o que son añadidos de forma manual por el administrador del sistema. Los ficheros alojados en el mismo aún no han sido procesados.

- **rad_achieve**

Directorio en el que se alojan los ficheros que ya han sido procesados y según su resultado se dirigirán al directorio *correct* o *quarentine*.

- **quarentine**

Directorio que almacena los ficheros cuyo resultado de su procesado ha sido erróneo.

- **correct**

Directorio que almacena los ficheros cuyo resultado de su procesado ha sido satisfactorio. Estos ficheros se guardan en el directorio que tiene como nombre el valor del identificador del radiómetro que generó ese fichero.

- **'id_rad'**

Directorio que almacena los ficheros que provienen concretamente de un determinado radiómetro con identificador 'id_rad'. Por ejemplo, el directorio llamado *090309* estará formado por todos aquellos ficheros generados por el radiómetro 090309 cuyo resultado de procesado ha sido satisfactorio.

- **'year_exec'**

Directorio que almacena los ficheros que han sido procesados en un año concreto. Por ejemplo, el directorio llamado *2018* estará formado por todos aquellos ficheros generados por un determinado radiómetro cuyo procesado se ha realizado durante el año 2018 y su resultado es satisfactorio.

A continuación, se representa en la *Ilustración 19* el directorio **Programs** más en detalle:

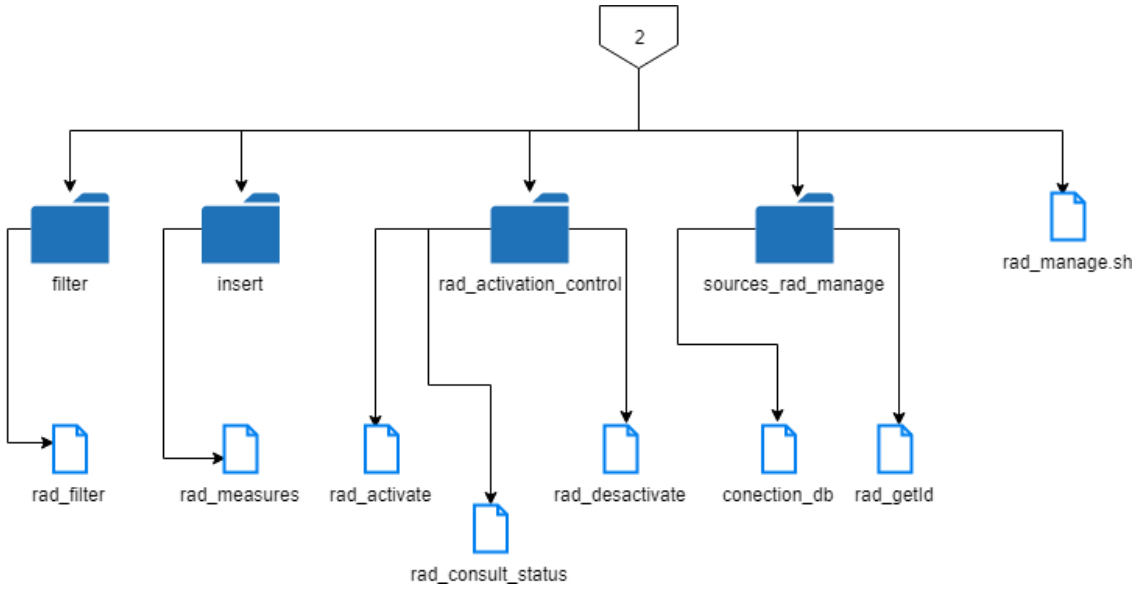


Ilustración 19: Estructura del directorio Programs

Seguidamente se describen los recursos que lo forman:

- **filter**

Directorio que contiene el script *rad_filter.c* que se encarga de filtrar los ficheros para verificar si las cabeceras de estos son válidas o no.

- **insert**

Directorio que contiene el script *rad_measures.c* que se encarga de realizar las inserciones de los datos en la base de datos.

- **sources_rad_manage**

Directorio formado por los scripts *conection_db.sh* y *rad_getId*.c. El primero actúa como librería que contiene los datos necesarios para realizar la conexión con la base de datos. El segundo se encarga de obtener el valor de los identificadores de los radiómetros.

- **rad_activation_control**

Directorio formado por los scripts *rad_activate.sh*, *rad_desactivate.sh* y *rad_consult_state.sh*. Básicamente permiten al administrador activar, desactivar o consultar el estado del procesado y carga de ficheros.

3.4.4. Diagramas de actividades

En este apartado se presentan los diagramas de actividad de todo el sistema de carga. A partir de estos diagramas se puede describir de forma más específica la funcionalidad del sistema, así como su flujo de comportamiento.

Se realizan diagramas a partir de las características principales del sistema, así como la característica de filtrado, inserción y gestión.

- **Característica de filtrado**

A continuación, en la *Ilustración 20*, se representa el comportamiento del sistema en relación con la validación de ficheros dependiendo de su cabecera.

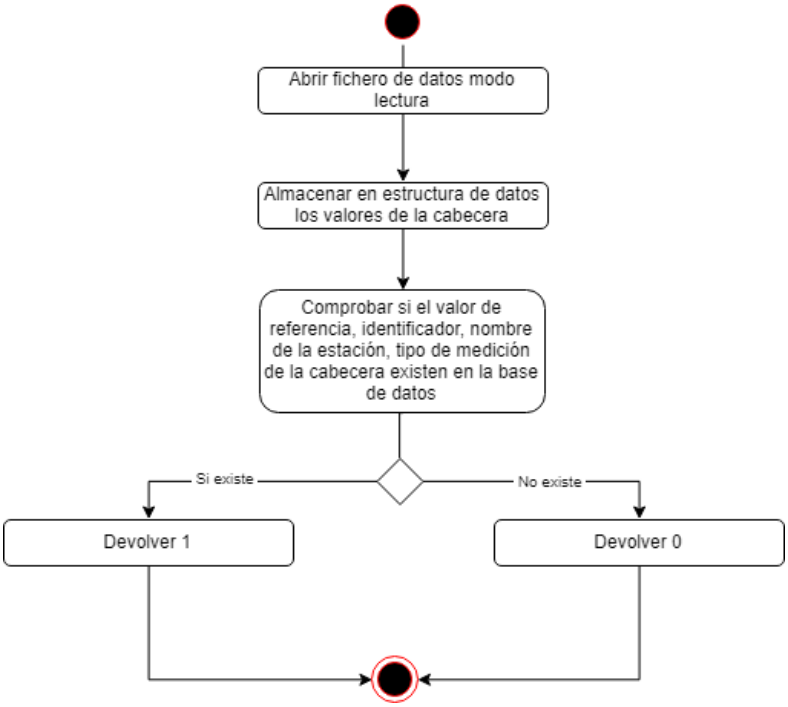


Ilustración 20: Diagrama de actividades del proceso de filtrado

- **Característica de inserción**

En *Ilustración 21* se representa el comportamiento del sistema en relación con el proceso de inserción de la información de los ficheros en la base de datos.

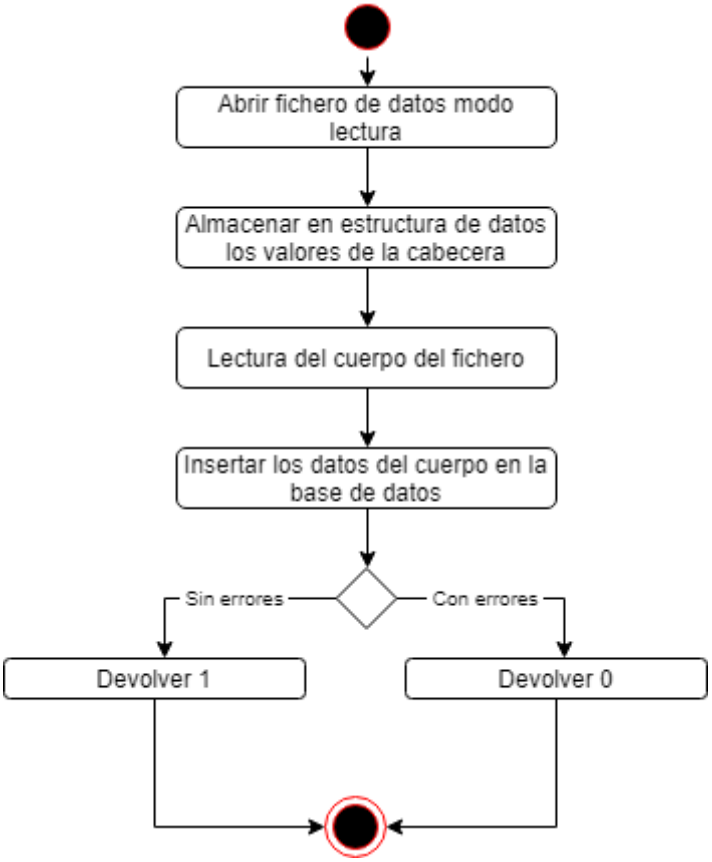


Ilustración 21: Diagrama de actividades del proceso de inserción

- **Característica de gestión**

En este caso la característica de gestión se subdivide en varias subcaracterísticas: Tareas de administrador y sincronía de ejecución, trazabilidad

- **Tareas de administrador**

Se tienen en cuenta todos aquellos aspectos relacionados con la intervención del administrador en el sistema. En este caso lo referente al control de activación de procesado, la adición manual de ficheros al directorio de cola del servidor y la automatización de tareas.

- Activación del procesado

En la *Ilustración 22* se representa el flujo de actividades cuanto el administrador decide activar la ejecución del procesado de ficheros.

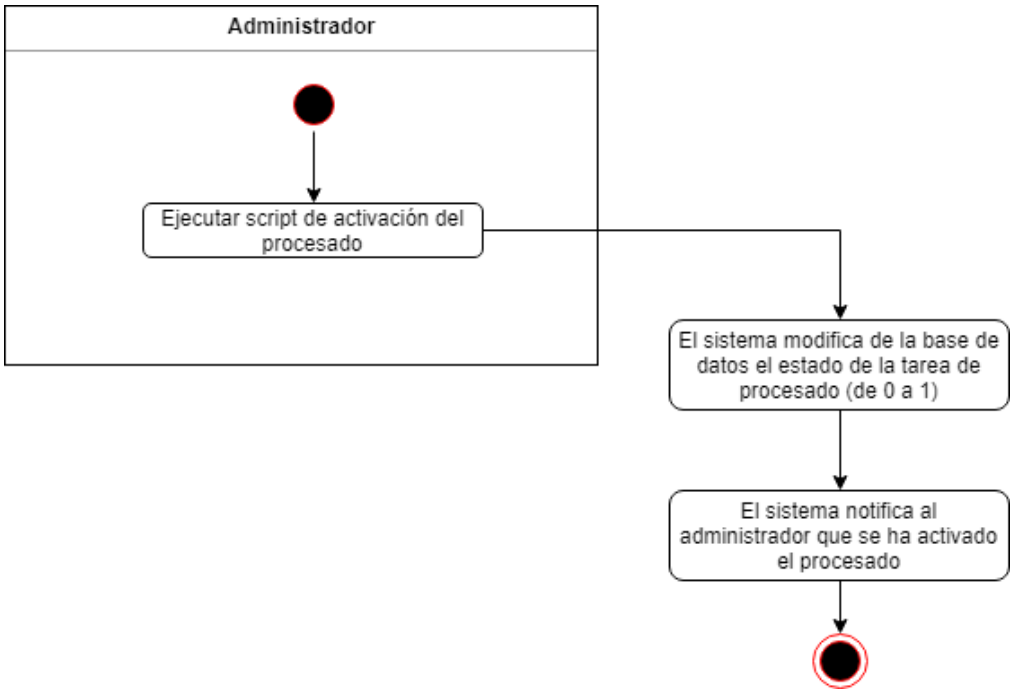


Ilustración 22: Diagrama de actividades de gestión (activar procesado)

- Desactivación del procesado

En la *Ilustración 23* se representa el flujo de actividades cuanto el administrador decide detener la ejecución del procesado de ficheros.

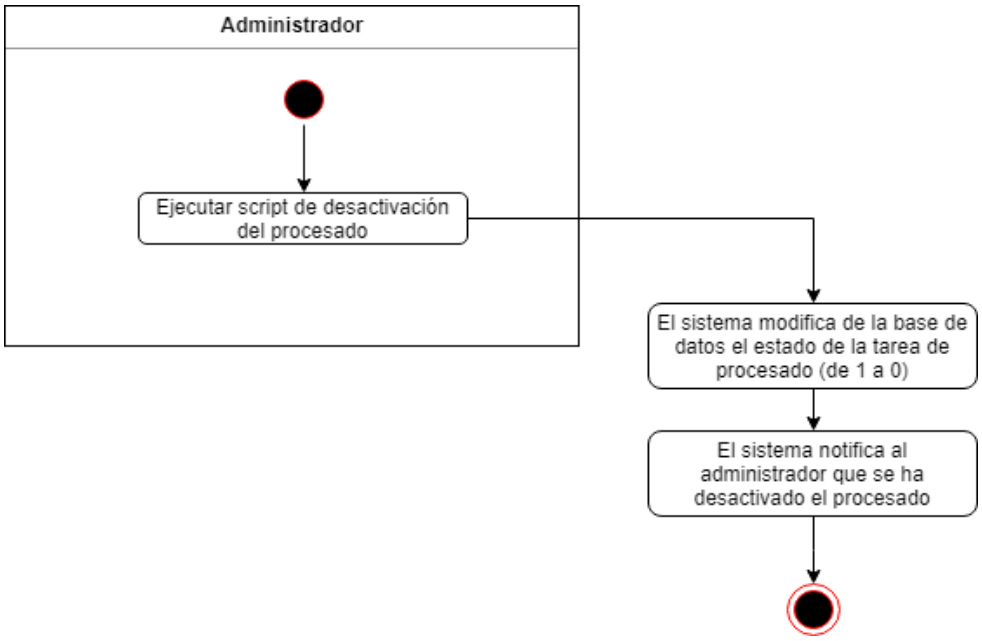


Ilustración 23: Diagrama de actividades de gestión (desactivar procesado)

- Comprobar estado de procesado

En la *Ilustración 24* se representa el flujo de actividades cuando el administrador decide consultar el estado de ejecución del procesado de ficheros.

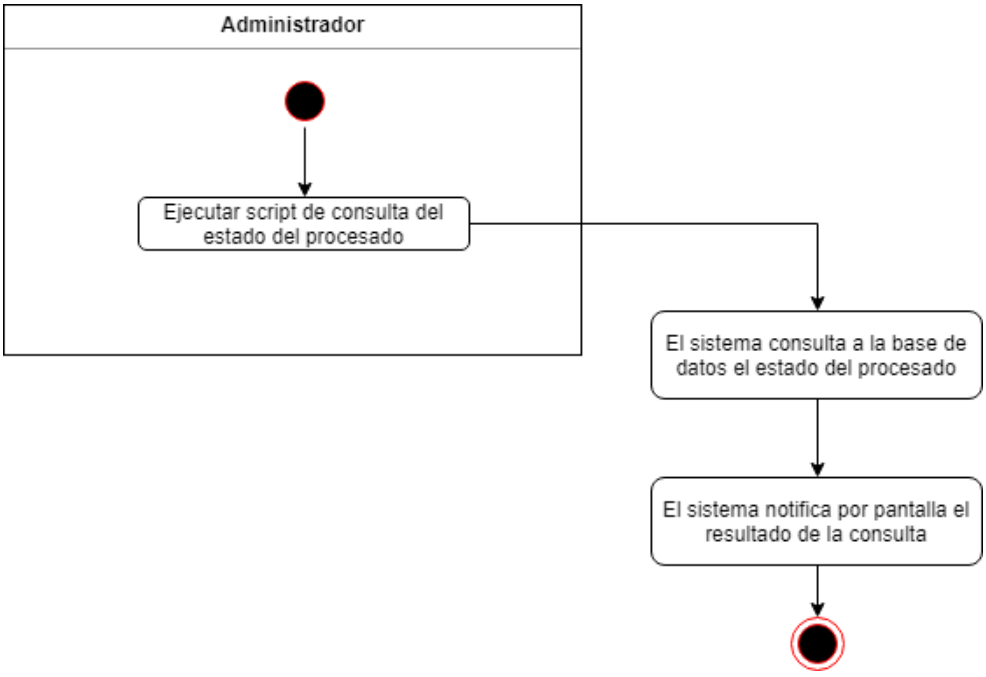


Ilustración 24: Diagrama de actividades de gestión (consultar estado de procesado)

- Adición manual de ficheros

En la *Ilustración 25* se representa el flujo de actividades cuando el administrador decide añadir un fichero de forma manual al directorio que actúa como cola de procesado.

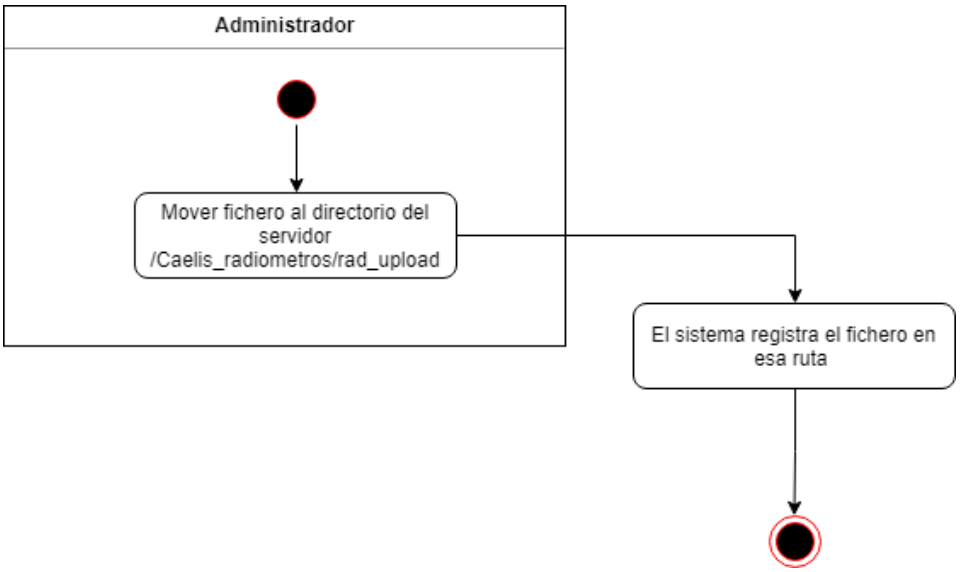


Ilustración 25: Diagrama de actividades de gestión (adición de ficheros)

- Automatización de tareas

En la figura _ se representa el flujo de actividades cuando el administrador decide crear y lanzar un servicio que se encarga de ejecutar el script se encarga de procesar los ficheros teniendo en cuenta si está el procesado en un estado activo o no.

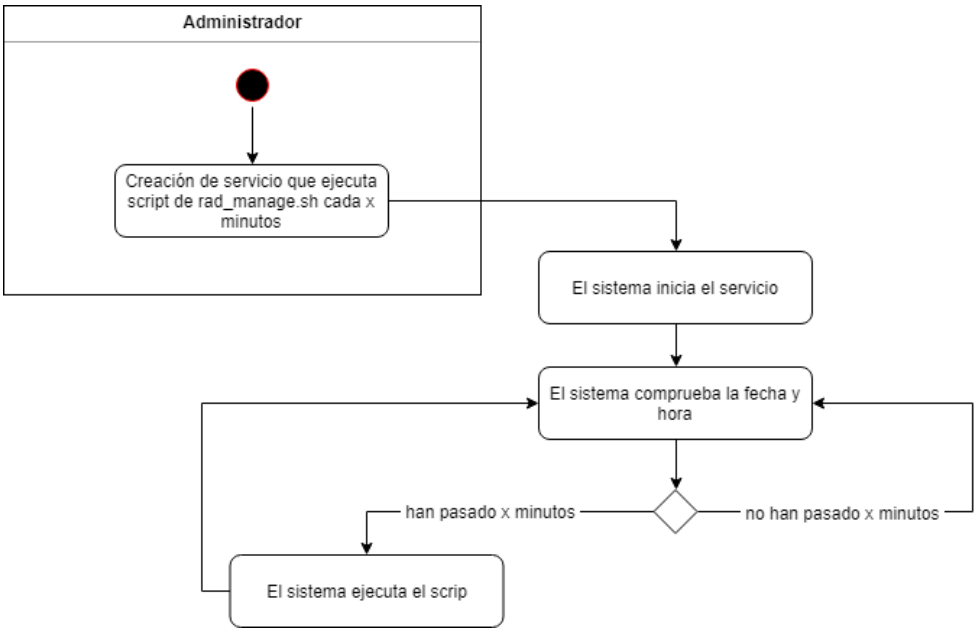


Ilustración 26: Diagrama de actividades de gestión (automatización)

- **Sincronía de ejecución y trazabilidad**

Por otro lado, en la *Ilustración 27* se hace referencia a todo el proceso de ejecución secuencial de programas y el registro de sus resultados de ejecución.

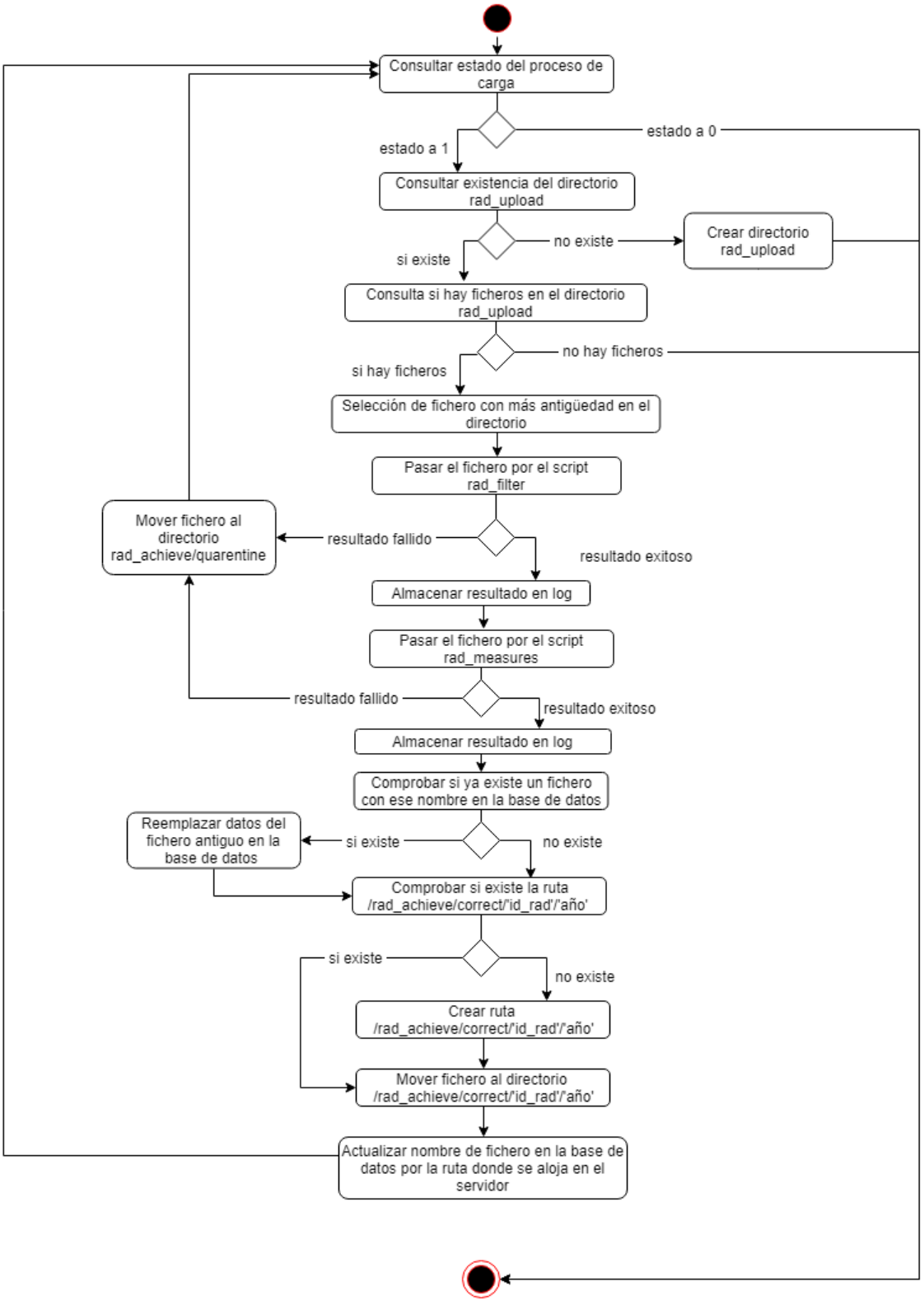


Ilustración 27: Diagrama de actividades gestión (Trazabilidad y Sincronía de ejecución)

3.4.5. Diagramas de estructura

A la hora de diseñar software pertinente que permita alcanzar los objetivos propuestos en el sistema de carga, se ha seguido un conjunto de estrategias basadas en programación estructurada. El objetivo es que, una vez conocida la solución al problema, abordar su complejidad por medio de particionamiento en módulos y su organización en jerarquía adecuada. De esta manera se recurre a los **diagramas de estructura** que permiten hacer el sistema más comprensible.

Cada programa va a estar formado por módulos o subprogramas que están formados por los siguientes atributos:

- **Entrada**: información recibida cuando se invoca.
- **Salida**: información que devuelve al módulo que lo invocó.
- **Función**: lo que hace para transformar los datos de entrada en datos de salida.
- **Nombre**: por el cual es referenciado.

Los módulos son representados por rectángulos, pero también pueden existir módulos predefinidos o de librerías los cuales han sido definidos previamente en algún sitio. Cuando los módulos se comunican intercambian información en forma de parámetros que aparecen en la llamada. A continuación, en la *Ilustración 28*, se representan estos elementos:

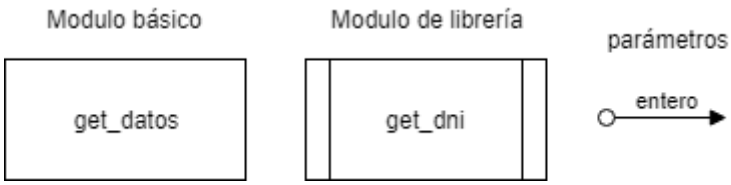


Ilustración 28: Elementos del diagrama de estructura

Una vez conocidos los elementos de los diagramas de estructura, se representan los diagramas de los distintos programas necesarios para el sistema de carga.

- **rad_filter** (*/Caelis_radiometros/ProgramsServer/filter/*)

El programa se encarga de verificar que la información de las cabeceras de los ficheros alojados en el directorio *rad_upload* son correctas. Recibe por parámetro el nombre de un fichero y retorna un valor entero, indicando si el fichero es válido o no en función de su cabecera. Utiliza el módulo ***read_head*** de una librería externa (*read_parameters.h*), que lee y almacena los valores de las cabeceras de los ficheros y seguidamente, el módulo propio filtrar consulta a la base de datos para comprobar que los datos son válidos. Para conectarse a la base de datos se hace uso del módulo ***get_conection_data*** de otra librería externa (*conection.h*) que retorna la información necesaria para conectarse a la base de datos del servidor.

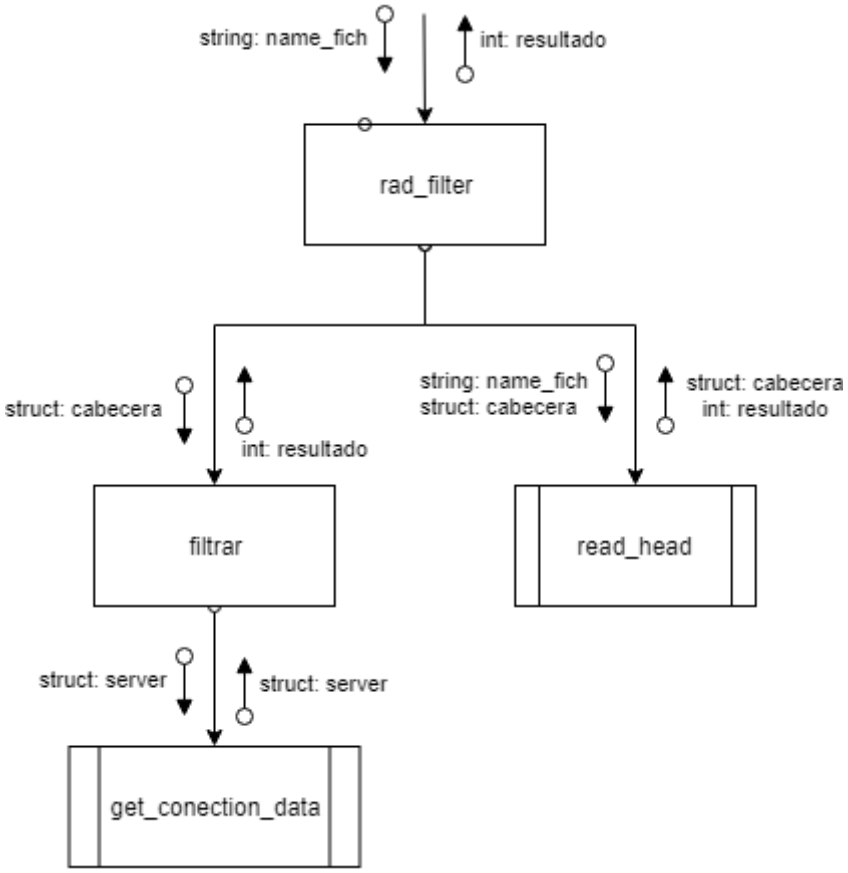


Ilustración 29: Diagrama de estructura (rad_filter.c)

- **rad_measures** (*/Caelis_radiometros/ProgramsServer/insert/*)

El programa se encarga de normalizar los datos del cuerpo del fichero de tal manera que los inserta en la base de datos previamente diseñada y estructurada. El programa necesita un parámetro de entrada que será el nombre de los ficheros a procesar y, por otro lado, devuelve un valor entero indicando si la inserción se realiza correctamente o no.

En cuanto al uso de módulos, en este caso también utiliza ***read_head*** de la librería externa *read_parameters.h*. Utiliza el módulo propio ***read_Body*** en el que se realiza la conexión a la base de datos, se leen los parámetros del cuerpo del fichero y se almacenan en la base de datos en su tabla correspondiente. Para ello, *read_Body* a su vez se apoya en los siguientes módulos:

  - ***get_conection_data***: perteneciente a la librería externa *connection.h* y encargado de retornar los parámetros correspondientes que permiten conectarse a la base de datos.
  - ***rad_insert_downloaded_measure***: módulo perteneciente a *rad_measures* que se encarga de insertar en la correspondiente tabla de la base de datos la información relacionada con los ficheros de datos.
  - **rad_update_downloaded_measure**: módulo perteneciente a *rad_measures* que se encarga de modificar los valores de la tabla correspondiente de la base de datos que está relacionada con los ficheros de datos.
  - ***rad_insert_scene***: módulo perteneciente a *rad_measures* que se encarga de insertar en la correspondiente tabla de la base de datos la información relacionada con los escenarios de datos del fichero.
  - **rad_insert_measure**: módulo perteneciente a rad_measures que se encarga de insertar en la correspondiente tabla de la base de datos la información relacionada con las medidas de datos de los escenarios.

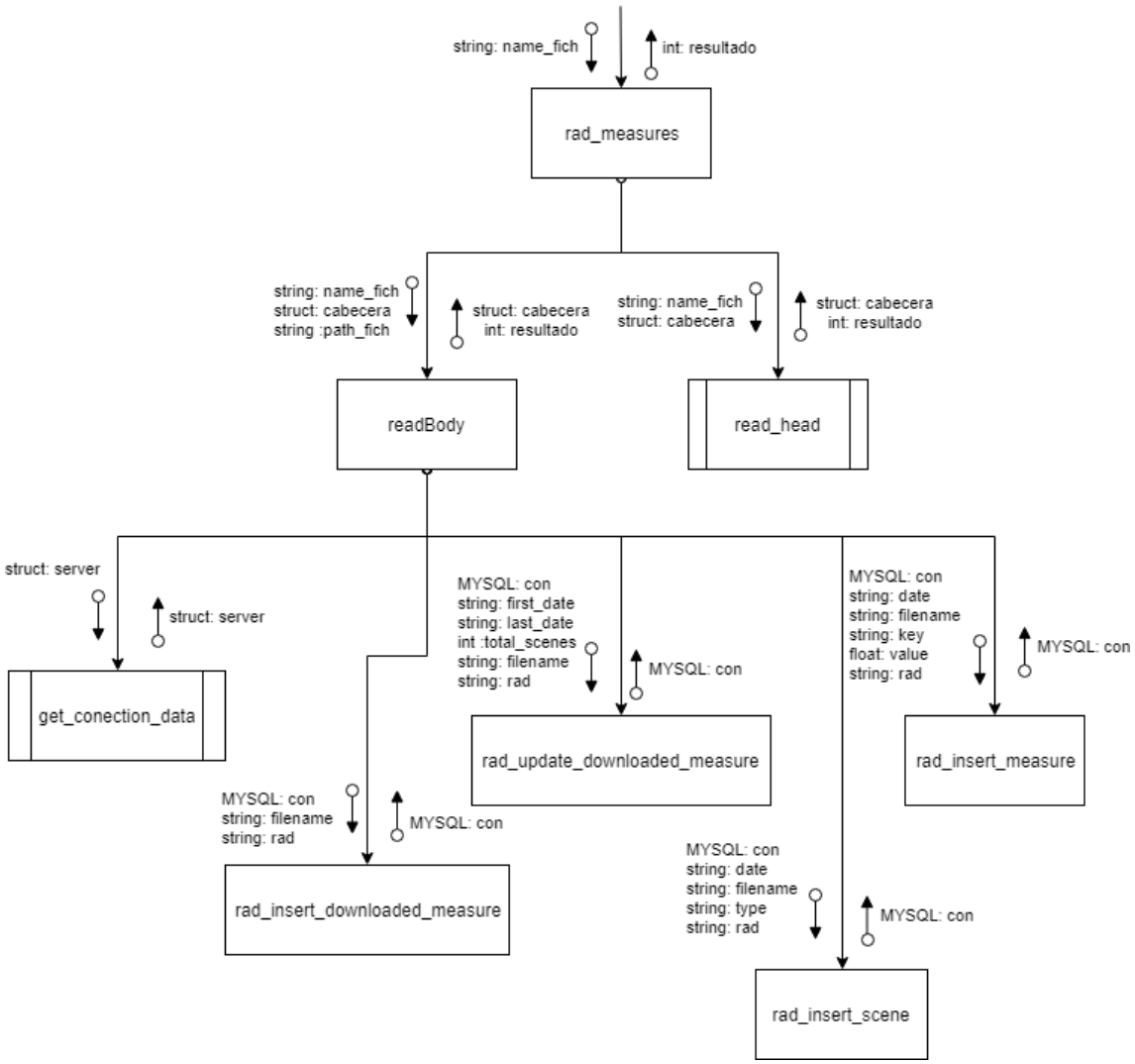


Ilustración 30: Diagrama de estructura (rad_measures.c)

- ✓ **rad_manage** (*/Caelis_radiometros/ProgramsServer/*)

Este programa es el que se encarga de gestionar el procesado de ficheros, es decir, llevará acabo la sincronía y la secuencialidad de la ejecución de los programas necesarios para logar este sistema de carga. Por otro lado, también es el que se encarga de la trazabilidad mediante log y directorios.

Como se puede ver en la *Ilustración 31*, se hace uso de módulos internos como externos.

- ***moveToCorrect***: se encarga de cambiar un fichero de ruta una vez que ha sido procesado y su resultado ha sido positivo.
- ***moveToQuarentine***: se encarga de cambiar un fichero de ruta una vez que ha sido procesado y su resultado ha sido negativo.
- ***deleteFich****:* se encarga de eliminar un fichero de la base de datos que tenga las características que se le pasan como parámetro.
- ***updateNameFich***: se encarga de modificar el nombre de un fichero registrado en la base de datos. Su nuevo nombre será el que corresponde con la ruta del directorio donde se encuentra alojado en el servidor.
- ***setLog***: inserta en la base de datos registros que almacenan los resultados de las ejecuciones que han sido positivas teniendo en cuenta el programa que se ejecuta, el fichero que se procesa y la fecha y hora en que se realiza.
- ***rad_getId***: modulo externo que retorna el valor del identificador del radiómetro que generó el fichero que se está procesando.
- ***conection_db***: modulo externo que ofrece una ayuda a la hora de realizar operaciones sobre la base de datos ya que optimiza consultas y proporciona seguridad a la hora de conectarse a la base de datos del servidor.
- ***rad_filter***: hace referencia al programa representado en la *Ilustración 29*.
- ***rad_measure***: hace referencia al programa representado en la *Ilustración 30*.

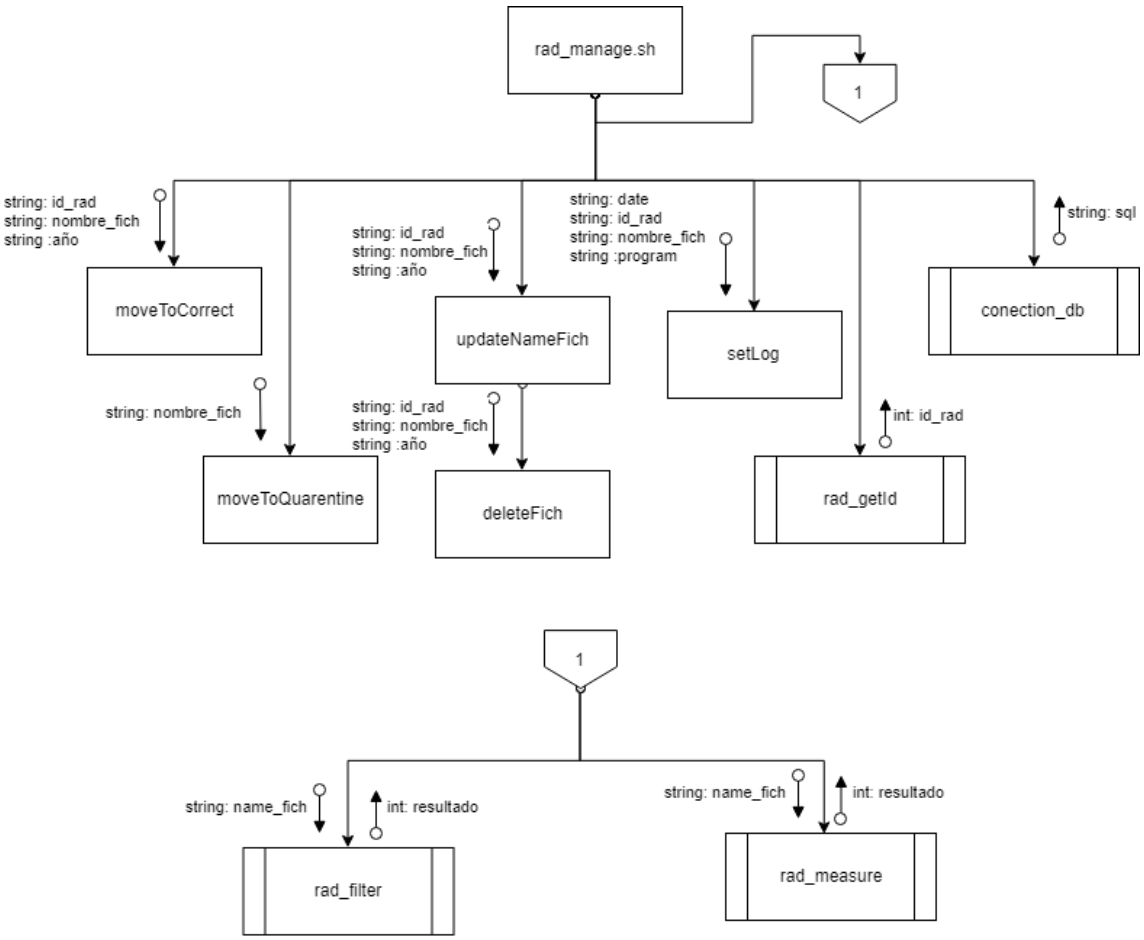


Ilustración 31: Diagrama de estructura (rad_manage.sh)

- ✓ **rad_activate, rad_desactivate, rad_consult_status** (*/Caelis_radiometros/ ProgramsServer/rad_activation_control/*)

Finalmente queda mencionar aquellos programas con los que interactúa el administrador y que permiten activar, desactivar y consultar el estado de ejecución del procesado y carga de ficheros. Se trata de 3 programas sencillos que únicamente necesitan un módulo externo que les permita conectarse a la base de datos del servidor y poder hacer operaciones sobre ella. El módulo que ofrece esa funcionalidad es ***conection_db.sh*** (ya mencionado previamente).

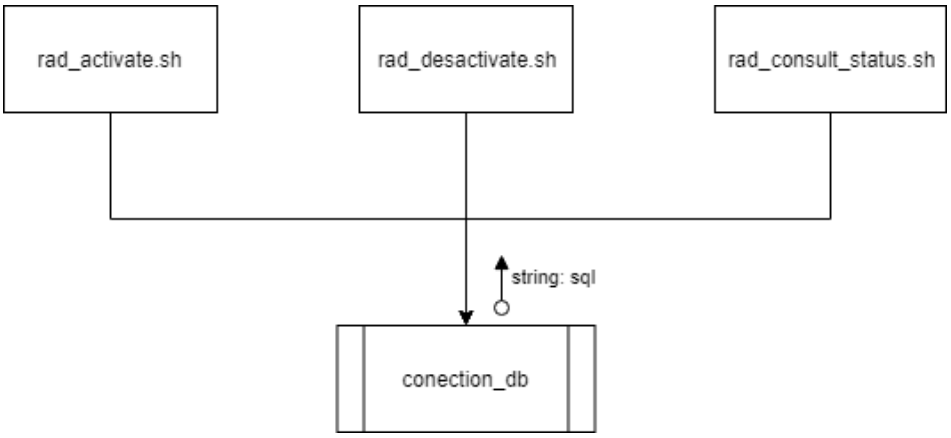


Ilustración 32: Diagrama de estructura (rad_activate.sh, rad_desactivate.sh, rad_consult_status.sh)

3.3. Implementación

Esta etapa del proceso de desarrollo del software es en la que se ha programado el código de la aplicación. Para ello se han tenido en cuenta todas las herramientas, lenguajes y tecnologías utilizadas para la implementación del código que da lugar al sistema de carga.

3.4.6. Servidor

El servidor sobre el que corre el sistema de carga es el proporcionado por el cliente (goacf.opt.cie.uva.es). Este cuenta con la distribución **GNU/Linux Debian** instalada.

3.4.7. Lenguajes utilizados

Desde un principio el sistema se subdividió en 3 características principales, el filtrado de datos, la inserción de datos y la gestión del sistema. Se ha tenido en cuenta que para la elaboración del filtrado y la inserción se ha utilizado **lenguaje *C***, mientras que para todo lo relacionado con la gestión se ha utilizado lenguaje ***BashScript***. El motivo de utilizar estos lenguajes hace referencia a la eficiencia de programar a bajo nivel. El lenguaje *C* se ha utilizado sobre todo para el tratamiento de la información de los ficheros y su lectura. Por otro lado, al trabajar bajo un servidor Linux, es más sencilla y rápida la programación en *Bash*, como por ejemplo a la hora de mover archivos en directorios, realizar búsquedas de ficheros…

Para el desarrollo de la base de datos se ha recurrido a un enfoque relacional basado en tablas y por lo tanto, se ha utilizado lenguaje ***SQL*** ejecutado sobre el servidor de bases de datos *MySQL*.

3.4.8. Herramientas para el desarrollo

A continuación, se lista las herramientas que han intervenido en el desarrollo del sistema de carga:

- ***CodeBlocks***

Entorno de desarrollo integrado de código abierto y que soporta múltiples compiladores, entre ellos *Clang*, *Visual C++* y el que se ha tenido en cuenta en este caso *GCC*. Se ha utilizado para el desarrollo de todos aquellos programas escritos en lenguaje *C*.

- **Notepad++**

Se trata de un editor de código fuente libre con soporte para varios lenguajes de programación. Este se ha utilizado para el desarrollo de los scripts que se han escrito en *Bash*.

- ***MySQLWorkBench***

Herramienta visual de diseño de bases de datos que integra desarrollo de software, administración, diseño, gestión y mantenimiento para el sistema gestor de bases de datos *MySQL*.

- ***phpMyAdmin***

Herramienta escrita en *PHP* que tiene la intención de manejar la administración de *MySQL* a través de páginas web, utilizando internet.

- ***VirtualBox***

Se ha utilizado para generar una máquina virtual con sistema operativo *Ubuntu* (*Linux*). Se ha manejado para simular el sistema operativo del servidor real en el que se va a desplegar el sistema de carga. Una vez que todos los programas y scripts han sido probados en local y funcionan correctamente, se configurarán para ser desplegados en el servidor real.

- ***Git / GitHub***

Software de control de versiones pensado en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente. Se puede llevar un control en local, pero también se puede ampliar este control almacenando repositorios en remoto mediante *GitHub*.

En este caso, se ha llevado el control tanto del código *SQL* de la base de datos como del código *C* y *Bash* del sistema.

- https://github.com/Angelotas/Caelis_DB
- https://github.com/Angelotas/Caelis_radiometros

Dentro de este último repositorio se almacenan dos directorios: *Programs* y *ProgramsServer*. Uno de ellos contiene el código de los programas configurados para desplegarse en un servidor localhost (el utilizado en la máquina virtual). *ProgramsServer* contiene esos mismos programas ya configurados para trabajar en el servidor real.

Si existe algún tipo de problema para visualizar los repositorios, puede ser debido a que se encuentren en modo privados.

- ***Putty + Filecilla***

Filecilla se trata de un cliente FTP que permite la conexión desde nuestro equipo a un servidor, en este caso al http://goacf.opt.cie.uva.es. Cuenta con una interfaz de usuario que hace más amena la gestión de ficheros en el servidor. Se ha utilizado básicamente para la subida de los recursos al servidor, tanto programas como ficheros de prueba *.rad* con los que se han podido realizar pruebas.

Putty se trata de un cliente SSH que se ha utilizado para establecer conexión ssh con el servidor http://goacf.opt.cie.uva.es. Una vez que se han subido los programas al servidor, permite lanzar programas de forma manual, activar o desactivar el sistema de carga, gestionar la automatización de la ejecución del sistema mediante *crontab*…

- ***Crontab***

Se trata de un programa que se utiliza en sistemas *UNIX* que permite programar la ejecución de otros scripts, es decir, para automatizar tareas.

Con el comando sudo `crontab -l` se pueden listar los programas que *crontab* está gestionando. Con el comando `crontab -e` se pueden añadir o quitar programas de la lista.

El formato de las tareas es el siguiente:

```
* * * * * /ruta/al/script.sh
```

Los asteriscos de izquierda a derecha:

- Minutos: de 0 a 59.
- Horas: de 0 a 23.
- Día del mes: de 1 a 31.
- Mes: de 1 a 12.
- Día de la semana: de 0 a 6, siendo 0 el domingo.

En el caso del sistema de carga, como se ha establecido que la tarea de procesamiento de ficheros se ejecute cada 5 minutos, la sentencia es la siguiente:

```
*/5 * * * *
/home/angel/Caelis_radiometros/Programs/rad_manage.sh
```

3.4. Pruebas

Esta fase es una de las partes más importantes del proceso de desarrollo ya que es el periodo en el que se han detectado problemas y se han verificado las funcionalidades esperadas del sistema. Para la realización de pruebas se han tenido en cuenta aquellas con carácter dinámico, es decir, centrándose en el comportamiento de los programas a través de su ejecución y, de esta manera, comprobar que el resultado es satisfactorio.

Hay que tener en cuenta que, durante la fase del desarrollo, a medida que se van implementando funciones y métodos específicos de cada módulo del sistema, se van realizando pruebas. Entonces se comprueba que cada función realiza su trabajo de forma correcta. A este tipo de pruebas se las denomina **pruebas de caja blanca**.

Por otro lado, una vez que los módulos estaban desarrollados con todos sus métodos y funciones, es el momento de realizar las **pruebas de caja negra**. En este caso de evalúa el comportamiento del sistema sin conocer el funcionamiento interno, únicamente se hace un estudio teniendo en cuenta las entradas que recibe y las salidas que produce. En este caso es importante saber qué es lo que hace el programa y no el cómo lo hace.

3.4.1. Pruebas de caja blanca

Módulo *rad_filter.c*

- ✓ **Método `filtrar`**: recibe una estructura de datos que almacena los valores de la cabecera. Para que el retorno de la función sea 1 es necesario que en la base de datos existan registros que coincidan con los de la estructura recibida en la función. Para ello, los datos recibidos deben tener formato correcto y la conexión con la base de datos debe ser satisfactoria. En cuanto a la conexión con la base de datos, se han realizado pruebas en una alojada en un servidor local y en la base de datos real que utiliza el sistema.

- ✓ **Método `get_conection_data`**: recibe una estructura de datos en la que se van a almacenar los valores necesarios para la conexión con el servidor. Después devuelve esa misma estructura rellena. Únicamente tendrá un resultado satisfactorio cuando el formato de los datos que se almacenan en la estructura es correcto.

- ✓ **Método `read_head`**: recibe la ruta donde se aloja un fichero con formato *.rad* y una estructura de datos vacía de tipo `cabecera` donde se almacenarán los datos de la misma para luego ser devueltos. El método devolverá un resultado satisfactorio en caso de que el fichero exista en esa ruta, y en caso de que exista, la cabecera debe tener el mismo formato que el que se aprecia en la *Ilustración 12*.

Módulo *rad_measures.c*

- ✓ **Método `readBody`**: recibe el nombre de un fichero con formato *.rad*, una estructura de datos a la que añadir nueva información y la ruta donde se encuentra el fichero. El resultado será válido siempre que el nombre del fichero recibido exista en la ruta recibida. Además de existir el fichero, el formato de su contenido tiene que ser similar al formato que se aprecia en la *Ilustración 12.* Otro requisito necesario es que la conexión con la base de datos se establezca de forma correcta.

- ✓ **Método `read_head`**: Se aplican las mismas pruebas que en el programa *rad_filter*.

- ✓ **Método `get_conection_data`**: Se aplican las mismas pruebas que en el programa Rad_filter.

- ✓ **Método `rad_insert_downloaded_measure`**: recibe un parámetro de tipo `MYSQL` que establece la conexión con la base de datos. Por otro lado, recibe el valor del nombre del fichero y del identificador del dispositivo. Para que la inserción en la base de datos sea correcta, el parámetro de tipo `MYSQL` debe ser correcto para conectarse a la base de datos. Y, por otro lado, los valores del nombre del fichero y el identificador del radiómetro deben seguir las restricciones establecidas en la base de datos `MySQL` en cuanto a unicidad, integridad y formato. Por ejemplo, si se está intentando insertar un registro cuyo nombre de fichero e identificador de radiómetro ya existe en la base de datos, la inserción no se llevará a cabo, aunque la ejecución del programa no finalizará. También ocurrirá lo mismo en caso de que no exista una referencia al identificador del radiómetro en la base de datos. En caso de que la inserción sea correcta, se comprueba que únicamente se inserta el nombre del fichero y el id del radiómetro al que hace referencia, dejando a `NULL` el resto de los campos.

- ✓ **Método `rad_update_downloaded_measure`:** recibe un parámetro `MYSQL` debe de ser válido para conectarse a la base de datos. Recibe otros datos como fechas, un entero con el valor del total de escenarios leídos, el nombre del fichero y el identificador del radiómetro. Para que se lleve a cabo la modificación, los parámetros de entrada, deben respetar las restricciones establecidas en la base de datos `MySQL`.

- ✓ **Método `rad_insert_scene`:** recibe un parámetro `MYSQL` y otros datos como una fecha, el nombre del fichero, un tipo de medición y el identificador del radiómetro. Para una inserción satisfactoria la conexión de a la base de datos debe ser correcta y el resto de los datos deben respetar las restricciones establecidas en la base de datos `MySQL`.

- ✓ **Método `rad_insert_measure`:** recibe un parámetro MYSQL y otros datos como fecha, nombre de fichero, clave, valor e identificador de radiómetro. La inserción es satisfactoria en casos similares a los descritos previamente.

Módulo *rad_manage.sh*

- ✓ **Método `moveToCorrect`:** recibe por parámetro los valores necesarios que dan lugar a la ruta donde se tiene que mover un fichero. El resultado del método será correcto siempre que la unión de los parámetros recibidos dé lugar a una ruta existente en el servidor.

- ✓ **Metodo `moveToQuarentine`:** recibe por parámetro el nombre del directorio donde mover el fichero. Para un resultado correcto, la ruta del directorio debe existir en el servidor.

- ✓ **Método `deleteFich`:** recibe por parámetro el nombre del fichero que se debe borrar de la base de datos. Para que el borrado sea correcto es necesario que el fichero exista en la base de datos y se elimine de forma previa de las demás tablas que lo referencian.

- ✓ **Método `updateNameFich`:** recibe por parámetro los valores que tiene que tener el nuevo nombre que va a tomar un registro de fichero de la base de datos. Para que se modifique de forma correcta, es necesario que exista el que se quiere modificar. También es necesario que el nuevo nombre que va a tomar no exista en la base de datos. En ese caso se abortaría el modificado. Por último, al realizar la modificación sobre el registro, se deberán modificar en cascada todos aquellos registros que lo referencien.

- ✓ **Método `setLog`:** recibe por parámetro una fecha, un nombre de fichero, un identificador de radiómetro y un nombre de programa. Para que se realice una inserción de un registro log en la base de datos, es necesario que el formato de la fecha sea correcto y, por otro lado, el resto de los valores recibidos por parámetro existan en la base de datos ya que el registro referencia a los registros de otras tablas. Por ejemplo, si el nombre del programa que se pasa por parámetro fuera *rad_example*, no se crearía un nuevo registro en la tabla de la base de datos correspondiente al log ya que no existe ninguna referencia al programa *rad_example* en otra tabla (únicamente *rad_filer* y *rad_measure*).

- ✓ **Método `rad_getId`:** únicamente devuelve un entero indicando la línea del fichero en la que se encuentra el valor del identificador del radiómetro. Este programa no tendrá errores ya que el valor siempre será devuelto. El problema está en el caso de que se cambie el formato de la cabecera del fichero y en esa línea devuelta ya no se recoja el valor del id del radiómetro.

- ✓ **Método `conection_db`:** devuelve una cadena que permite realizar operaciones con la base de datos de forma óptima y segura. Para ello utiliza un fichero de configuración de mysql que encapsula los datos del host, usuario, contraseña… para que no sean accesibles. De esta manera únicamente será válido este método cuando exista ese fichero de configuración en la ruta especificada, y sus datos sean correctos y válidos para conectarse a la base de datos.

- ✓ **Método `rad_filter`:** recibe por parámetro el nombre de un fichero y genera un resultado en función del procesado de este. En ese procesado influyen todos los métodos indicados en el módulo *rad_filter*, por lo tanto, su resultado va influido por todos los métodos que forman su módulo.

- ✓ **Método `rad_measure`:** recibe por parámetro el nombre de un fichero y genera un resultado en función del procesado de este. En ese procesado influyen todos los métodos indicados en el módulo *rad_measures*, por lo tanto, su resultado va influido por todos los métodos que forman su módulo.

3.4.2. Pruebas de caja negra

En estas pruebas, se inicializa el servicio que se encarga de ejecutar el script rad_manage.sh cada 5 minutos. Éste se encarga de recoger todos los ficheros que haya en el directorio *rad_upload* y procesarlos uno por uno teniendo en cuenta la antigüedad como factor de orden de procesado. Las pruebas que se han realizado a nivel global han sido en el servidor *goacf.opt.cie.uva.es* donde se implantará de forma real la aplicación y por otro lado en el ordenador personal bajo una distribución *Linux* utilizando un servidor local de prueba sobre el que se ha creado una réplica de la base de datos real. Para ello, únicamente ha sido necesario cambiar una serie de parámetros que vienen indicados en el *Manual de instalación* de este documento.

Las pruebas realizadas son las siguientes:

1. Comprobar que el servicio que ejecuta *rad_manage.sh* funciona correctamente.

Poblar el directorio *rad_upload* con ficheros con formato *.rad*, comprobar que el estado del procesado está activado y verificar si antes de 5 minutos se han procesado los ficheros del directorio.

2. Comprobar que la consulta del estado del procesado funciona.

Se ha ejecutado el script *rad_consult_status.sh* alojado en el directorio *rad_activation_control*. Se comprueba que devuelve una notificación por consola indicando si el estado está activo o inactivo. Para verificar su resultado se accede a la interfaz de la base de datos mediante *phpMyAdmin* y se comprueba en la tabla *rad_action* dentro de la base de datos *caelis* si el estado se encuentra con valor `0` o `1`, desactivado o activado respectivamente.

3. Comprobar que la activación del estado del procesado funciona.

Se ha ejecutado el script *rad_activate.sh* alojado en el directorio *rad_activation_control*. Se comprueba mediante el script *rad_consult_status.sh* que el estado es activo. Por otro lado, poblar el directorio *rad_upload* con ficheros con formato *.rad* y comprobar si antes de 5 minutos han sido procesados.

4. Comprobar que la desactivación del estado del procesado funciona.

Se ha ejecutado el script *rad_desactivate.sh* alojado en el directorio *rad_activation_control*. Se comprueba mediante el script *rad_consult_status.sh* que el estado está inactivo. Por otro lado, poblar el directorio *rad_upload* con ficheros con formato *.rad* y comprobar si pasados 5 minutos los ficheros continúan en el mismo directorio y por lo tanto no se han procesado.

5. Comprobar que la búsqueda de fichero funciona correctamente.

Con el servicio en marcha y el estado de procesado activado, añadir un fichero al directorio *rad_upload* que tenga un formato *.rad*. Comprobar si en menos de 5 minutos ha sido procesado verificando que el fichero ya no está en el directorio *rad_upload*. Por otro lado, añadir otro fichero al directorio, pero en este caso con un formato diferente, por ejemplo *.txt*. Una vez que pasen 5 minutos comprobar que el fichero no se ha procesado y continúa en el mismo directorio.

6. Comprobar que el filtrado funciona correctamente.

Con el servicio en marcha y el estado de procesado activado, añadir un fichero al directorio *rad_upload* que tenga formato *.rad* y alguno de los valores de la cabecera no se encuentre en la base de datos. Por ejemplo, como en la *Ilustración 33*, que el nombre de la estación no exista en la base de datos.

```
##########Cabecera##########
CMP21
090309
ABCDE
SW-AVG, SW-MAX, SW-MIN, SW-STD, TEMP
##########Medidas##########
#fecha,media,max,min,std,temp#
2017-07-13 10:44:00,7.81,7.81,10.81,0,null
2017-07-13 10:45:00,7.81,7.81,7.81,.002,null
2017-07-13 10:46:00,7.84,7.86,7.81,.015,null
2017-07-13 10:47:00,7.86,7.87,7.84,.007,null
2017-07-13 10:48:00,7.86,7.87,7.85,.003,null
2017-07-13 10:49:00,7.86,7.87,7.85,.007,null
2017-07-13 10:50:00,7.87,7.88,7.85,.007,null
2017-07-13 10:51:00,7.88,7.89,7.85,.013,null
2017-07-13 10:52:00,7.88,7.9,7.85,.016,null
```

Ilustración 33: Fichero con cabecera errónea

Se comprueba que funciona correctamente si una vez procesado el fichero, su nueva ruta se dirige al directorio */Caleis_Radiometros/Files/quarentine*.

En este caso, el dato erróneo de la cabecera era la dirección, pero también se ha probado asignando valores incorrectos al resto de parámetros. Por otro lado, se ha realizado este mismo proceso con un fichero cuya cabecera es válida. Entonces comprobar que en la base de datos hay un nuevo registro en la tabla del log (*rad_log_data_downloaded*) que indica que se ha ejecutado el programa *rad_filter* en una fecha concreta y sobre un fichero concreto.

7. Comprobar que la inserción de un nuevo fichero funciona correctamente.

En este caso se realizan varias pruebas. Primero se comprueba que el servicio está en marcha y el estado de procesado activo. A partir de aquí se realizan diferentes comprobaciones:

- Inserción correcta

Introducir fichero con formato *.rad* en el directorio *rad_upload*. El fichero tendrá una cabecera válida y el formato del cuerpo del archivo también es correcto. Entonces una vez que se procesa comprobar que las tablas de la base de datos se completan correctamente, tanto las tablas que guardan la información del fichero y de las medidas, como la tabla que almacena el log de procesados. En ese caso debe haber un nuevo registro en la tabla del log indicando que se ha ejecutado *rad_measures* sobre el fichero concreto en una fecha concreta.

- Sin repetición de datos

Introducir el mismo fichero anterior y comprobar que al ser procesado se sobrescriben los datos anteriores, aunque en este caso serán iguales. El único aspecto que se actualiza es la fecha de ejecución del programa que se almacena en el log.

- Omitir valores `null`

Comprobar que cuando un fichero se procesa y alguno de los campos de un escenario tiene valor `null`, dicha medida no es insertada en la base de datos. Por ejemplo, en la *Ilustración 34* todos los valores del parámetro `TEMP` tienen valor `null` y no se añaden a la base de datos.

- Omitir líneas en blanco en el fichero

Dejar una línea en blanco en el cuerpo del fichero (*Ilustración 34*) y pasar a procesarlo. Entonces se comprueba que el número de escenarios que tiene el fichero no se ve alterado en la base de datos por esa línea vacía.

```
##########Cabecera##########
CMP21
090309
Valladolid
SW-AVG, SW-MAX, SW-MIN, SW-STD, TEMP
##########Medidas##########
#fecha,media,max,min,std,temp#
2017-07-13 11:14:00,7.75,7.76,7.73,.006,null
2017-07-13 11:15:00,7.79,7.82,7.74,.022,null
2017-07-13 11:16:00,7.82,7.83,7.8,.01,null
2017-07-13 11:17:00,7.82,7.84,7.81,.009,null

2017-07-13 11:18:00,7.84,7.86,7.82,.011,null
2017-07-13 11:19:00,7.83,7.86,7.82,.013,null
2017-07-13 11:20:00,7.87,7.89,7.86,.007,null

2017-07-13 11:21:00,7.87,7.9,7.83,.024,null
2017-07-13 11:22:00,7.86,7.88,7.84,.007,null

2017-07-13 11:23:00,7.86,7.88,7.84,.008,null
```

Ilustración 34: Fichero con líneas en blanco en el cuerpo

- Inserción con formato para mediciones de tipo *LW* (Radiación Infrarroja)

A diferencia del resto de tipo de mediciones, *LW* cuenta con un parámetro mas (`TEMP2`). Añadir un fichero al directorio cuya cabecera recoja que el tipo es *LW* y añadir escenarios que tengan ese nuevo parámetro (véase en la *Ilustración 35*).

```
##########Cabecera##########
CMP21
090309
Valladolid
LW-AVG, LW-MAX, LW-MIN, LW-STD, TEMP, TEMP2
##########Medidas##########
#fecha,media,max,min,std,temp#
2017-07-13 11:14:00,7.75,7.76,7.73,.006,null, 0.15
2017-07-13 11:15:00,7.79,7.82,7.74,.022,null, 0.14
2017-07-13 11:16:00,7.82,7.83,7.8,.01,null, 0.10
2017-07-13 11:17:00,7.82,7.84,7.81,.009,null, 0.14
2017-07-13 11:18:00,7.84,7.86,7.82,.011,null, 0.14
2017-07-13 11:19:00,7.83,7.86,7.82,.013,null, 0.17
2017-07-13 11:20:00,7.87,7.89,7.86,.007,null, 0.14
2017-07-13 11:21:00,7.87,7.9,7.83,.024,null, null
2017-07-13 11:22:00,7.86,7.88,7.84,.007,null, 0.14
2017-07-13 11:23:00,7.86,7.88,7.84,.008,null, 0.3
```

Ilustración 35: Fichero con tipo de medición LW y parámetro TEMP2

Comprobar que al ser procesado se insertan las medidas con clave `TEMP2` de forma correcta en la base de datos.

- Fichero movido a */correct*

Comprobar que cuando un fichero se ha procesado correctamente desde el directorio *rad_upload*, se ha movido a su nueva ruta */correct/id/año/nombre*, donde *id* será el identificador del radiómetro que generó ese fichero, *año* será el año actual en el que se ha procesado el fichero y el *nombre* será la cadena finalizada con *.rad*. En caso de que la ruta no esté creada, ya sea porque este fichero es el primero que se procesa de los que ha generado un determinado radiómetro o porque en este año es el primer fichero que se procesa de ese radiómetro, se autogenerará dicha ruta donde se almacenará el fichero procesado correctamente.

- Registro en log de base de datos

Comprobar que tras el procesado de cada fichero se insertan dos nuevos registros en el log. Uno indicando que se ejecuta *rad_filter* y otro indicando que se ejecuta *rad_measure* sobre ese fichero.

8. Comprobar que la inserción de un fichero existente funciona correctamente.

A partir de un fichero existente en el directorio *correct*, indicando que se ha procesado y cargado correctamente en la base de datos toda su información, supongamos que el administrador quiere modificar su cuerpo y añadir otras 4 líneas más equivalentes a 4 escenarios de mediciones más. Entonces ese fichero se mueve de nuevo al directorio *rad_upload* para volver a ser procesado. Comprobar que existen en la base de datos todos los escenarios previos y además aquellos 4 nuevos. Comprobar también que la fecha de procesado de ese fichero se ha actualizado.

9. Comprobar que se procesan los ficheros entrados manualmente por el administrador.

Este tipo de prueba se verifica al realizar todas las pruebas anteriores hasta ahora ya que, para todas ellas, los ficheros se han ido añadiendo al directorio *rad_upload* de forma manual simulando que yo soy el administrador del sistema. Básicamente con seleccionar ficheros *.rad*, moverlos al directorio mencionado y esperar a que sean procesados ya bastaría.

10. Comprobar que el borrado de un fichero en la base de datos borra el resto de los datos que lo referencian desde otras tablas.

La prueba se realiza de dos maneras:

1- Directamente sobre la base de datos eliminando un registro de la tabla correspondiente a los ficheros (*rad_data_downloaded_measure*) y comprobando que el resto de registros que referencian a este se han eliminado. Eliminaría en cascada los registros referentes a él en las tablas *rad_data_scene*, *rad_data_measures*, *rad_log_data_downloaded*.

2- Añadiendo un fichero al directorio *rad_upload* que ya ha sido insertado previamente en la base de datos. Hay que comprobar que el sistema se encarga de borrar el antiguo registro de la base de datos con todas las referencias a otras tablas y entonces se insertaría el nuevo fichero con sus correspondientes escenarios y medidas.

11. Comprobar que el borrado de un escenario elimina todas las mediciones referentes a él.

Se realizan pruebas directamente operando sobre la base de datos a través de phpMyAdmin, donde se accede a un registro de tipo escenario y se elimina manualmente. Una vez borrado comprobar que no existen registros de medidas en la base de datos que hagan referencia a dicho escenario.

Capítulo 4
Documentación técnica
Sistema de Visualización

4.1. Análisis

4.1.1. Características del sistema

Las características del sistema son todas aquellas funcionalidades principales con las que el sistema debe contar para el cliente obtenga los resultados que espera del producto a desarrollar.

Id	Nombre	Descripción
Caract-01	**Representación de medidas**	
Caract-01.1	Filtro Fecha	El sistema debe ser capaz de filtrar los datos a representar en función de la fecha seleccionada.
Caract-01.1.1	Dia concreto	El sistema representa los datos de un día concreto seleccionado por el usuario.
Caract-01.1.2	Rango días	El sistema representa los datos de una serie de días consecutivos.
Caract-01.2	Filtro Dispositivo	El sistema recibe las medidas generadas por determinado radiómetro seleccionado.
Caract-01.3	Filtro Medición	El sistema recibe las medidas de un tipo de medición seleccionado.
Caract-01.4	Mostrar Gráfica	Plasmar los datos ya filtrados en una gráfica

Tabla 58: Catact-01 del S. Visualización

Id	Nombre	Descripción
Caract-02	**Representación de estaciones**	
Caract-02.1	Mostrar estaciones en el mapa	El sistema muestra en un mapamundi la localización de las estaciones radiométricas.
Caract-02.1.1	Detalles estación	Detalles de la estación, así como el nombre de la estación, latitud, longitud, identificadores de los radiómetros instalados en las misma…

Tabla 59: Catact-02 del S. Visualización

4.1.2. Árbol de características

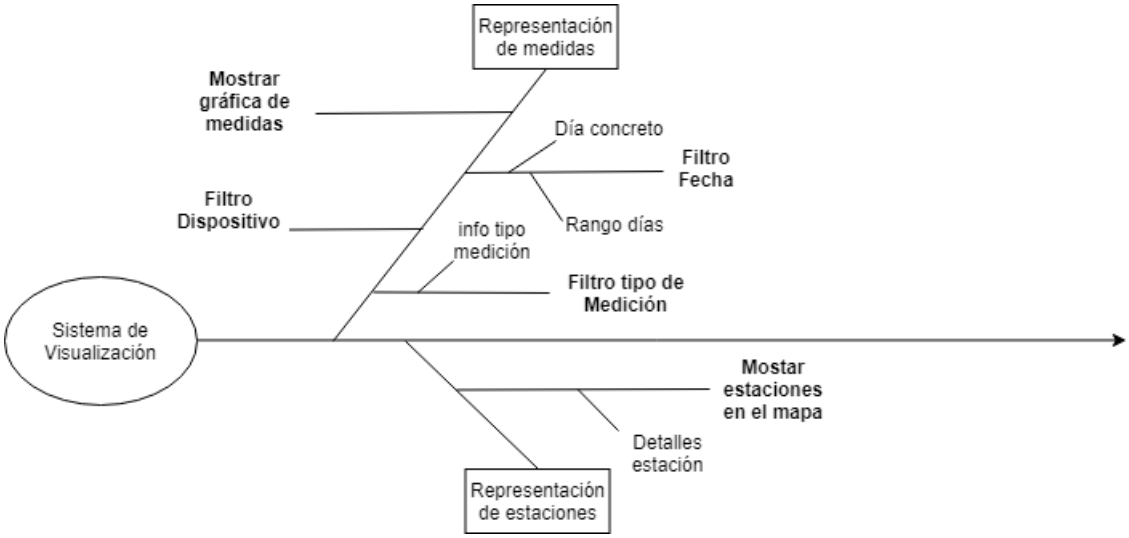


Ilustración 36: Árbol de características del S. Visualización

4.1.3. Diagrama de contexto

En el centro del diagrama se ubica el sistema de visualización de datos y, como se puede observar, recibe información externa a través de otro sistema externo, que en este caso es el sistema de carga de datos encargado de poblar la base de datos.

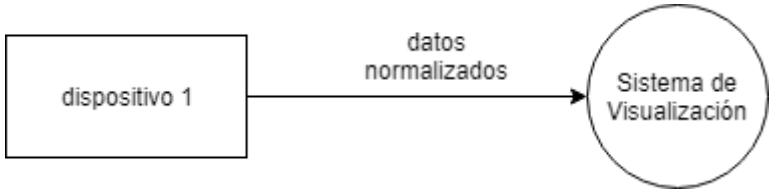


Ilustración 37: Diagrama de contexto del S. Visualización

4.1.4. Perfil de los usuarios

Usuario	Función
Usuario común	Es el cliente principal del sistema. Cualquier usuario que acceda a la web puede ver representados los datos recogidos por los dispositivos radiométricos. También puede ver situadas las estaciones radiométricas en el mapa.

Tabla 60: Perfiles de usuario del S. Visualización

4.1.5. Requisitos de usuario

A continuación, se listan todas aquellas acciones en las que el usuario interactúa con el sistema.

Id	Descripción
RU-01	El usuario puede seleccionar la opción del menú de gráficos de datos.
RU-02	El usuario indica que quiere ver los datos graficados.
RU-03	El usuario selecciona el identificador del radiómetro por el que quiere filtrar los datos.
RU-04	El usuario selecciona el tipo de medición por el que desea filtrar los datos
RU-05	El usuario selecciona el día concreto para filtrar los datos.
RU-06	El usuario selecciona el día de inicio y de fin para filtrar los datos.
RU-07	El usuario puede ver los datos concretos de una medida.
RU-08	El usuario puede seleccionar la opción del menú de mapa de estaciones radiométricas.
RU-09	El usuario pude ver la información específica de las estaciones.
RU-10	El usuario puede hacer zoom al mapa de estaciones
RU-11	El usuario puede ver una nota informativa sobre el tipo de medición seleccionado.

Tabla 61: Requisitos de usuario del S. Visualización

4.1.6. Diagrama de casos de uso

Con el siguiente diagrama se representan las relaciones establecidas entre los diferentes casos de uso para el sistema de carga.

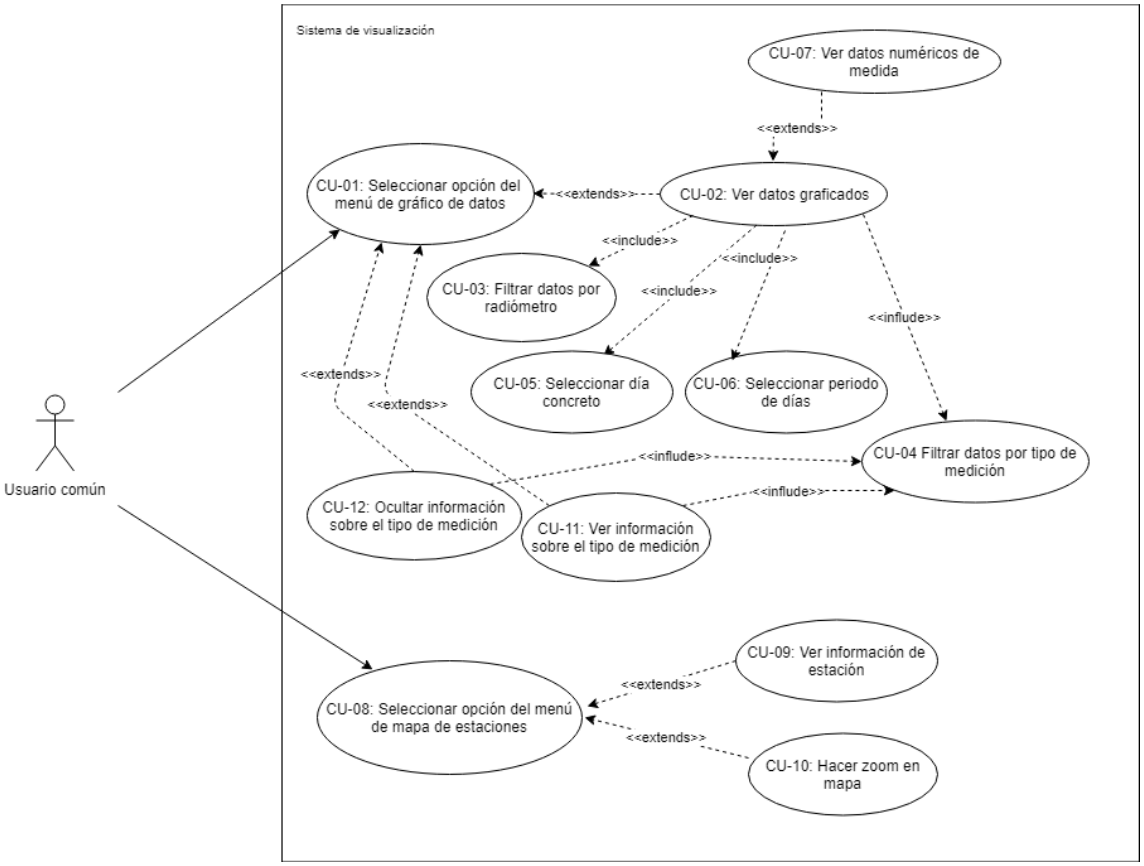


Ilustración 38: Diagrama de casos de uso del S. Visualización

4.1.7. Especificación de casos de uso

Id y Nombre	**CU-01: Seleccionar opción del menú de gráfico de datos.**
Actor	Usuario común
Descripción	El usuario puede seleccionar la opción del menú de gráficos de datos.
Precondiciones	
Flujo normal	1. El usuario selecciona el ítem del menú “Gráficos de Datos”. 2. El sistema carga la vista correspondiente de interfaz de usuario de Gráficos de Datos. 3. El sistema realiza la petición que recibe los datos de la base de datos para mostrar el listado de radiómetros. 4. El sistema realiza la petición que recibe los datos de la base de datos para mostrar el listado de tipos de mediciones. 5. El sistema carga los inputs de tipo *select* con las respuestas de las peticiones correspondientes.
Flujo alternativo	
Frecuencia	Alta: En bastantes ocasiones se selecciona esta opción. Además, es la vista principal

Tabla 62: Especificación del CU-01 S. Visualización

Id y Nombre	**CU-02: Ver datos graficados.**
Actor	Usuario común
Descripción	El usuario indica que quiere ver los datos graficados.
Precondiciones	
Flujo normal	1. El usuario pulsa el botón “Graficar datos”. 2. El sistema realiza la correspondiente petición que recibe los datos de la base de datos respetando los filtros. 3. El sistema genera la gráfica a partir de los datos previamente recibidos.
Flujo alternativo	<u>Flujo Excepcional 1</u> 1. El campo de fecha de inicio está vacío. 2. La petición tiene una respuesta vacía por parte del servidor. 3. El sistema mantiene la gráfica vacía. <u>Flujo Excepcional 2</u> 1. El campo de fecha de inicio es mayor a la fecha actual 2. La petición tiene una respuesta vacía por parte del servidor. 3. El sistema mantiene la gráfica vacía. 4. El sistema muestra un mensaje de error. <u>Flujo Excepcional 3</u> 1. El campo de fecha de inicio es mayor al de la fecha de fin. 2. La petición tiene una respuesta vacía por parte del servidor. 3. El sistema mantiene la gráfica vacía. 4. El sistema muestra un mensaje de error <u>Flujo Excepcional 4</u> 1. Los campos de fecha de inicio y de fin se llevan una diferencia mayor a 5 días. 2. La petición tiene una respuesta vacía por parte del servidor. 3. El sistema mantiene la gráfica vacía. 4. El sistema muestra un mensaje de error. <u>Flujo Excepcional 5</u> 1. La petición realizada con los filtros seleccionados tiene una respuesta vacía. 2. El sistema mantiene la gráfica vacía. 3. El sistema muestra un mensaje de warning.
Frecuencia	Alta: La principal importancia de la aplicación recae sobre esta funcionalidad, por lo tanto, su frecuencia de aparición será alta.

Tabla 63: Especificación del CU-02 S. Visualización

Id y Nombre	**CU-03: Filtrar datos por radiómetro.**
Actor	Usuario común
Descripción	El usuario selecciona el identificador del radiómetro por el que quiere filtrar los datos.
Precondiciones	PRE-01: Estar en la opción del menú “Gráficos de Datos” PRE-02: Es necesario que la base de datos esté poblada.
Flujo normal	1. El usuario accede al componente web correspondiente al selector de identificador de radiómetro. 2. El sistema despliega el listado de radiómetros disponibles. 3. El usuario selecciona el identificador deseado. 4. El sistema registra la opción seleccionada en el componente html correspondiente.
Flujo alternativo	
Frecuencia	Alta: Para graficar datos es necesario incluir el filtrado por radiómetro, por lo tanto, su frecuencia es alta.

Tabla 64: Especificación del CU-03 S. Visualización

Id y Nombre	**CU-04: Filtrar datos por tipo de medición.**
Actor	Usuario común
Descripción	El usuario selecciona el tipo de medición por el que desea filtrar los datos.
Precondiciones	PRE-01: Estar en la opción del menú “Gráficos de Datos”. PRE-02: Es necesario que la base de datos esté poblada.
Flujo normal	1. El usuario accede al componente web correspondiente al selector de tipo de medición. 2. El sistema despliega el listado de tipos de mediciones disponibles. 3. El usuario selecciona la opción deseada. 4. El sistema registra la opción seleccionada en el componente html correspondiente. 5. El sistema añade la información del tipo seleccionado al elemento html que muestra la información adicional acerca del tipo de medición seleccionada.
Flujo alternativo	
Frecuencia	Alta: Para graficar datos es necesario incluir el filtrado por tipo, por lo tanto, su frecuencia es alta.

Tabla 65: Especificación del CU-04 S. Visualización

Id y Nombre	**CU-05: Seleccionar día concreto.**
Actor	Usuario común
Descripción	El usuario selecciona el día concreto para filtrar los datos.
Precondiciones	PRE-01: Estar en la opción del menú “Gráficos de Datos”.
Flujo normal	1. El usuario selecciona la fecha deseada para el campo “Fecha de Inicio”. 2. El sistema registra la opción seleccionada en el componente html correspondiente.
Flujo alternativo	
Frecuencia	Alta: Para graficar datos es necesario incluir el filtrado por fecha concreta, por lo tanto, su frecuencia es alta.

Tabla 66: Especificación del CU-05 S. Visualización

Id y Nombre	**CU-06: Seleccionar periodo de días.**
Actor	Usuario común
Descripción	El usuario selecciona el día de inicio y de fin para filtrar los datos.
Precondiciones	PRE-01: Estar en la opción del menú “Gráficos de Datos”.
Flujo normal	1. El usuario selecciona la fecha deseada para el campo “Fecha de Fin”. 2. El sistema registra la opción seleccionada en el componente html correspondiente.
Flujo alternativo	
Frecuencia	Media: Su frecuencia es menor a la de selección de fecha inicial puesto que la fecha de fin no es obligatoria para la visualización de las medidas de un día concreto.

Tabla 67: Especificación del CU-06 S. Visualización

Id y Nombre	**CU-07: Ver datos numéricos de medida.**
Actor	Usuario común
Descripción	El usuario puede ver los datos concretos de una medida.
Precondiciones	PRE-01: Estar en la opción del menú “Gráficos de Datos”. PRE-02: Es necesario que la base de datos esté poblada. PRE-03: Es necesario que las peticiones generadas a partir de los filtros devuelvan valores.
Flujo normal	1. El usuario coloca el foco sobre cualquier punto de la gráfica generada. 2. El sistema muestra los valores de ese punto. Se tiene en cuenta la fecha, hora y valor concreto de la medición.
Flujo alternativo	
Frecuencia	Media: Es una funcionalidad adicional, no presenta gran relevancia.

Tabla 68: Especificación del CU-07 S. Visualización

Id y Nombre	**CU-08: Seleccionar opción del menú de mapa de estaciones.**
Actor	Usuario común
Descripción	El usuario puede seleccionar la opción del menú de mapa de estaciones radiométricas.
Precondiciones	PRE-01: La base de datos debe estar poblada.
Flujo normal	1. El usuario selecciona el ítem del menú “Mapa de Estaciones”. 2. El sistema carga la vista correspondiente a la opción de menú. 3. El sistema realiza una petición para obtener la información de las estaciones radiométricas. 4. El sistema realiza una petición para obtener la información de los radiómetros que se encuentran instalados en cada estación radiométrica. 5. El sistema puebla la vista con los datos recibidos en la petición.
Flujo alternativo	
Frecuencia	Media: No es una funcionalidad con gran importancia por lo que espera una frecuencia media.

Tabla 69: Especificación del CU-08 S. Visualización

Id y Nombre	**CU-09: Ver información de estación.**
Actor	Usuario común
Descripción	El usuario pude ver la información específica de las estaciones.
Precondiciones	PRE-01: Estar en la opción del menú “Mapa Estaciones”. PRE-02: La base de datos debe estar poblada.
Flujo normal	1. El usuario pulsa sobre una estación representada en el mapa. 2. El sistema muestra la información correspondiente a la estación indicada (nombre, coordenadas y radiómetros instalados.)
Flujo alternativo	
Frecuencia	Media: No es una funcionalidad con gran importancia por lo que espera una frecuencia media.

Tabla 70: Especificación del CU-09 S. Visualización

Id y Nombre	**CU-10: Hacer zoom.**
Actor	Usuario común
Descripción	El usuario puede hacer zoom al mapa de estaciones
Precondiciones	PRE-01: Estar en la opción del menú “Mapa Estaciones”. PRE-02: La base de datos debe estar poblada.
Flujo normal	1. El usuario pulsa sobre el icono correspondiente al + o – para ampliar alejar el foco del mapa. 2. El sistema aleja o acerca el mapa.
Flujo alternativo	
Frecuencia	Media: No es una funcionalidad con gran importancia por lo que espera una frecuencia media.

Tabla 71: Especificación del CU-10 S. Visualización

Id y Nombre	**CU-11: Ver información sobre el tipo de medición.**
Actor	Usuario común
Descripción	El usuario puede ver información adicional acerca del tipo de medición por el que se puede filtrar.
Precondiciones	PRE-01: La tabla de la base de datos que tenga el tipo de mediciones debe estar poblada. PRE-02: El elemento de información debe estar oculto. PRE-03: El usuario debe haber seleccionado un tipo de medición en el filtro.
Flujo normal	1. El usuario pulsa sobre el icono de información situado al lado de la etiqueta “Tipo”. 2. El sistema hace visible un elemento html que muestra la información del tipo que está marcado en el select del filtro.
Flujo alternativo	
Frecuencia	Baja: Solo es información adicional a la que el usuario puede acceder si está interesado en saber de qué se trata un tipo de medición concreta. Se espera que la aplicación sea utilizada por personal con conocimientos en el área meteorológico, por lo que no se requerirá la visualización de dicha información.

Tabla 72: Especificación del CU-11 S. Visualización

Id y Nombre	**CU-12: Ocultar información sobre el tipo de medición.**
Actor	Usuario común
Descripción	El usuario puede ocultar la información adicional acerca del tipo de medición por el que se puede filtrar.
Precondiciones	PRE-01: La tabla de la base de datos que tenga el tipo de mediciones debe estar poblada. PRE-02: El elemento de información debe estar visible. PRE-03: El usuario debe haber seleccionado un tipo de medición en el filtro.
Flujo normal	1. El usuario pulsa sobre el icono de cerrar situado en el elemento html de la información adicional. 2. El sistema oculta el elemento html que muestra la información del tipo que está marcado en el select del filtro.
Flujo alternativo	
Frecuencia	Baja: Solo es información adicional a la que el usuario puede acceder si está interesado en saber de qué se trata un tipo de medición concreta. Se espera que la aplicación sea utilizada por personal con conocimientos en el área meteorológico, por lo que no se requerirá la visualización de dicha información.

Tabla 73: Especificación del CU-12 S. Visualización

4.1.8. Requisitos funcionales

En este apartado, el análisis se centra en los aspectos más dinámicos relacionados con los requisitos funcionales. Estos requisitos describen comportamientos y funciones que el sistema tiene que seguir para lograr los objetivos propuestos en este sistema de carga.

RF-ID	DESCRIPCIÓN
RF-01	El sistema carga la vista de la opción del menú “Gráficos de Datos”
RF-02	El sistema realiza una petición para obtener el listado de radiómetros que hay en la base de datos.
RF-03	El sistema realiza una petición para obtener el listado de tipos de mediciones que hay en la base de datos.
RF-04	El sistema carga el html select con la respuesta de la petición.
RF-05	El sistema realiza la petición para obtener las mediciones que cumplan los parámetros establecidos en el filtro.
RF-06	El sistema muestra la gráfica con los datos cargados en ella.
RF-07	El sistema muestra un mensaje de error indicando que la fecha de inicio no puede ser posterior que la fecha de fin.
RF-08	El sistema muestra un mensaje de error indicando que la fecha de inicio no puede ser posterior a la fecha actual.
RF-09	El sistema muestra un mensaje de error indicando que la diferencia de días entre la fecha de inicio y la de fin es superior a 5 días.
RF-10	El sistema despliega el listado de radiómetros que se pueden seleccionar.
RF-11	El sistema registra el identificador de radiómetro como opción marcada en el filtro.
RF-12	El sistema despliega el listado de tipos de mediciones que se pueden seleccionar.
RF-13	El sistema registra el tipo de medición como opción marcada en el filtro.
RF-14	El sistema registra la fecha de inicio seleccionada por el usuario.
RF-15	El sistema registra la fecha de fin seleccionada por el usuario.
RF-16	El sistema genera un tooltip de la gráfica con los datos de una medida concreta.

RF-17	El sistema carga la vista HTML de la correspondiente vista de mapa de estaciones.
RF-18	El realiza una petición para obtener la información de las estaciones radiométricas de la base de datos.
RF-19	El realiza una petición para obtener la información de los radiómetros que hay instalados en las distintas estaciones radiométricas.
RF-20	El sistema muestra el mapa con la localización de las estaciones radiométricas.
RF-21	El sistema genera un tooltip con los datos de la estación radiométrica indicada (nombre, coordenadas y radiómetros instalados).
RF-22	El sistema aleja el mapa.
RF-23	El sistema acerca el mapa.
RF-24	El sistema cambia el contenido del componente html que muestra la información adicional sobre el tipo de medición marcada por el usuario.
RF-25	El sistema hace visible el componente html que muestra la información adicional sobre el tipo de medición marcada por el usuario.
RF-26	El sistema oculta el componente html que muestra la información adicional sobre el tipo de medición marcada por el usuario.

Tabla 74: Requisitos funcionales del S. Carga

Para hacer un seguimiento y validación de los requisitos funcionales en cuanto a los requisitos de usuario *Tabla 61* y a las características del sistema *Ilustración 36*, se ha elaborado la siguiente matriz de trazabilidad:

	RU-01	RU-02	RU-03	RU-04	RU-05	RU-06	RU-07	RU-08	RU-09	RU-10	RU-11	RU-12	Carac-01	Carac-02
RF-01	X												X	
RF-02	X												X	
RF-03	X												X	
RF-04	X												X	
RF-05		X											X	
RF-06		X											X	
RF-07		X											X	
RF-08		X											x	
RF-09		X											X	
RF-10			X										X	
RF-11			X										X	
RF-12				X									X	
RF-13				X									X	
RF-14					X								X	
RF-15						X							X	
RF-16							X						X	

RF-17								X						X
RF-18								X						X
RF-19								X						X
RF-20								X						X
RF-21									X					X
RF-22										X				X
RF-23										X				X
RF-24											X		X	
RF-25												X	X	
RF-26				X									X	

Tabla 75: Matriz de trazabilidad del S. Visualización

4.1.9. Requisitos no funcionales

Los requisitos no funcionales **(RNF)** se encargan de describir las propiedades y las restricciones que el sistema debe cumplir. A un nivel inferior se pueden encontrar los Atributos de calidad **(AC)** que son aquellos que tienen en cuenta todo aquello relacionado con la disponibilidad, la usabilidad, seguridad y rendimiento del sistema.

RNF	DESCRIPCIÓN
RNF-01	El tipo de gráfica en la que se representan las mediciones será de tipo lineal.
RNF-02	El eje horizontal de la gráfica recoge los periodos de tiempo
RNF-03	El eje vertical de la gráfica recoge los valores de las medidas.

Tabla 76: Requisitos no funcionales del S. Visualización

AC	DESCRIPCIÓN
AC-01	Cuando se realiza un filtrado por día concreto, la fecha inicial no puede ser posterior a la actual.
AC-02	Cuando se realiza un filtrado por varios días, la fecha de fin no puede ser anterior a la de inicio.
AC-03	Cuando se realiza un filtrado por varios días, la fecha de fin no puede exceder en más de 5 días a la fecha inicial.
AC-04	La aplicación será compatible con cualquier navegador que soporte HTML5.
AC-05	La aplicación estará disponible 24x7
AC-06	La distribución de los componentes html se ajustan al tamaño de la pantalla del dispositivo en el que se visualice.
AC-07	Las vistas de la aplicación se cargarán en el momento en el que el usuario lo solicite. Carga de componentes html por eventos.

Tabla 77: Atributos de calidad del S. Visualización

4.2. Diseño

4.2.1. Arquitectura lógica

A continuación, se representan los componentes lógicos que intervienen en la aplicación web, así como la relación que existe entre ellos.

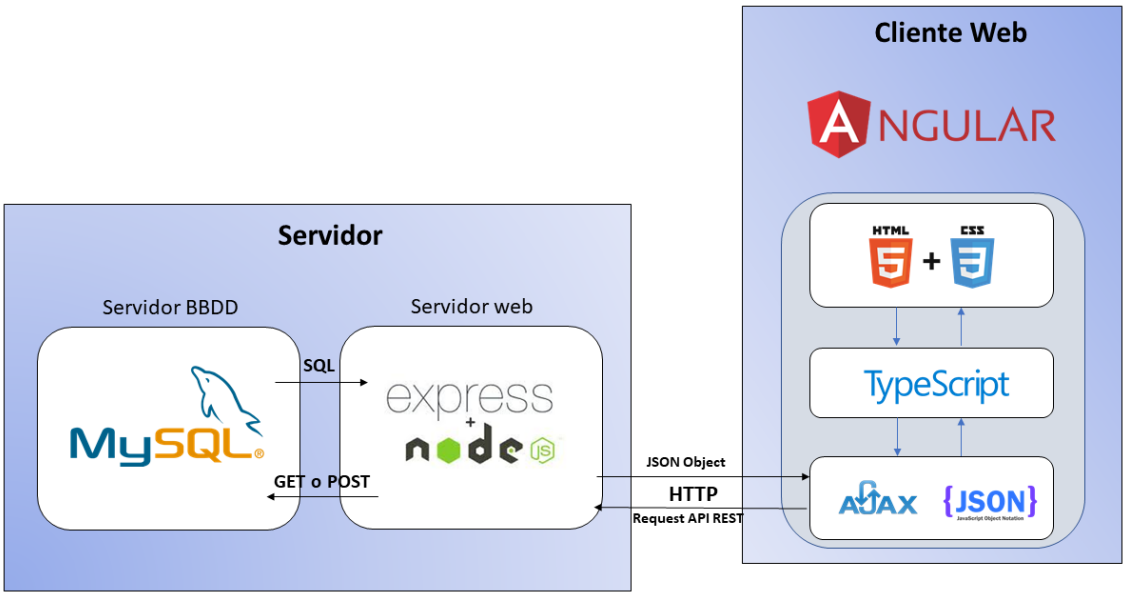


Ilustración 39: Arquitectura lógica del S. Visualización

La arquitectura está basada en la de una aplicación web de tipo **Single Page Application** en conjunto con un **API REST**. El flujo de trabajo funciona de la siguiente manera:

1. Desde el cliente, con *Angular* se hacen llamadas *AJAX* al *API* en el servidor *Node*.
2. El servidor Node consulta a la base de datos (*MySQL*) dependiendo de la llamada que requiera.
3. La base de datos retorna contenido *SQL* al servidor *Node*.
4. El servidor *Node* transforma el contenido *SQL* a formato *JSON*.
5. El servidor *Node* envía el *JSON* al cliente *Angular*.

El resultado final de todo este proceso es que desde *Angular* es capaz de mostrar todo el contenido HTML sin necesidad de recargar la página. Básicamente, *Angular* carga los *templates* HTML en la ejecución inicial y a partir de ahí va poblando los componentes a través de los *JSON* recibidos de las llamadas asíncronas al *API REST*, creando así la Single Page Application mencionada previamente.

4.2.2. Arquitectura física

A continuación, se representan los componentes físicos que participan en la arquitectura de la aplicación web, así como la relación existente entre ellos.

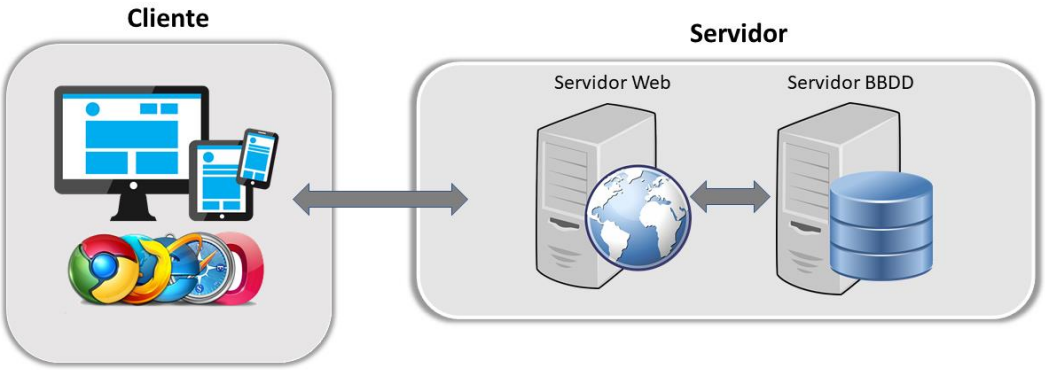


Ilustración 40: Arquitectura física del S. Visualización

El cliente será cualquier dispositivo que disponga de conexión a internet para acceder a la página web a través de un navegador. En el servidor, en donde se desplegará la aplicación, se encuentra el servidor web y el servidor de la base de datos. Éste primero contendrá la aplicación en *Angular* y el servidor *Node*. El servidor de la base de datos tendrá alojados todos aquellos datos que necesita la aplicación web.

4.2.3. Diagramas de secuencia

A través de los diagramas de secuencia se permite entender el flujo de ejecución de un proceso del sistema a partir de un vistazo rápido. En los diagramas que se representan a continuación, se muestra la interacción entre los distintos actores del sistema, así como los mensajes de comunicación que existen entre ellos.

- **Cliente**: es el encargado de disparar los eventos a través de un navegador. Esto hace que el sistema actúe de una manera u otra.
- **Sistema**: se encarga de modelar la aplicación software. Se ha considerado que tanto el servidor de aplicaciones como el API REST se encuentran dentro de este sistema.
- **MySQL**: es el gestor de base de datos del sistema forma parte del servidor de bases de datos. Se encarga de la persistencia de los datos con los que trabaja el sistema.

A la hora de representar los diagramas se han tenido en cuenta únicamente aquellos casos de uso que se han considerado más relevantes y que requieren una mayor lógica de negocio, así como la llamada a servicios.

CU-01: Seleccionar opción del menú "gráfico de datos"

En el siguiente diagrama de secuencia, una vez que el usuario ya selecciona la opción correspondiente del menú, el sistema carga inicialmente la vista referente. Una vez cargada, esta se va poblando con los datos en *JSON* recibidos de los servicios. Básicamente, se rellenan las opciones de los *inputs* de tipo *select*, con los datos traídos de la base de datos mediante un servicio *REST*.

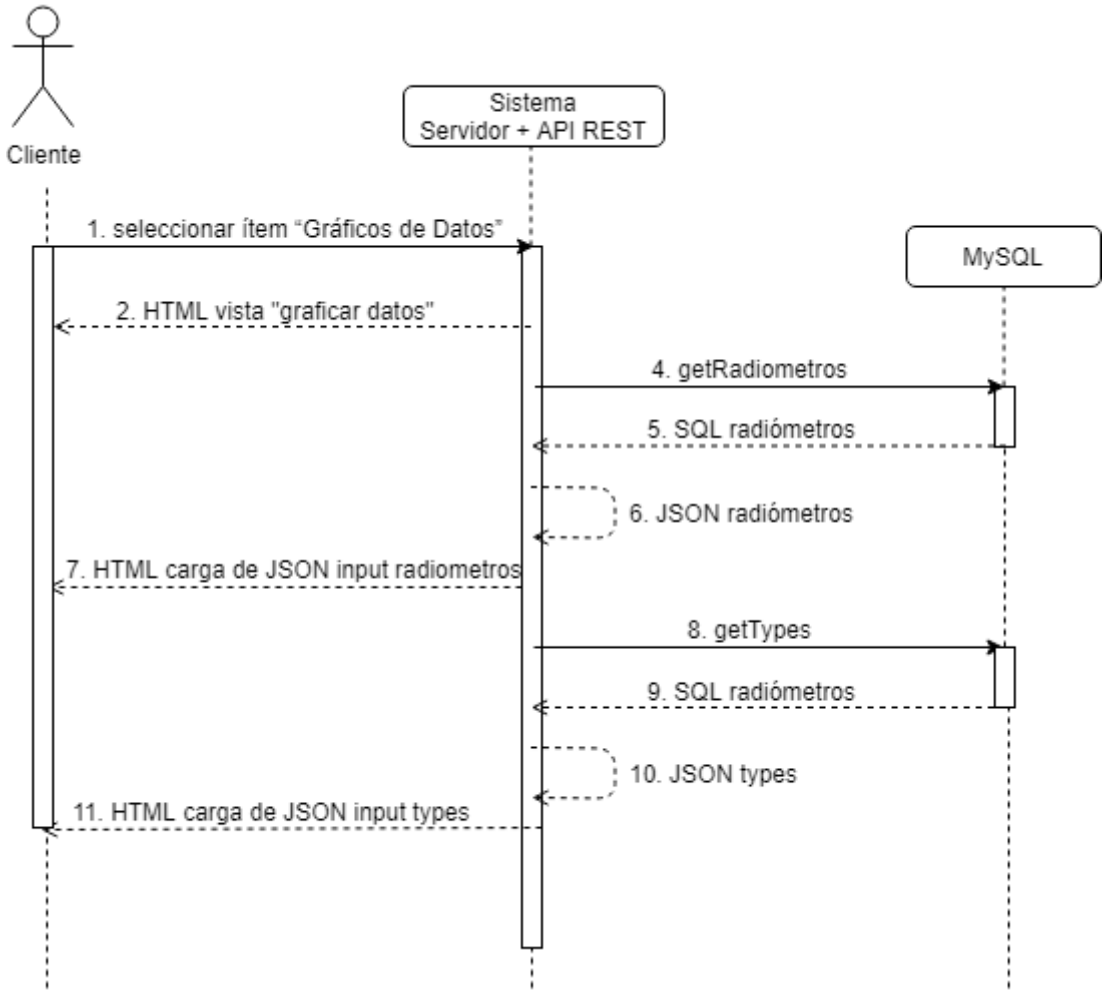


Ilustración 41: Diagrama de secuencia para CU-01

CU-02: Ver datos graficados.

Previamente, el usuario ha tenido que establecer los filtros por los que desea graficar los datos, teniendo en cuenta el radiómetro que genera los datos, el tipo de medición que realiza y la fecha. Esta última puede ser de un día concreto o de un periodo de hasta 5 días consecutivos teniendo en cuenta si el usuario deja vacío o no el campo de fecha de fin.

Cuando el usuario pulsa el botón de graficar datos, el sistema comprueba si la fecha inicial es posterior a la fecha actual. En caso afirmativo lo hace notificar al cliente y en caso contrario se pasa a verificar si el usuario ha rellenado el campo fecha de fin y, si es así, tener en cuenta las posibles excepciones que puedan surgir. Si no existe ningún inconveniente con las fechas, el sistema realiza un servicio *REST* que recoge de la base de datos las mediciones que cumplen los requerimientos establecidos en los filtros. Por último, se comprueba si la respuesta del servicio está vacía o, si en su defecto ha encontrado datos. En este último caso, el sistema genera la gráfica correspondiente y en caso contrario, la deja vacía y notifica al usuario que no hay medidas para los parámetros seleccionados.

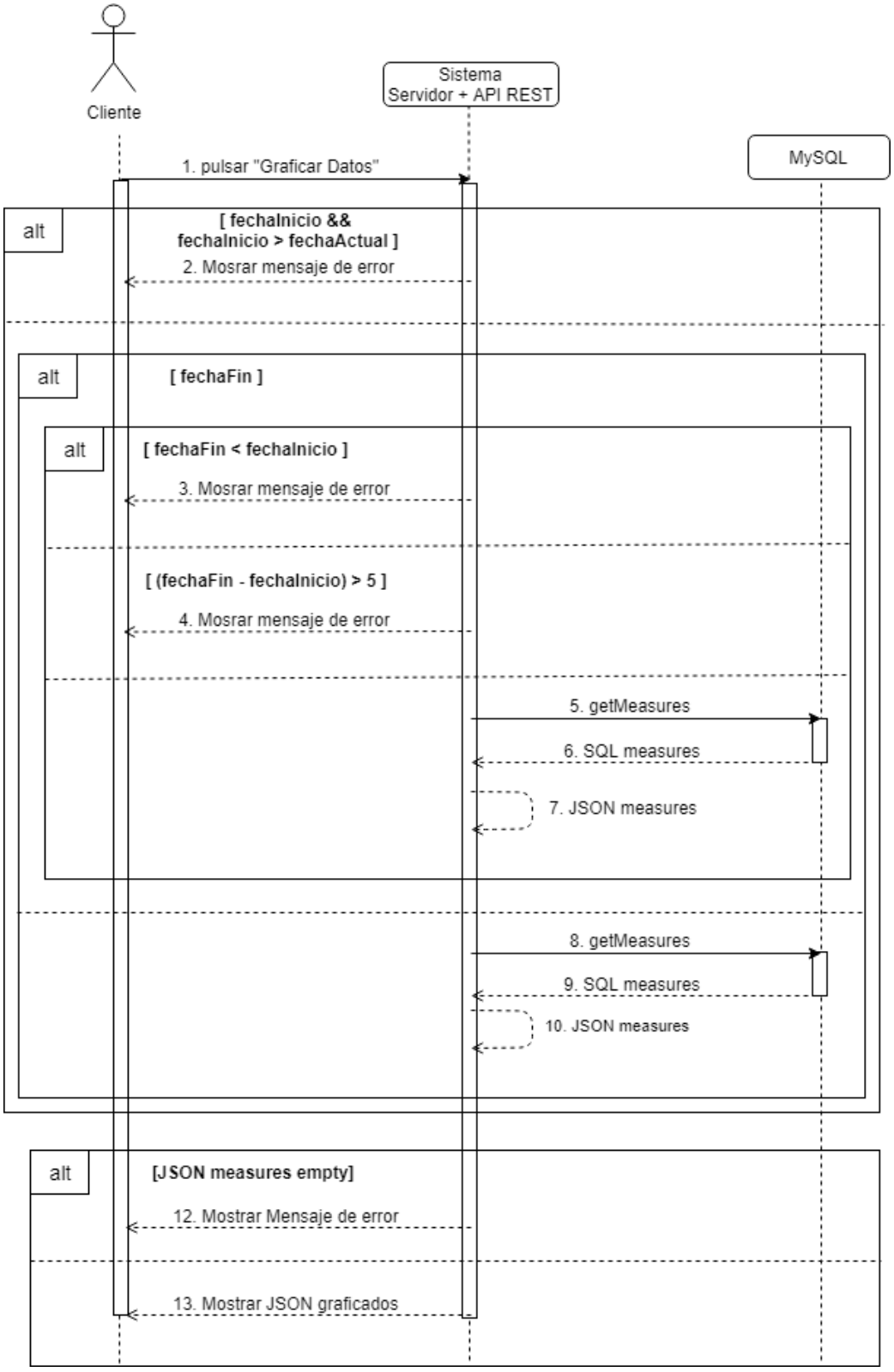


Ilustración 42: Diagrama de secuencia para CU-02

CU-08: Seleccionar opción del menú “Mapa de Estaciones”.

En el siguiente diagrama de secuencia, una vez que el usuario ya selecciona la opción correspondiente del menú, el sistema carga inicialmente la vista referente. Una vez cargada, esta se va poblando con los datos en *JSON* recibidos de los servicios. Básicamente, se parte de una carga inicial del componente mapa y una vez recibida la respuesta de los servicios *REST* de información de estaciones y radiómetros que se encuentran instalados en cada una de ellas, se pintan en el mapa los datos.

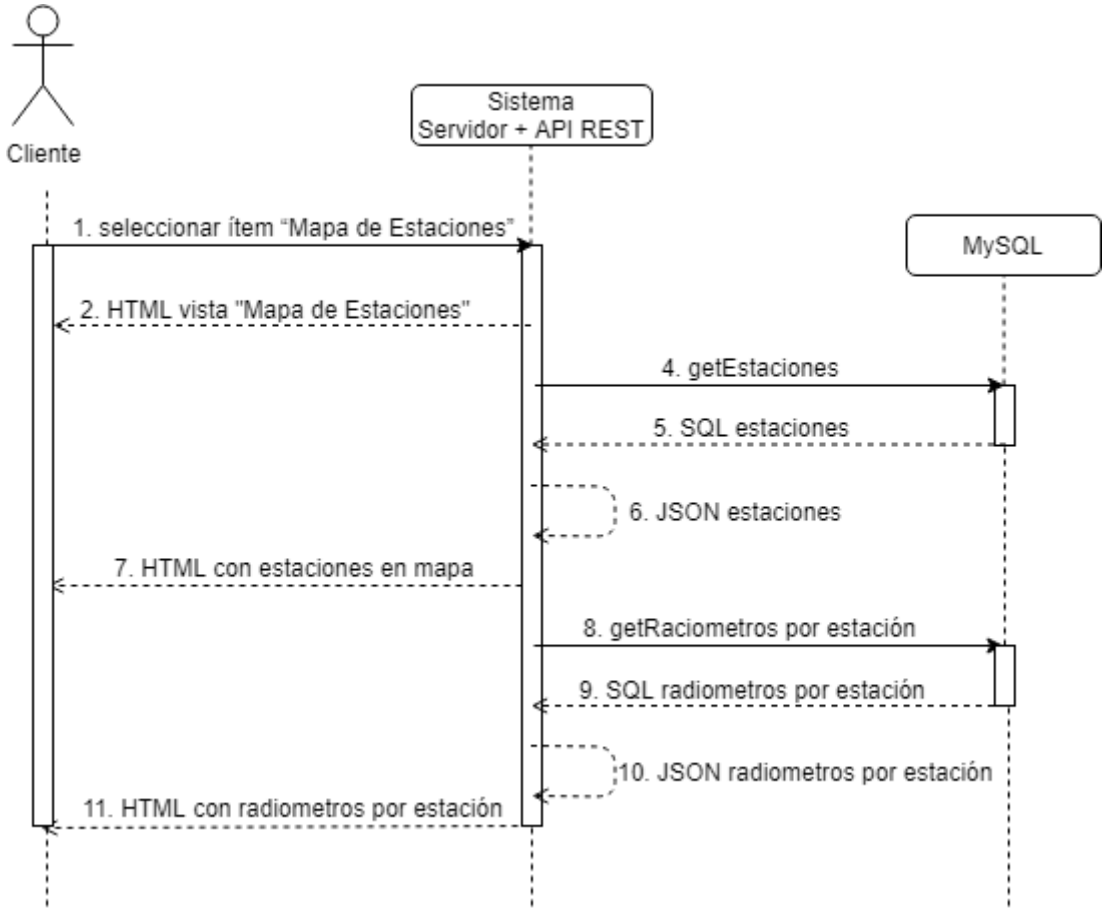


Ilustración 43: Diagrama de secuencia para CU-08

A continuación, se va a representar el flujo de mensajes entre los componentes de los módulos del sistema. A diferencia de lo anterior, ahora en vez de tener actores del sistema, tenemos componentes de módulos. Es necesario definir previamente cuáles son los módulos que dan lugar a la aplicación con sus respectivos componentes:

- **Módulo de Datos**: Es el módulo correspondiente a la vista graficar datos. En él se permite representar gráficamente las mediciones que hay almacenadas en la base de datos respetando una serie de parámetros que previamente han sido establecidos por el usuario. Este módulo está compuesto por los siguientes componentes:
    - **Componente Datos:** Es el componente principal del módulo. En él se realizan las llamadas a servicios para obtener los datos que posteriormente hay que graficar. También contiene el elemento html correspondiente a la gráfica. A su vez, es padre de los componentes Form e Info.
    - **Componente Form:** Componente hijo del componente datos. Este está formado por una serie de inputs que permiten establecer los parámetros a través de los cuales se va a filtrar a la hora de realizar peticiones de mediciones. Tener en cuenta que los inputs para seleccionar un id de radiómetro o un tipo de medición, se van a rellenar a partir de las respuestas de las llamadas a servicios que traen la información correspondiente.
      Como salidas hacia el componente padre existen 3. Una primera será cuando el usuario cambia la opción del tipo de medición por el que filtrar. Una segunda cuando el usuario indique que quiere ver la información adicional acerca de la medición seleccionada por el usuario. Por último, devolverá los filtros que ha escogido el usuario tras pulsar el botón de graficar datos.
    - **Componente Info:** Componente hijo del componente datos. Este únicamente permite al usuario ver la información acerca de cada uno de los tipos de mediciones existentes en la base de datos y a través de los cuales el usuario puede filtrar a la hora de representar los datos. Como entrada recibe un objeto con la información que va a tener que mostrar acerca del tipo de medición seleccionada por el usuario. Como salida, envía un mensaje al componente padre indicando que el usuario ha decidido ocultar la información adicional.

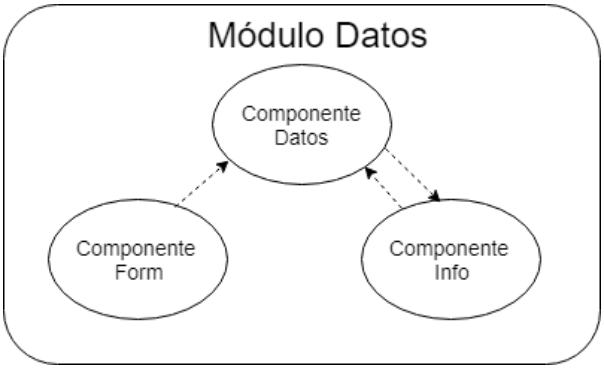


Ilustración 44: Componentes del módulo de datos

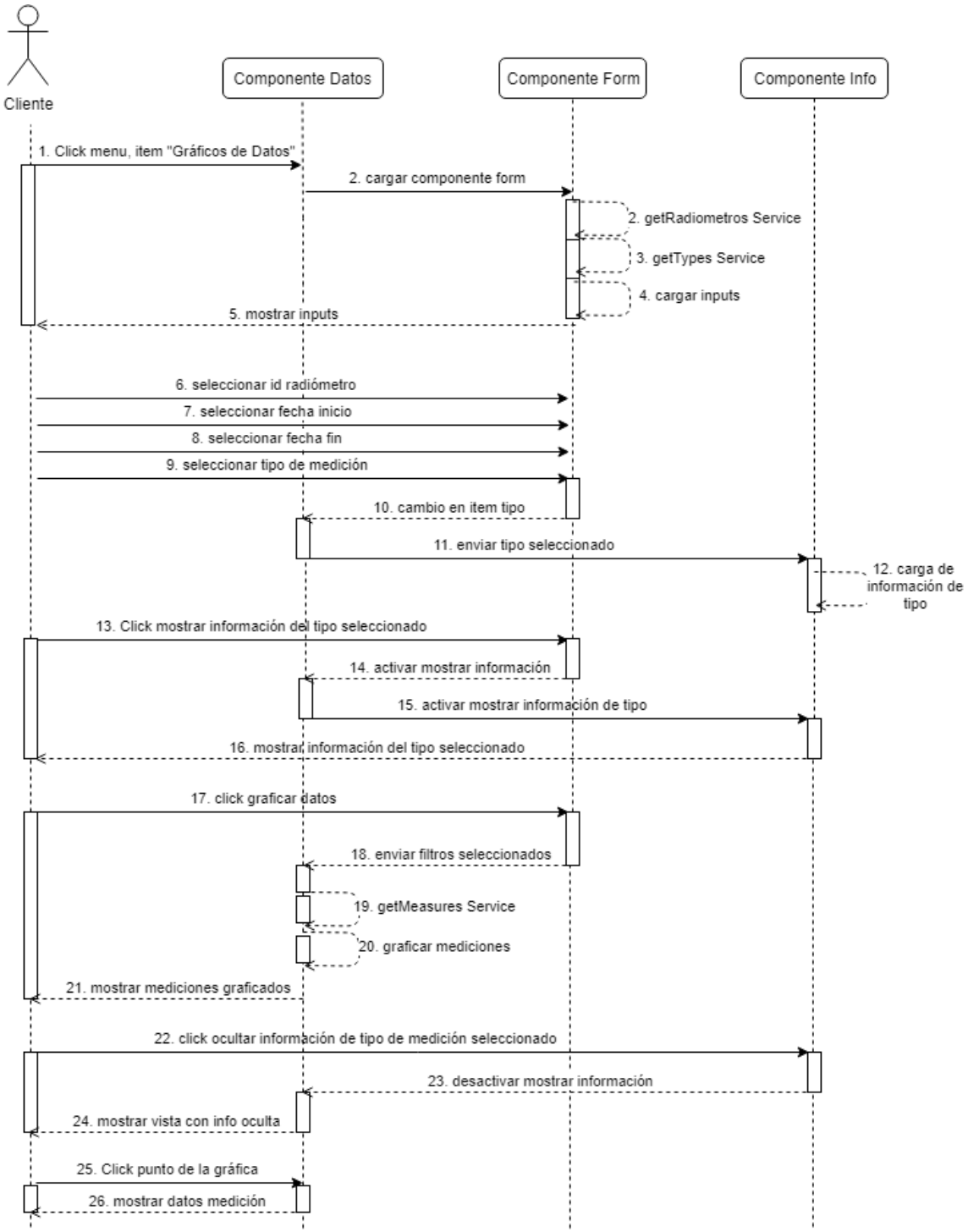


Ilustración 45: Diagrama de secuencia para Módulo de Datos

- **Módulo Estaciones:** Es el módulo correspondiente a la funcionalidad de representar en un mapa la localización de las estaciones radiométricas junto con otros datos de interés como el nombre de la estación, sus coordenadas y los radiómetros instalados en cada una de ellas. Este módulo únicamente cuenta con un componente.
    - **Componente Estaciones:** Es el componente principal y único del módulo. En él se realizan las llamadas a servicios correspondientes para traer de la base de datos la información de las estaciones y sus respectivos radiómetros.

Ilustración 46: Componentes del módulo de estaciones

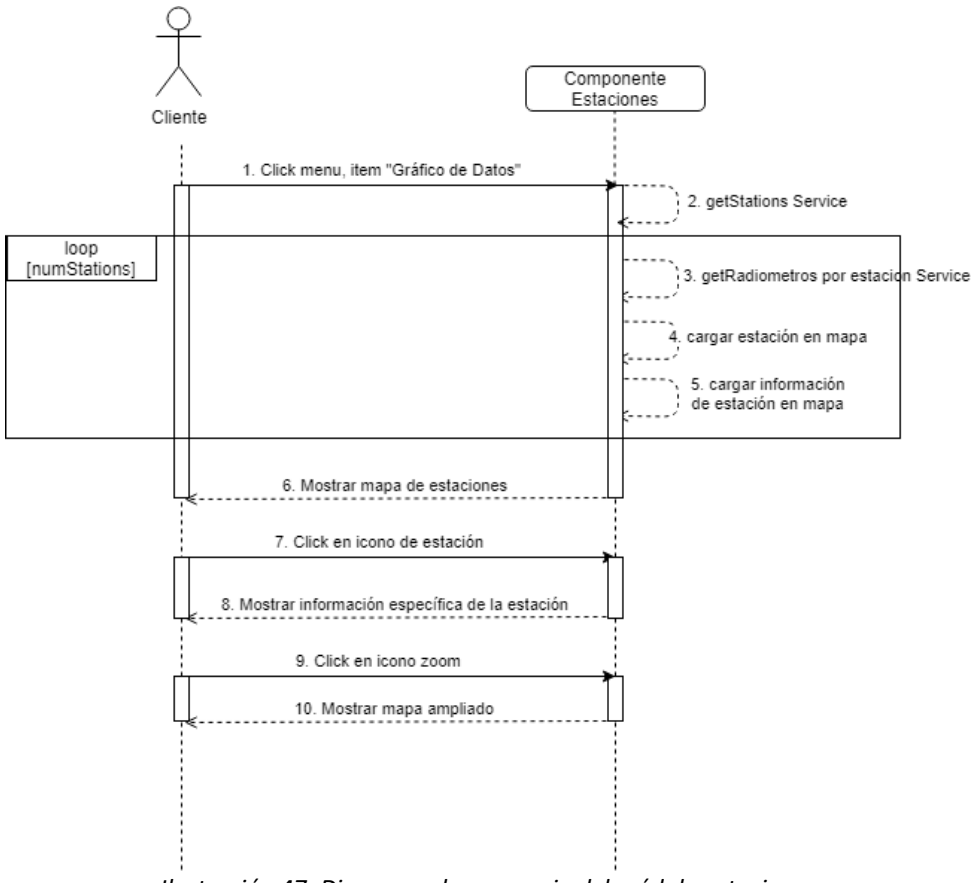


Ilustración 47: Diagrama de secuencia del módulo estaciones

4.2.4. Diseño de la interfaz

Este apartado recoge todos los aspectos relacionados con la distribución gráfica de los componentes en el dispositivo sobre el que se va a consumir la aplicación. El principal objetivo es que la interfaz sea intuitiva y fácil de utilizar. Al tener únicamente dos vistas no se han contemplado problemas en este sentido. Por otro lado, en cuanto a la navegabilidad, estará siempre disponible un menú horizontal en el que se puede cambiar de una ruta a otra de la aplicación.

En principio, la aplicación está orientada a desplegarse sobre dispositivos con pantallas grandes en un entorno web debido a que a medida que el tamaño de la pantalla se va reduciendo, el componente del gráfico de datos se iría reduciendo hasta llegar a un punto en el que difícilmente se puede apreciar la información de la misma.

A continuación, se detalla el diseño que se ha seguido para el diseño de las diferentes vistas de la aplicación junto con sus componentes internos.

Se ha tenido en cuenta que, para cada una de las vistas de la aplicación, se ha seguido una estructura formada por componentes estáticos, así como la cabecera con el título y el icono, el menú general y el pie de página. Por otro lado, también cuenta con un espacio reservado para el contenido dinámico que irá variando en función de la ruta a la que se acceda desde el navegador.

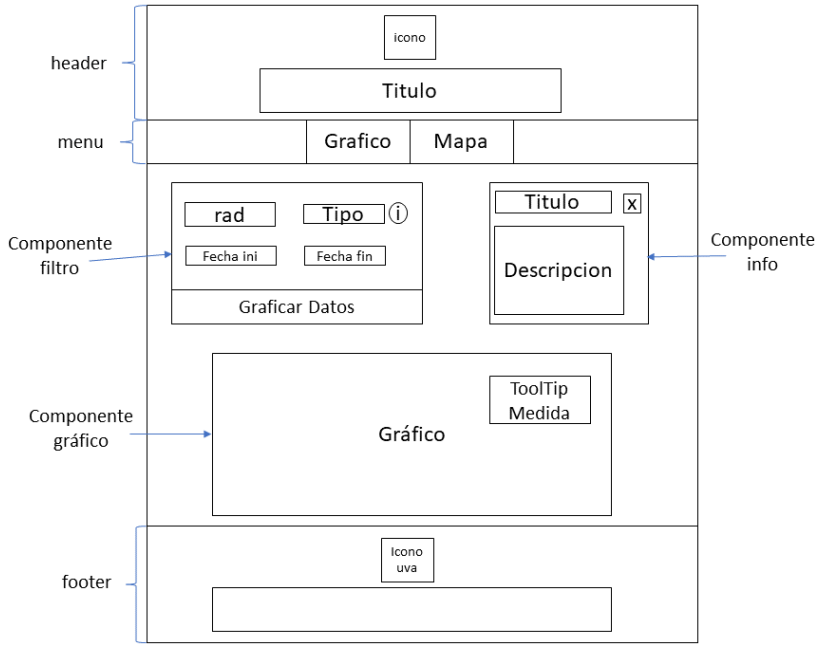
Gráficos de Datos	
Descripción	Es la pantalla que recoge toda la funcionalidad que permite generar gráficas de mediciones. Todo esto a partir de unos parámetros que previamente ha escogido el usuario a través del componente filtro. También existe la posibilidad de hacer visible otro componente adicional que permite ver la información relativa a los diferentes tipos de mediciones que hay disponibles a la hora de hacer el filtrado por tipo.
Ruta	localhost:4200/medidas
Activación	1- Punto de inicio de la aplicación 2- Desde el ítem del menú general.
Diseño	
Eventos	• Acceso a la vista de Mapa de Estaciones • Hacer visible la información sobre el tipo de medición • Cambiar el valor del input Tipo de tipo select • Click en el botón de graficar datos para generar gráfica • Mostar tooltip de la gráfica al colocarse encima de un punto de esta.

Tabla 78: Interfaz de usuario de la vista Gráficos de Datos

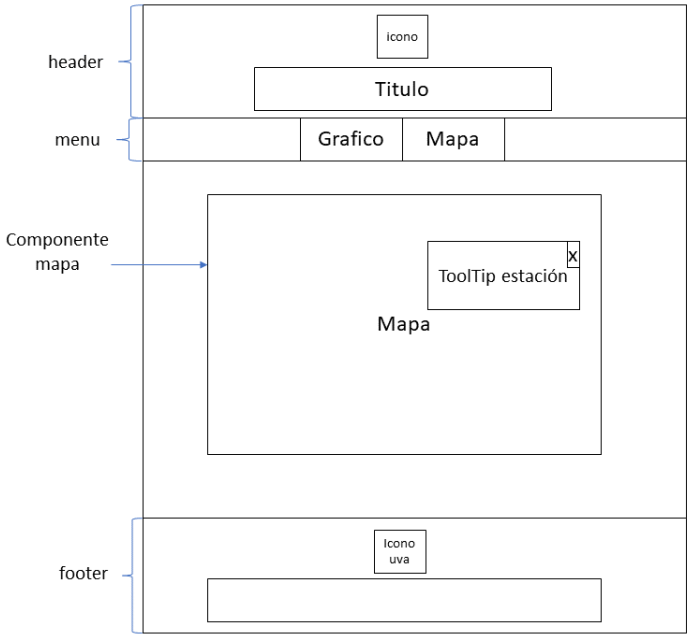
Mapa de Estaciones	
Descripción	Es la pantalla que recoge toda la funcionalidad que permite generar un mapa donde se puede visualizar la localización exacta de cada una de las estaciones radiométricas registradas en la base de datos. También se puede ver la información específica de cada una de ellas, así como el nombre, las coordenadas y los radiómetros instalados en ella.
Ruta	localhost:4200/estaciones
Activación	1- Desde el ítem del menú general.
Diseño	
Eventos	• Acceso a la vista de Mapa de Estaciones • Click sobre zoom + • Click sobre zoom – • Mover posición del mapa • Click sobre estación, genera tooltip para visualizar datos de la estación • Click para cerrar tooltip del map.

Tabla 79: Interfaz de usuario de la vista Mapa de Estaciones

4.3. Implementación

Para la implementación de la aplicación web se han empleado diferentes tecnologías teniendo en cuenta las diferentes capas de la arquitectura lógica. Se ha seguido el cuadro de herramientas basado en un conjunto de software utilizado para el desarrollo web de páginas dinámicas basados en *JavaScript*. A continuación, se clasifican las tecnologías utilizadas en front-end *Angular5* y servidor *Express* con *NodeJS*.

4.3.1. Angular5

4.3.1.1. Tecnologías previas

Para el desarrollo del lado cliente se ha recurrido al framework *Angular5* con sus respectivas tecnologías. Antes de entrar más en detalle en *Angular*, se van a introducir dichas tecnologías que le dan funcionalidad:

- ***HTML5***

Es el lenguaje más importante y conocido para la creación de páginas web, dándoles una estructura y un contenido. Se basa en etiquetas, que son breves instrucciones de apertura y cierre.

En concreto, *HTML5* es la última versión de *HTML*. Esta incluye una nueva estructura que facilita la forma de generar contenido con estructuras estándar como `<article>`, `<footer>`, `<header>`, `<nav>` y `<section>` que facilitan mucho la construcción de la estructura de la aplicación en cuanto a la adición de información.

También incluye mejoras sobre la etiqueta `<input>`, pudiendo indicar si es una URL, un email, un entero…

Por otro lado, cabe destacar que al ser la última versión incluirá la corrección de errores en etiquetas y comportamientos que podrían afectar al correcto funcionamiento de la aplicación.

- ***CSS3***

Al igual que *HTML* no es considerado un lenguaje de programación, sino un lenguaje de estilo que separa el contenido de la forma y permite un mayor control sobre la apariencia y el estilo de la aplicación web. En este caso, se ha utilizado *CSS3*, la última versión de *CSS*, incluye nuevos efectos, con una mayor facilidad de aplicación. Permite conseguir que la parte visual de nuestros documentos sean más llamativos, ya sea incluyendo mejoras a la hora de diseñar bordes, incluir animaciones o transiciones en los elementos HTML… Esta última versión también corrige los errores de las versiones antiguas.

- ***JavaScript***

Es un lenguaje de programación orientado a objetos que es utilizado principalmente para la creación de webs dinámicas del lado cliente y, en ocasiones, del lado servidor (como se va a ver más adelante).

El aumento de las *APIs* lo ha convertido en uno de los lenguajes de programación más utilizados en el mundo.

La versión utilizada en el desarrollo de la aplicación es ***ES6*** (ECMAScript), la última versión de *JavaScript* que incluye nuevas propiedades como el uso de variables con ámbito (`let`), clases (`class`), módulos (`import` y `export`) y funciones arrow (`() => { }`).

4.3.1.2. Características *Angular5*

Una vez introducidas las tecnologías anteriores, ya podemos entrar en detalle en la definición de las características principales del framework *Angular5*.

Utilización de componentes

Se trata de un framework que extiende directamente las funcionalidades *HTML* y *CSS* utilizando **componentes**. Esto permite la reutilización de estos, permitiendo además interconexiones entre ellos. Cada componente cuenta con su propia interfaz de entrada y salida e incluye inyección de dependencias para añadir funcionalidades adicionales en cada uno de ellos. El ciclo de vida de los componentes está muy bien definido, permitiendo establecer el comportamiento esperado en el momento de su carga, actualización o destrucción.

Patrón arquitectónico basado en MVC

Por otro lado, utiliza un patrón arquitectónico basado en Modelo Vista Controlador (**MVC**) o similares, que permite reutilizar código y separar conceptos.

- Vistas: Será la representación de los datos o la información, es decir, el código del interfaz de usuario, básicamente el *DOM*/*HTML*/*CSS*.
- Controladores: Se encargan de la lógica de la aplicación, incluyendo los servicios para mover datos contra servidores o memoria local en *HTML5*.
- Modelo de la vista: El modelo es la estructura lógica que subyace a los datos. Asociado a este modelo, se define el `scope` que es el encargado de detectar los cambios en el modelo y proporciona el contexto a las plantillas.

Creación de SPAs

Un aspecto muy relevante es que permite crear **Simple-Page Aplications** (*SPAs*). Esto hace que el proceso de carga de la aplicación se realice completamente en una sola página y, de esta manera, aproxima más la aplicación web a una de escritorio. Hace uso de la asincronicidad para evitar bloqueos que supongan una carga adicional al servidor y, por lo tanto, permite escalar la aplicación y limitar la dependencia con el servidor.

A continuación, en la *Ilustración 48* e *Ilustración 49* se puede ver la diferencia entre el ciclo de vida de una página web tradicional y el de un SPA:

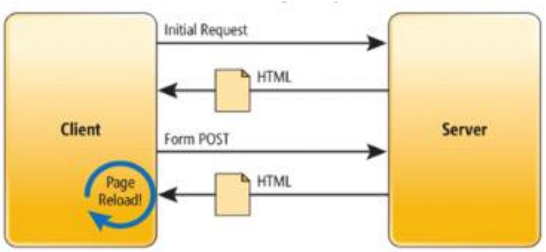


Ilustración 48: Ciclo de vida de una aplicación tradicional

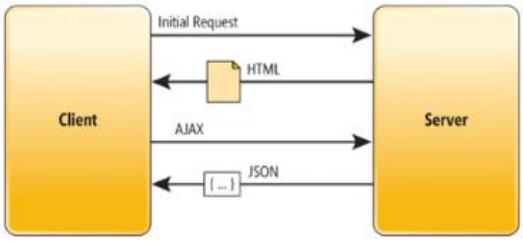


Ilustración 49: Ciclo de vida de una SPA

Programación en *ES6* + *TypeScript*

Angular5 utiliza *TypeScript*. Es un lenguaje de programación basado en *JavaScript*, en concreto parte de la versión *ES6*. Se trata de un lenguaje escrito sobre otro lenguaje que ofrece grandes beneficios como por ejemplo el tipado y las anotaciones entre otros.

Cuando se compila *TS*, éste convierte su código en *JS* común (transpilación), lo que significa que, si el navegador está basado en *JS*, este nunca llegará a saber si el código original fue realizado con *TS* y ejecutará el *JS* como lenguaje original.

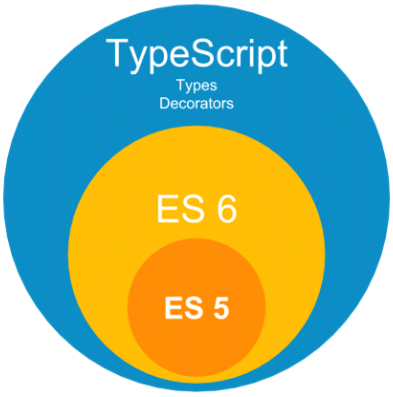


Ilustración 50: Superset de ES6

4.3.1.3. Angular-cli

***Angular** **C**ommand **L**ine **I**nterface* es una herramienta oficial para la gestión de proyectos. Principalmente ofrece comandos para todo el ciclo de desarrollo de una aplicación *Angular*:

- Generación de la estructura del proyecto inicial (basado en Webpack)
  - `ng new proyecto`
- Herramientas de generación de código
  - `ng generate module`
  - `ng generate component`
  - `ng generate provider`
- Modo desarrollo
  - Compilado automático de TypeScript
  - Arranque de servidor web
    - `ng serve`
  - Actualización del navegador (Live Reloading)
- Testing
- Construcción del proyecto para distribución
  - `ng build`

4.3.1.4. Estructura de la aplicación

A nivel general estos son los directorios principales del proyecto que parten de **`src`**:

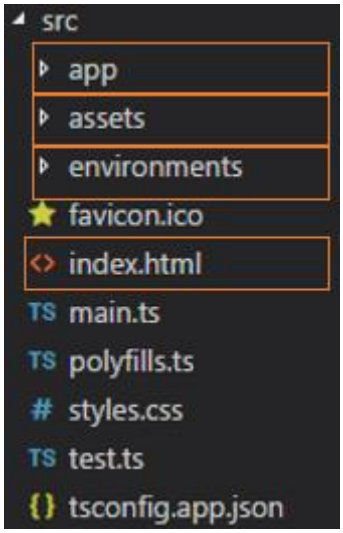


Ilustración 51: Directorios principales de la aplicación Angular

- **`app`**: contiene los módulos asociados a la funcionalidad de la aplicación.
- **`assets`**: contiene todos los recursos estáticos de la aplicación, así como imágenes de fondo de la aplicación, el icono de esta…
- **`environments`**: está formado por distintos ficheros que permiten crear propiedades de configuración. Depende de los parámetros indicados al hacer el `build`.
  - `dev` (entorno de desarrollo)
  - `prod` (entorno de producción)
- **`index.html`**: fichero raíz del proyecto. De él parte el *DOM* del proyecto.

A continuación, se describen los elementos que se encuentran dentro del módulo **`app`**:

- **Módulo *`app`***

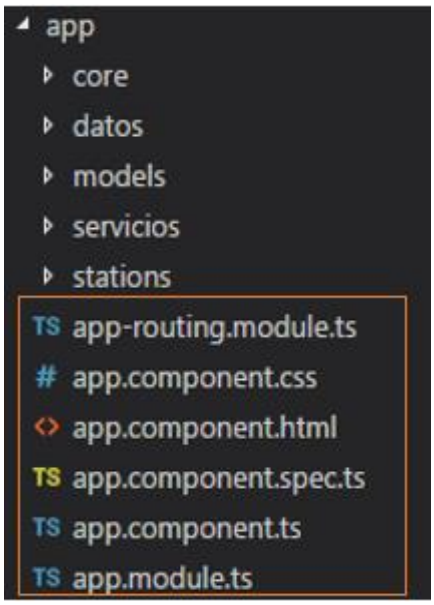


Ilustración 52: Módulo app

1) **`app.module.ts`**: En él se importan el resto de módulos que utiliza la aplicación, así como el fichero de rutas que carga cada uno de los módulos. También se importan los servicios que utiliza la aplicación.

2) **`app-routing.module.ts`**: se definen las rutas generales para cargar los diferentes módulos de la aplicación a través de la URL.

3) **`app.component.html`**: contiene el *html* que da lugar al *DOM* de la aplicación, incluyendo los componentes de los módulos que se van a cargar.

4) **`app.component.css`**: contiene el estilo del `app.component.html`.

5) **`app.component.ts`**: controlador que contiene la lógica, funciones, variables…

- **Módulo `core`**

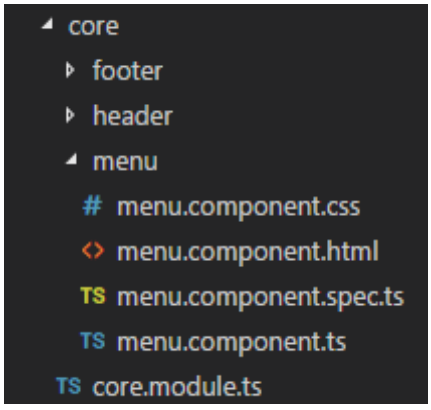


Ilustración 53: Módulo core

El módulo está formado por los componentes `header` (cabecera), `menu` (menú) y `footer` (pie). Estos componentes son incluidos en el `app.component.html` (*Ilustración 54*) de tal manera que establecen la estructura estática de la aplicación. Como se puede comprobar, cada componente tiene su *html*, *css* y *ts* independiente de tal manera que cada uno de ellos tiene sus propios elementos *html*, su propio estilo para ellos y su propia lógica.

```
<rad-header></rad-header>
<rad-menu></rad-menu>
<div class="content mt-4">
    <router-outlet>

    </router-outlet>
</div>
<div class="footer">
  <rad-footer></rad-footer>
</div>
```

Ilustración 54: Incluir componentes en app.component.html

Como se aprecia en la *Ilustración 54*, la estructura de la aplicación ya queda definida y será inmutable. Por otro lado, dentro de las etiquetas `router-outlet` es donde estará alojado el contenido dinámico, es decir, dependiendo de la ruta recogida en la URL, dentro de esa etiqueta habrá cargado un módulo u otro con sus respectivos componentes. En este caso, esos módulos serán el de **datos** o el de **estaciones**.

- **Módulo `datos`**

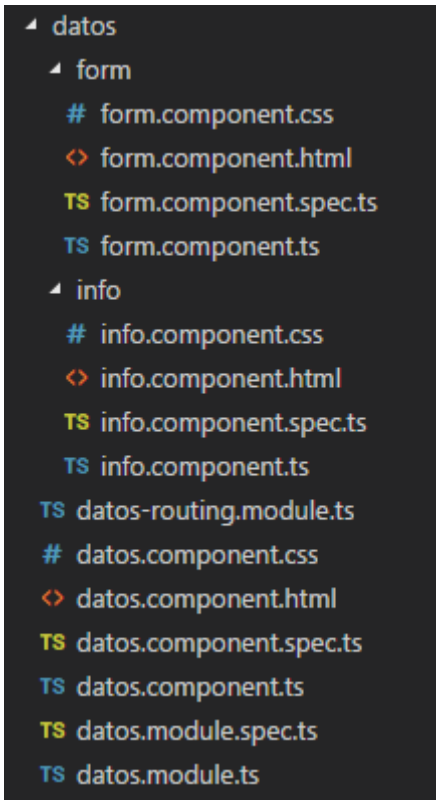


Ilustración 55: Módulo datos

6) **componente `datos`**: en él se cargan los componentes `form` e `info`. Este componente será el padre de ambos, por tanto, ellos son hermanos y para comunicarse entre ellos lo harán por medio de este componente. En él, se realizan las llamadas a los servicios que traen las mediciones que cumplen con los parámetros establecidos por el usuario en el componente `form`. Entonces, tanto los componentes como los servicios deben estar importados dentro de `datos.module.ts`.

7) **componente `form`**: contiene un formulario donde se establecen los filtros que el usuario desee para traer mediciones de la base de datos. También hace uso de servicios para obtener los listados de *ids* de radiómetros y tipos de mediciones que hay en la base de datos. Estos servicios también estarán incluidos en el *provider* del módulo (`datos.module.ts`).
Como salida devuelve un objeto con los filtros que ha seleccionado el usuario al dar a un botón de graficar.

- **componente `info`**: recibe como entrada un objeto de tipo Type que contiene el título, el nombre y la descripción del tipo de medición que el usuario ha seleccionado en el componente form. Emite como salida un booleano indicando si el usuario desea ocultar dicho componente o no, dentro del componente padre de datos.

En la *Ilustración 56* se puede ver como se despliegan los componentes `form` e `info` dentro del componente padre `datos`. Nótese que las entradas hacia un componente se registran con `[]` y las salidas desde el mismo con `()`.

```
<div class="col-md-6" style="margin-left: 10%">
  <rad-form
    (outChangeForm)="onChangeForm($event)"
    (outShowInfoStatus)="showInfo($event)"
    (outTypeSelected)="getTypeSelected($event)"
  >
  </rad-form>
  <div *ngIf="dateInvalid" class="alert alert-danger">{{ errorMsg2 }}</div>
  <div *ngIf="responseEmpty" class="alert alert-warning">{{ errorMsg }}</div>
</div>

<div [style.opacity]="showStatus ? '1' : '0'" [style.visibility]="showStatus ? 'visible' : 'hidden'">
  <rad-info [oTypeMeasureSelected]="oTypeSelected" (outCloseInfo)="showInfo($event)"></rad-info>
</div>
```

Ilustración 56: Componentes form e info dentro del componente padre datos

- **Módulo `stations`**

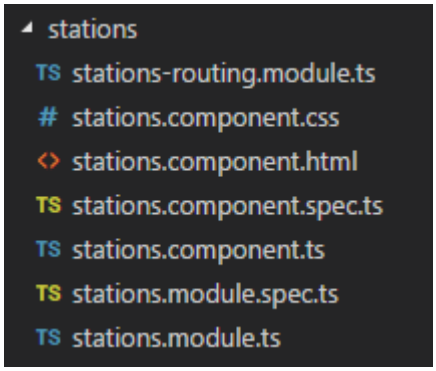


Ilustración 57: Módulo stations

- **componente `stations`:** Es el único componente del módulo. Hace uso de varios servicios por lo que es necesario importarlos en el *provider* del `stations.module.ts`.

- **Servicio**

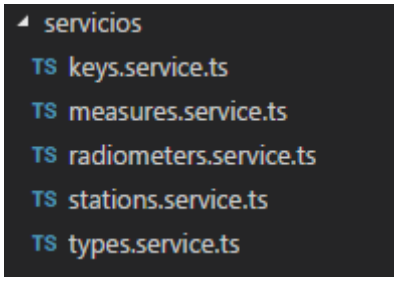


Ilustración 58: Servicios de la aplicación

- **`measures.service.ts`:** servicio que recibe todas las medidas de la base de datos que respetan los filtros establecidos por el usuario.

- `radiometers.service.ts`: por un lado recibe de la base de datos un listado de los radiómetros registrados. Por otro lado, recibe los radiómetros que hay instalados en una estación concreta. Esta será pasada por parámetro a la hora de llamar al servicio.

- `stations.service.ts`: obtiene de la base de datos el listado de las estaciones con radiómetros instalados.

- `types.service.ts`: obtiene de la base de datos el listado de los tipos de mediciones existentes en ella.

- **Models**

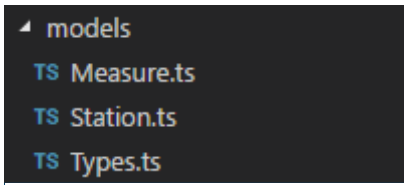


Ilustración 59: Modelos de la aplicación Angular

Se trata de clases que contienen el formato que van a guardar los datos al ser traídos a través de los servicios. En otras palabras, servirán para mapear los datos.

4.3.1.5. Módulos externos

- **Componente de gráfica lineal**

Para la creación del componente referente a la gráfica lineal de tiempo se ha utilizado la librería **`ng2-charts chart.js`** que permite la creación de este tipo de componentes.

Es necesario que, una vez instalados los módulos necesarios, `ChartsModule` sea importado en el módulo en que se vaya hacer uso de la gráfica, en este caso en el módulo de datos (`datos.module.ts`).

Una vez importado, se añade el *html* del componente del módulo donde se quiere desplegar, en este caso en `datos.component.html`.

```
<canvas baseChart #baseChart="base-chart"
  [chartType]="'line'"
  [datasets]="chartData"
  [labels]="chartLabels"
  [options]="chartOptions"
  [legend]="false"
  [colors]="myColors">
</canvas>
```

Ilustración 60: Despliegue de la gráfica

Como se aprecia en la *Ilustración 60*, el componente recibe numerosas entradas:

`[chartType]`: indica el tipo de gráfica, en este caso es lineal.

`[datasets]:` recibe el array de datos que se van a introducir en el eje Y de la gráfica. En este caso serán los valores de las mediciones que irán variando dependiendo de los resultados obtenidos de las llamadas al servicio.

`[labels]`: recibe el array de *String* que se van a introducir en el eje X de la gráfica. En este caso serán las fechas de las mediciones obtenidas por el servicio.

`[options]`: una serie de opciones adicionales, como si se quiere un tamaño de gráfica responsivo, el título que tendrá la misma, el estilo de la línea y los puntos que forman la gráfica y la gestión de la generación de *tooltips* que ofrecen la información de cada punto de la gráfica.

`[legend]`: no interesa leyenda puesto que es una única línea la que se va a representar.

`[colors]`: establece el estilo de la gráfica en cuanto a colores.

- **Componente mapa**

Para la integración de *Google-maps* dentro de la aplicación se ha instalado el módulo AGM. Una vez instalado, se importa en el módulo en el que se desea desplegar el mapa, en este caso en el de *stations* (`stations.module.ts`). También es necesario incluir un *apiKey* de Google que se encarga de gestionar el flujo de peticiones del usuario concreto que tiene asignada una Key específica.

Con el módulo ya importado, lo siguiente es incluir el componente en el *html* (`stations.component.html`).

```
<agm-map
  [latitude]="lat"
  [longitude]="lng"
  [zoom]="zoom"
  [minZoom]="minZoom"
  [scrollwheel]="false"
  [mapTypeId]="'hybrid'">

  <agm-marker
    *ngFor="let s of aStations"
    [latitude]="s.latitude"
    [longitude]="s.longitude">
```

Ilustración 61: Despliegue de mapa 1

8) **agm-map**: recibe como entrada la latitud y longitud donde estará centrado el mapa, el zoom desde el que se parte, el zoom mínimo aceptado, la inhabilitación del zoom mediante el scroll y el tipo de mapa, que en este caso es híbrido.

9) **agm**-marker: Se encarga de introducir las etiquetas en el mapa. Éstas representan la localización exacta de cada una de las estaciones a través de sus coordenadas.

Dentro del componente `agm-marker` irá el componente `agm-info-window` que hace referencia al *tooltip* que se genera al pulsar sobre cualquier *marker* que hace referencia a una estación. Ver el código en la *Ilustración 62*:

```
<agm-info-window>
  <label for="tabla">Información de la estación</label>
  <table name="tabla" >
    <tr>
      <th>Nombre </th>
      <th>{{s.name}}</th>
    </tr>
    <tr>
      <th>Coordenadas </th>
      <th> ({{s.latitude}}, {{s.longitude}}) </th>
    <tr>
      <th>Radiometros instalados</th>
      <th>
        <p style="margin: 0px;" *ngFor="let r of s.radiometers">· {{r}}</p>
      </th>
    </tr>
  </table>
</agm-info-window>
```

Ilustración 62: Despliegue de mapa 2

Como se puede observar, por cada *marker*, se generará una ventanita que muestra la información referente a la estación seleccionada en formato tabla. Se ofrece el nombre de la estración, sus coordenaas y un listado de los radiómetros que tiene instalado.

4.3.1.6. Otras librerías adicionales

- ***BootStrap***

Es el framework de *HTML*, *CSS* y *JS* más conocido para el desarrollo de front-end. Permite crear de forma sencilla interfaces de usuario responsivas.

Para utilizarlo primero hay que instalarlo para que se incluyan las dependencias *Bootstrap*. Una vez hecho esto, se incorpora el *CSS* en el fichero de configuración de angular-cli (`angular-cli.json`)

- ***FontAwesome***

Es una librería que proporciona elementos de interfaz basándose únicamente en la incorporación de *CSS*. Básicamente, esta librería se ha utilizado para incluir iconos de forma sencilla.

Para utilizarlo es necesario instalar las dependencias y luego se importa como *CSS* en `styles`.

```
"styles": [
  "styles.css",
  "../node_modules/font-awesome/css/font-awesome.css"
],
```

Ilustración 63: Font Awesome en angular-cli

4.3.2. NodeJS + Express

4.3.2.1. NodeJS

Node es un entorno que trabaja en tiempo de ejecución que permite el desarrollo de toda clase de herramientas de lado servidor en *JavaScript*. Esta ejecución en tiempo real se centra en ser utilizada fuera del contexto de aplicación web, es decir, ejecutarse directamente en un servidor.

Se puede crear de forma sencilla un servidor web básico que escuche cualquier clase de peticiones en una URL concreta. El problema existe cuando se quiere añadir el manejo específico de diferentes verbos *HTTP* (*GET*, *POST*, *DELETE*…) o gestionar de forma separada diferentes rutas. En ese caso hay que escribir demasiado código. Por esta razón se ha utilizado el framework *Express*.

4.3.2.2. ExpressJS

Es el framework más popular de *Node* y ofrece mecanismos para crear de forma sencilla *APIs REST* que soportan cualquier petición *HTTP*. Esta es la principal razón por la que se ha utilizado este framework para el lado servidor. Pero también incluye otras funcionalidades como la integración de motores de renderización de vistas para generar respuestas mediante la introducción de datos en plantillas (*JADE*). Por otro lado, permite gestionar el puerto por el que se quiere acceder al servidor y también ajustar las plantillas que se quieren utilizar para renderizar las respuestas. Finalmente, añade procesamiento de peticiones middleware que permite gestionar mejor las peticiones añadiendo más seguridad al *API*.

4.3.2.3. Estructura de la aplicación

A continuación, se muestra cómo Express organiza los directorios y ficheros para tener como resultado un *API REST*:

- **`app.js`**: sería el fichero raíz, es en el que se establecen las diferentes URLs a las que se las hace peticiones y sus correspondientes funciones que son activadas cuando el servidor detecta una petición a una determinada URL. Desde este archivo también se gestionan los errores en caso de que la petición dé un resultado vacío o en caso de que la URL no exista. También cabe destacar que es aquí donde se dan permisos a las aplicaciones web para que le puedan hacer peticiones.

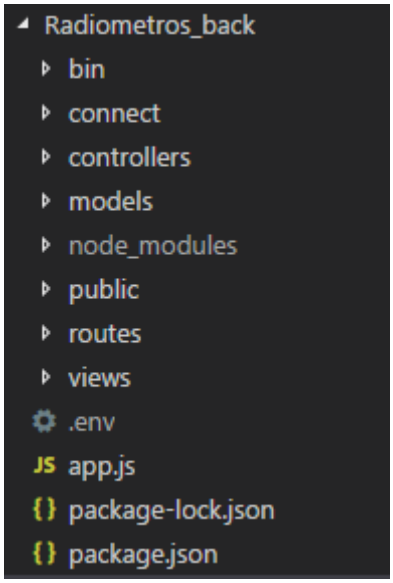


Ilustración 64: Estructura general de la aplicación ExpressJS

- **`routes`**: En el directorio se encuentran los archivos que contienen los métodos *HTTP* (*GET*, *POST*…). Sirven para reenviar las peticiones recibidas desde `app.js` a las funciones de controlador relativas.
- **`controllers`**: los archivos de este directorio se encargan de mediar entre `routes` y `models`, pudiendo formatear el resultado de la petición en este punto.
- **`models`**: los ficheros de este directorio son los que se encargan de conectar a la base de datos y hacer la lógica necesaria para obtener la información que el cliente desea recibir.
- **`connect`**: los archivos de este directorio se utilizan para establecer conexiones con sistemas externos como por ejemplo la base de datos.
- **`.env`**: archivo que contiene las variables de entorno para la conexión a la base de datos. Será utilizado por el módulo `dotenv`.

A continuación, se adjuntan capturas de código para seguir la traza para un ejemplo en el que se hace una petición *GET* a la ruta */radiometros*:

1) Partiendo desde `app.js`

En primer lugar, el servidor comprueba si el cliente desde el que se hace la petición a la URL tiene permisos. En este caso sí que tendrá acceso ya que suponemos que esta se hace desde la aplicación cliente `http://localhost:4000` (*Ilustración 65*)

```
var originsWhitelist = ['http://localhost:4200'];
var corsOptions = {
  origin: function(origin, callback){
        var isWhitelisted = originsWhitelist.indexOf(origin) !== -1;
        callback(null, isWhitelisted);
  },
  credentials:true
}
```

Ilustración 65: Permitir peticiones a la app Angular

Seguidamente comprueba si está la URL solicitada entre las registradas en el servidor. Como se puede observar, sí que existe por lo que se carga el módulo `radRouter` correspondiente con `./routes/radiometer.router` (*Ilustración 66*).

```
var radRouter = require("./routes/radiometer.router");
app.use('/radiometros/measures', measuresRouter);
app.use('/radiometros/stations', stationsRouter);
app.use('/radiometros/types', typesRouter);
app.use('/radiometros/keys', keysRouter);
app.use('/radiometros', radRouter);
```

Ilustración 66: Rutas del API REST

Si se hubiera dado el caso de que no existe la URL, el error habría sido gestionado por el siguiente código que renderiza el mensaje de error en una página *HTML* (*Ilustración 67*).

```
// error handler
app.use(function(err, req, res, next) {
  // set locals, only providing error in development
  res.locals.message = err.message;
  res.locals.error = req.app.get('env') === 'development' ? err : {};

  // render the error page
  res.status(err.status || 500);
  res.render('error',{ message: err.message || 'Unknown error' , status: err.status, error: err});
});
```

Ilustración 67: Manejador de errores

2) Se accede a routes/radiometer.router.js

```
var radController = require("../controllers/radiometer.controller");
```

```
router.get('/', (req, res, next) => {
    radController.getRad()
    .then(response => {
      console.log('-->RAD BIEN')
      if (!response.length) {
        const err = new Error ("Radiometers not found")
        err.status = 404;
        next(err)
      }
      else
        res.json(response)
    })
    .catch(err => {
      console.log('-->RAD MAL')
      next(err)
    })
});

module.exports = router;
```

Ilustración 68: radiometer.router.js

Se detecta que la petición es un *GET* por lo que se lanzaría la función `getRad` que se encuentra en el controlador. Aquí entra en juego la asincronía (promesas) ya que el resultado se mostrará, ya sea error o contenido, cuando la función `getRad` del controlador haya acabado.

3) Se accede a controllers/radiometer.controller.js

```
var radModel = require ("../models/radiometer.model");
```

```
function getRad () {
    return radModel.getRad();
}
```

Ilustración 69: radiometer.model.js

Desde el router se llama a esta función y espera a que esta función del controller le devuelva el resulado, pero como se puede apreciar, esta función a su vez llama a otra función que se encuentra en el modelo.

4) Se accede a models/radiometer.model.js

A través de la clase `Database` que se encuentra en `../connect/Database.js`, se permite realizar peticiones a la base de datos mediante su método `query()`.

```
var Database = require('../connect/Database.js');
let database = new Database();

function getRad() {
    var query = 'SELECT r.id,r.ref,i.station FROM rad_radiometer r, rad_installation i WHERE r.id = i.rad'
    return database.query(query);
}
```

Ilustración 70: getRad() de radiometer.model.js

En este caso el método `getRad()` devuelve una promesa asíncrona ya que las peticiones a bases de datos se realizan de esa manera.

En resumen, en el `model` se genera una promesa que obtiene la información que el cliente solicita. Ésta es llamada por el `controller` y a su vez del `controller` es llamada desde `router`, que es donde se consume dicha promesa, y envía el resultado al cliente en el momento que la esta se ha resuelto.

5) Utiliza connect/Database.js

Es una clase que encapsula todo el proceso de conexión a la base de datos. Por un lado establece la conexión de forma segura a través de las variables de entorno que se encuentran en `.env`. Por otro lado, cuenta con un método `query()` que recibe una consulta y la clase se encarga de devolver una promesa, teniendo en cuenta si el resultado de la consulta es correcto o no.

```
var dotenv = require('dotenv').config();
const mysql = require( 'mysql' );
class Database {
    constructor( config ) {
        this.connection = mysql.createConnection({
            host: process.env.DB_HOST,
            user: process.env.DB_USER,
            password: process.env.DB_PASS,
            database: process.env.DB_DATABASE
    });}
    query( sql, args ) {
        var jsonResponse = '';
        return new Promise( ( resolve, reject ) => {
            this.connection.query( sql, args, ( err, result, fields ) => {
                if ( err )
                    return reject(new Error('Problema en la conexión a la base de datos'));
                jsonResponse = JSON.stringify(result)
                resolve(JSON.parse(jsonResponse))
            });
        } );
    }
```

Ilustración 71: Clase Database.js

4.3.2.4. Módulos externos

- ***Mysql***

Controlador de *node.js* para conectar con *mysql*. Para más información acerca de su uso e instalación acceder aquí.

- ***Cors***

Permite establecer una lista de los clientes a los que se les permite realizar peticiones al servidor. Para más información acerca de su uso e instalación acceder aquí.

- ***Dotenv***

Permite establecer variables de entorno para almacenar información como credenciales para conexiones a base de datos. Para más información acerca de su uso e instalación acceder aquí.

4.3.2.5. API REST

Un *API REST* es un servicio que nos provee de funciones que dan la capacidad de hacer uso de un servicio web externo dentro de la propia aplicación web. En este caso, se ha construido un *API REST* para acceder a la información de la base de datos `caelis` que nos resulte interesante para la construcción de la aplicación web. El resultado que ofrece este *API* es en formato *JSON* que será fácil de manipular desde la aplicación *Angular*.

Hay que tener en cuenta que la creación del *API* se ha realizado para que únicamente se hagan peticiones desde la aplicación web *Angular*, pero no se descarta que este pueda ser consumido en un futuro por otras aplicaciones.

A continuación, se listan las URLs mediante las que se puede acceder a diferente tipo de información de la base de datos:

GET http://localhost:3000/radiometros/measures

Obtiene todas las mediciones almacenadas en la base de datos. A esta petición se la pueden añadir parámetros para filtrar por:

- Filtrado por fecha concreta → `?date1=2017-07-15`
- Filtrado por periodo de tiempo → `?date1=2017-07-15&date2=2017-07-20`
- Filtrado por clave → `?key=AVG`
- Filtrado por tipo de medición → `?type=SW`
- Filtrado por id de radiómetro → `?rad=090309`
- Combinación de todos ellos → `?rad=090309&date1=2017-07-15&date2=2017-07-20&key=AVG&type=SW`

Ejemplo respuesta:


```
0:
    key_:       "AVG"
    date_:      "2017-07-14 00:01:00"
    filename:   "/rad_achieve/correct/090309/2018/116.rad"
    rad:        "090309"
    value_:     -0.002
    type_:      "SW"
```


Ilustración 72: JSON respuesta a petición /measures

`GET http://localhost:3000/radiometros/stations`

Obtiene toda la información relacionada con las estaciones radiométricas que existen en la base de datos. A esta petición se la pueden añadir parámetros para filtrar por:

- Filtrar por nombre de estación → `?nameStation=Valladolid`

Ejemplo respuesta:


```
0:
    name:           "Valladolid"
    latitude:       41.6636
    longitude:      -4.70583
    elevation:      705
    aeronet_name:   "Valladolid"
    rolo_name:      "Valladolid"
  description:      "The photometers are situated on the roof of the new Faculty of Sciences,
                    University\r\nof Valladolid. The city of Valladolid (Spain) has 300.000
                    inhabitants. The Faculty is\r\nlocated on the north-eastern edge of the city."
    stable:         "yes"
    last_connection: "2017-12-20T08:37:51.000Z"
```


Ilustración 73: JSON respuesta a petición /stations

`GET http://localhost:3000/radiometros/types`

Obtiene el listado de los tipos de mediciones posibles que hay registradas en la base de datos.

Ejemplo respuesta:


```
0:
    name_:          "DIF"
    title:          "Radiación Solar Difusa"
  description:      "La radiación fotosintéticamente activa es la cantidad de radiación integrada del
                    rango de longitudes de onda que son capaces de producir actividad fotosintética.
                    Se encuentra en el umbral de los 400 a 700 nm dentro del espectro de radiación
                    visible. Conocer este tipo de longitudes de onda resulta útil para adecuar la
                    iluminación en entornos artificiales o para elegir mallas de sombreo aplicadas en
                    la obtención de la radiación solar difusa."
```


Ilustración 74: JSON respuesta a petición /types

GET http://localhost:3000/radiometros/keys

Obtiene el listado de los tipos de claves de mediciones posibles que hay registradas en la base de datos.

Ejemplo respuesta:

Ilustración 75: JSON respuesta a petición /keys

GET http://localhost:3000/radiometros/

Obtiene el listado de los radiómetros que se encuentran en la base de datos. Éstos deben estar instalados, ya que también se devuelve el nombre de la estación en la que se encuentra.

Ejemplo respuesta:

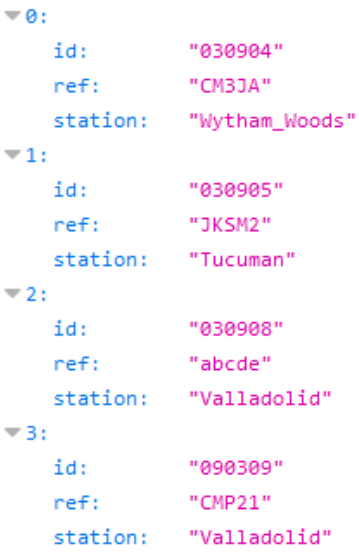


Ilustración 76: JSON respuesta a petición /radiometros

4.3.3. Herramientas para el desarrollo

A continuación, se lista las herramientas que han intervenido en el desarrollo la aplicación del sistema de visualización.

- ***Visual Studio Code***

Entorno de desarrollo creado por *Microsoft* para *Windows*, *Linux* y *masOS*. Ayuda a la escritura ágil de código con inclusión de *Plugins* adecuados a cada lenguaje. Cuenta con su propio terminal para ejecutar comandos y ver el resultado de la ejecución de programas. También cuenta con una consola de depuración y una funcionalidad para control de versiones del código fuente, donde en este caso se ha hecho uso de *Git*. Y *GitHub*.

Los Plugins que se han utilizado son los siguientes:

- ***Angular UI Bootstrap Snippets***

Creador: HerrHerrmann

Detalles: Ofrece atajos a la hora de crear componentes *html* mediante *BootStrap*

- ***Angular v6 Snippets***

Creador: John Papa

Detalles: Ofrece atajos a la hora de trabajar con código *TypeScript*.

- ***Path Intellisense***

Creador: Christian Kohler

Detalles: Ofrece autocompletado a la hora de incluir rutas a ficheros.

- ***Node.JS***

Se ha utilizado como entorno *JavaScript* en el servidor sobre el que se lanza la aplicación web.

- ***npm*** (**N**ode **P**ackage **M**anager)

Es un gestor de paquetes que ofrece facilidades a la hora de trabajar con *Node*, ya que gracias a él podemos tener cualquier librería disponible con una sola línea de código. Nos ayuda a administrar nuestros módulos, distribuir paquetes y agregar dependencias de forma sencilla.

- ***Git / GitHub***

En este caso, el código, tanto front como back se ha ido almacenando en un repositorio en el que se pueden ver sus versiones y cómo ha ido avanzando el desarrollo de la aplicación.

- https://github.com/Angelotas/Radiometros_Web

Si existe algún tipo de problema para visualizar el proyecto, puede ser debido a que el repositorio se encuentra privado.

- **Google Chrome**

Navegador web sobre el que se ha podido visualizar la aplicación web y sus resultados. Actúa como cliente a la hora de hacer peticiones. En concreto este navegador ofrece muy buenas herramientas tanto para diseño de la aplicación como para el debug de la misma.

Se han incluido las siguientes extensiones:

- ***JSON Viewer***

Permite formatear el código *JSON* devuelto por el servidor para su mejor entendimiento.

https://chrome.google.com/webstore/detail/json-viewer/gbmdgpbipfallnflgajpaliibnhdgobh

- ***Augury***

Ofrece herramientas que ayudan en el desarrollo, en concreto a la hora de hacer debug.

https://chrome.google.com/webstore/detail/augury/elgalmkoelokbchhkhacckoklkejnhcd

4.4. Pruebas

Se da por hecho que, para la realización de cada una de las pruebas, ya sean de caja blanco o de caja negra, se ha trabajado con una base de datos ya poblada previamente con datos reales que han sido cargados mediante el sistema de carga de este proyecto.

4.4.1. Pruebas de caja blanca

Se debe tener en cuenta que estas pruebas han sido realizadas y validadas durante el periodo de desarrollo del código de la aplicación, es decir, por cada método implementado, se han realizado sus correspondientes pruebas para verificar que funcionaba correctamente.

a) Lado cliente

En el lado cliente se han realizado pruebas sobre la aplicación web realizada en *Angular5*. Para ello, se han tenido en cuenta los diferentes módulos que la forman y los servicios que son utilizados en ellos.

1. Módulo datos

Componente `datos`

- ✓ **Método `checkDateDiferences (date1, date2)`**

  Verificar que devuelve false si la diferencia de días que hay entre la fecha correspondiente a `date1` y `date2` es menor a 5. En caso contrario devuelve true.

- ✓ **Método `checkDateOrder (date1, date2)`**

  Verifica que la fecha correspondiente a `date1` es anterior a la fecha correspondiente a date2. En ese caso devolverá `true` y en caso contrario `false`.

- ✓ **Método `checkAnteriorDate (date1)`**

  Verifica que la fecha pasada por parámetro es menor a la fecha actual. En ese caso, el método devolverá `true` y en caso contrario `false`.

- ✓ **Método `getDatos(event)`**

  Verificar que la llamada realizada al servicio `getMeasures()` se ha realizado correctamente y que el resultado obtenido es correcto y la gráfica de datos ha sido poblada con dichos datos. Por otro lado, también se ha controlado el caso de que exista un error en la llamada al servicio, dejando la gráfica de datos totalmente vacía.

- ✓ **Método `showInfo(status)`**

Comprobar que recibe correctamente el evento enviado desde el componente `form` o desde el componente `info`. Se encarga de hacer visible o no el componente `info` que muestra la información del tipo de medición que ha seleccionado el usuario. Verificar que si el evento llega desde el componente `form` se recibe `true`, haciendo la información visible y si llega desde el componente `info` se recibe `false`, ocultando la información.

- ✓ **Método `getTypeSelected(type)`**

Comprobar que siempre que el usuario cambia el tipo de medición seleccionada en el input del componente `form`, este será notificado al componente `datos` y le hace llegar un objeto con la información correspondiente del tipo.

- ✓ **Método `onChangeForm(event)`**

Comprobar que siempre que el usuario ha pulsado al botón de graficar datos del componente `form`, el componente `datos` reciba la notificación junto con un objeto que almacena los filtros seleccionados por el usuario.

Componente `form`

- ✓ **Método `onInit()`**

Verificar que la llamada a los servicios `getIds()` y `getTypes()` se gestionan correctamente. Una vez que ambos servicios han traído su información correspondiente, en los inputs del *html* relativos a los ids de radiómetros y los tipos de mediciones se completan correctamente. En caso de que fallen, los inputs permanecerán vacíos.

- ✓ **Método `sendValues()`**

Verificar que se envían correctamente los valores de los inputs seleccionados por el usuario al componente `datos`. Todo esto se realizará siempre que el usuario pulse el botón graficar datos.

- ✓ **Método `sendType()`**

Verificar que cada vez que cambia el valor del input correspondiente al tipo de medición, un objeto que contiene el nombre del tipo y la descripción será enviado al componente `datos`.

- ✓ **Método `chageShowStatus()`**

Verificar que el estado enviado al componente `datos` para la acción de mostrar el componente de información tiene valor `true`.

Componente `info`

- ✓ **Método** `closeClicked()`

  Verificar que el estado enviado al componente datos para la acción de mostrar el componente de información tiene valor false, es decir, notifica al componente padre de que tiene que ocultar el componente `info`.

2. Módulo stations

Componente `stations`

- ✓ **Método** `onInit()`

  Verificar que se gestionan las llamadas a servicios correctamente. Comprobar que cuando `getStations()` y `getIdByStation()` reciben una respuesta válida, las estaciones y su información correspondiente se mostrarán correctamente en el mapa. En caso contrario el mapa aparecerá vacío.

3. Servicios

La validación de cada uno de los métodos de los servicios que traen la información del servidor por medio de peticiones a un *API REST*, se valida en la consumición de cada uno de ellos en los módulos anteriores que hacen uso de estos.

b) Lado servidor

En el lado servidor se han realizado pruebas para comprobar si cada uno de los *endpoints* del *API REST* construido, genera el resultado que se esperan. Básicamente, por cada URL del apartado ***4.3.2.5 API REST*** se ha comprobado que el *JSON* que se ofrece como respuesta concuerda con los valores que se esperaban de la base de datos al hacer una consulta *SQL*. También se ha comprobado que cuando se introducen parámetros que generan una respuesta vacía, el servidor notifica correctamente que existe un error al no recibir información.

4.4.2. Pruebas de caja negra

En este tipo de pruebas se va a verificar el correcto funcionamiento de los requisitos que plantea el sistema en el análisis. Hay que tener en cuenta que estas pruebas van a ser complementarias a las anteriores ya que éstas se realizan a un mayor nivel de abstracción dejando de lado el comportamiento concreto de las funciones.

1. **Comprobar resultado al pulsar la opción del menú "*Gráficos de datos*"**

Los servicios que cargan de información el componente del formulario se realizan correctamente ya que los inputs "Radiometro" y "Tipo" están llenos de opciones.

Otra forma más técnica de probar si los servicios se han consumido correctamente es accediendo a las herramientas del navegador y comprobar en *Network* los servicios que se han llamado y el estado de la petición. A continuación, se observa que el estado de la respuesta es un `200 OK`.

types	200	xhr	zone.js:2969	2.8 KB	226 ms
radiometros	200	xhr	zone.js:2969	531 B	238 ms

Ilustración 77: resultado servicios types y radiometros

2. **Comprobar resultado al pulsar el botón "*Graficar datos*" incluyendo parámetros con resultados.**

Se han introducido parámetros que ya sabemos que van a devolver una respuesta.

- ✓ **Radiometro**: "`090309`"
- ✓ **Tipo**: "`SW`"
- ✓ **Fecha Inicio**: "`13/07/2017`"
- ✓ **Fecha Fin**: "`15/07/2017`"

Se pulsa el botón "*Graficar datos*" y comprobar si la gráfica se ha generado correctamente. También se puede verificar en *Network* (Debugger de *Chrome*) si el servicio `measures` se ha consumido correctamente.

measures?rad=09030...	200	xhr	zone.js:2969	175 KB	242 ms

Ilustración 78: Resultado servicio measures

También se han realizado pruebas dejando el parámetro **Fecha Fin** vacío y se ha comprobado que el resultado sigue siendo positivo.

3. Comprobar resultado al pulsar el botón "*Graficar datos*" incluyendo parámetro fecha de inicio mayor a la actual.

Se han incluido los siguientes parámetros:

- ✓ **Radiometro**: "090309"
- ✓ **Tipo**: "SW"
- × **Fecha Inicio**: "01/06/2019"
- ✓ **Fecha Fin**: ""

Primero se comprueba que la llamada al servicio no se realiza correctamente y recibe un **404 Not Found**. Por otro lado, en la interfaz, la gráfica debe estar vacía y se debe mostrar un cuadro de alerta indicando lo siguiente:

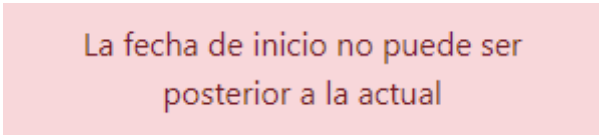


Ilustración 79: Alerta de fecha de inicio posterior a la actual

4. Comprobar resultado al pulsar el botón "*Graficar datos*" incluyendo parámetro fecha de inicio mayor al parámetro fecha de fin.

Se han incluido los siguientes parámetros:

- ✓ **Radiometro**: "090309"
- ✓ **Tipo**: "SW"
- × **Fecha Inicio**: "13/07/2017"
- × **Fecha Fin**: "10/072017"

Primero se comprueba que la llamada al servicio no se realiza correctamente y recibe un **404 Not Found**. Por otro lado, en la interfaz, la gráfica debe estar vacía y se debe mostrar un cuadro de alerta indicando lo siguiente:

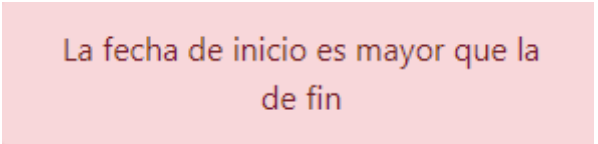


Ilustración 80: Alerta de fecha de inicio posterior a la de fin

5. **Comprobar resultado al pulsar el botón "*Graficar datos*" incluyendo parámetros fecha de inicio y fecha de fin con una diferencia entre ellos mayor a 5 días.**

Se han incluido los siguientes parámetros:

- ✓ **Radiometro**: "090309"
- ✓ **Tipo**: "SW"
- × **Fecha Inicio**: "13/07/2017"
- × **Fecha Fin**: "20/07/2017"

Primero se comprueba que la llamada al servicio no se realiza correctamente y recibe un **404 Not Found**. Por otro lado, en la interfaz, la gráfica debe estar vacía y se debe mostrar un cuadro de alerta indicando lo siguiente:

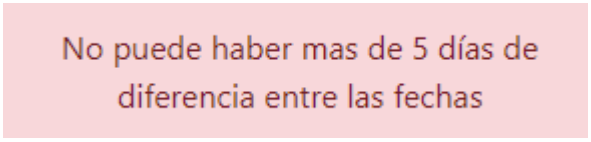


Ilustración 81: Alerta diferencia de fechas mayor a 5 días

6. **Comprobar resultado al pulsar el botón "*Graficar datos*" incluyendo parámetros que no traen respuesta del servidor.**

Se han incluido los siguientes parámetros:

- ✓ **Radiometro**: "090309"
- ✓ **Tipo**: "SW"
- × **Fecha Inicio**: "13/07/2016"
- × **Fecha Fin**: "15/07/2016"

Primero se comprueba que la llamada al servicio no se realiza correctamente y recibe un **404 Not Found**. Por otro lado, en la interfaz, la gráfica debe estar vacía y se debe mostrar un cuadro de alerta indicando lo siguiente:

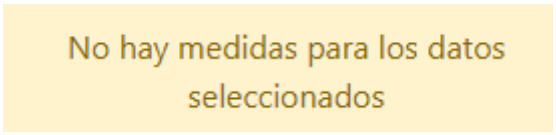


Ilustración 82: Alerta no hay datos con esos parámetros

7. **Comprobar resultado al pulsar al icono para mostrar información del tipo seleccionado.**

Pulsar el icono y verificar que se hace visible una ventana con la información acerca del tipo de medición que se encuentra marcada en el formulario.

También hay que comprobar que cada vez que se modifica el tipo de medición seleccionado, también lo hace el contenido de la ventana de información.

8. **Comprobar el resultado al pulsar sobre el icono de cerrar la ventana que muestra la información sobre el tipo de medición marcada.**

Pulsar el icono de la ventana de información y verificar que esta se oculta impidiendo ser vista por el usuario.

9. **Comprobar el resultado al pulsar sobre un punto de la gráfica**

Verificar que al pulsar sobre un punto concreto de la gráfica que se ha generado, se muestra un *tooltip* que ofrece la información en cuanto a la fecha de la medida y el valor concreto.

10. **Comprobar el resultado al pulsar sobre la opción del menú "*Mapa de estaciones*"**

Comprobar que el resultado de los servicios que traen los datos de las estaciones y sus respectivos radiómetros funcionan correctamente. En el mapa aparecen reflejados los iconos que hacen referencia a las estaciones situadas en el mapamundi.

Accediendo a la herramienta *Network* del navegador, verificar que los servicios correspondientes han obtenido una respuesta correcta `200 OK`.

stations	200	xhr	zone.js:2969	1.5 KB	92 ms
radiometros	200	xhr	zone.js:2969	531 B	23 ms
radiometros	200	xhr	zone.js:2969	531 B	26 ms
radiometros	200	xhr	zone.js:2969	531 B	25 ms

Ilustración 83: Resultado servicios stations y radiometros

11. **Comprobar el resultado al pulsar sobre una estación del mapa.**

Pulsar sobre el icono que hace referencia a una estación concreta y comprobar que se despliega un *tooltip* que muestra el nombre de la estación, sus coordenadas y los radiómetros que tiene instalados en ella.

Capítulo 5
Preparación de los sistemas

El objetivo de este capítulo es indicar los pasos necesarios a realizar para contar con dos sistemas independientes y listos para ser desplegados en producción en un servidor.

En este punto ambos sistemas han sido desarrollados teniendo un único directorio (`Caleis_Radiometros`) como resultado del primer sistema de carga y dos directorios para el sistema de visualización (`Radiometros_back` para el servidor y `RadiometrosAngular` para el front). En el caso del primer sistema, el resultado ya estaría listo para desplegarse en el servidor teniendo en cuenta el manual de instalación del siguiente capítulo para dejarlo todo configurado para el entorno en el que se quiera ejecutar. Sin embargo, para el segundo aún quedaría realizar un proceso de unificación de la parte front y back para posteriormente ser desplegado en producción.

Hasta ahora, en el desarrollo de la parte web se realizaban pruebas individualmente, es decir, por un lado, se lanzaba el *API REST* en el servidor *Node* (`localhost:3000`), y por otra la parte la aplicación web en un servidor de prueba generado por *angular-cli* (`localhost:4200`). El objetivo es unificar esas dos partes en una y poder integrar la aplicación front dentro de la aplicación *Node* del back, que será la que será desplegada como un servicio en el servidor `goacf.opt.cie.uva.es`.

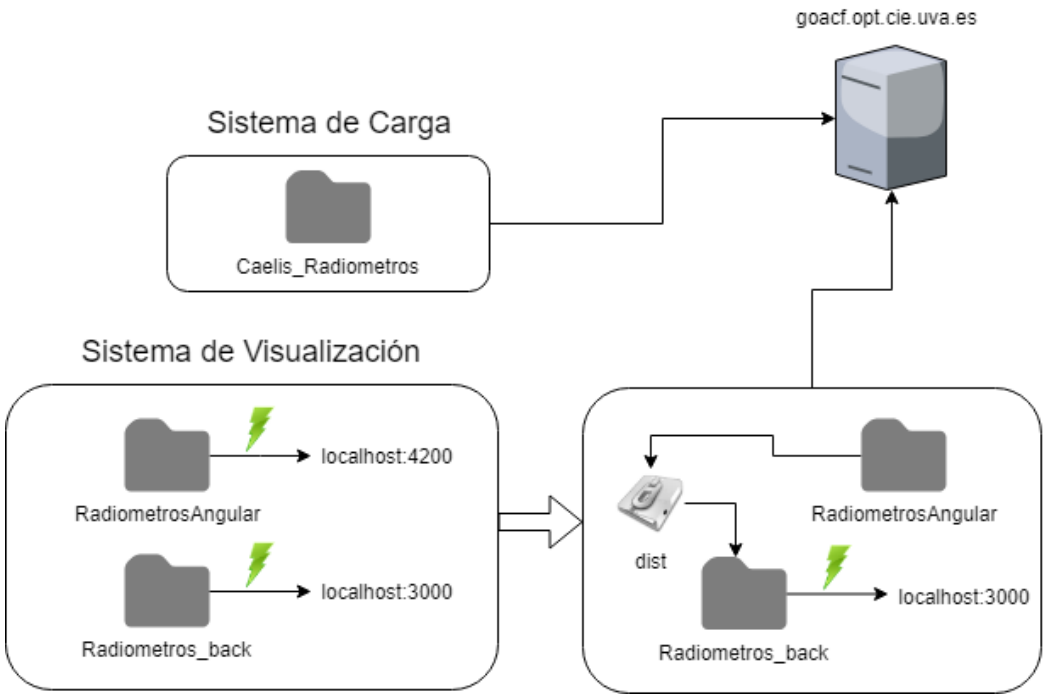


Ilustración 84: Despliegue de ambos sistemas en el servidor goa.opt.cie.uva.es

En primer lugar, lo que hay que hacer es compactar la aplicación *Angular* mediante un el comando `ng build –prod` que ofrece *angular-cli*. El resultado es un directorio `dist/` en el que queda la aplicación compilada. Este directorio está listo para ser desplegado sobre el servidor *Node*, en concreto en el directorio `/public`.

```
app.use(express.static(path.join(__dirname, 'public')));  //carga el proyecto angular
```

Ilustración 85: Asignar directorio public como recurso estático del servidor (app.js)

Con la línea de código anterior se indica que `/public` será una ruta estática que aprovechamos para desplegar nuestra aplicación *Angular*.

Ahora ya estaría nuestro sistema de visualización preparado para ser instalado, configurado y listo para ser desplegarse en el servidor `goacf.opt.cie.uva.es`.

Capítulo 6
Manuales

6.1 Manual de instalación

El manual de instalación está dirigido a la instalación de este software sobre un dispositivo con sistema operativo *GNU/Linux*. Sería conveniente que la persona encargada de esta instalación tenga conocimientos sobre los comandos básicos de este sistema operativo. Además, es conveniente que este encargado tenga permisos de super usuario ya que en algunos casos será necesario cambiar los permisos de archivos, realizar instalaciones de otros programas, tratar con variables de entorno… También será necesaria la inserción de las tablas correspondientes en la base de datos `caelis` ya existente en el servidor. En este caso, también será necesario que el usuario que va a realizar la instalación tenga los permisos necesarios sobre esta base de datos para poder modificar su estructura.

Como se puede ver, este manual de instalación no va dirigido a un usuario común final de la aplicación, sino al administrador del sistema que tiene los permisos necesarios sobre este y será el encargado de mantenerlo.

El proyecto, como está formado por dos sistemas, la instalación va a estar detallada para cada uno de ellos. Pero antes de esto, se van a indicar cuáles son los requisitos previos con los que tiene que contar el sistema:

- ***MySQL***: Es necesario tener instalado tanto el cliente como el servidor, así como las librerías de desarrollo. El nombre de los paquetes serán `mysql-client`, `mysql-server` y `libmysqlclient-dev`.
- ***Build-essential***: Son un conjunto de aplicaciones necesarias para la compilación. Si no se desea instalar el meta-paquete, es posible instalar el paquete `gcc`.
- ***Angular-cli***: Mediante `ng build` podemos construir el proyecto para dejarlo listo para su producción. En este caso la versión sobre la que se trabaja es la `1.7.4`.
- ***Node.js***: Es el ambiente de ejecución *JavaScript* sobre el que se va a lanzar la aplicación web junto al *API REST*. En este caso la versión sobre la que se trabaja es la `v8.9.4`.
- ***npm***: es el administrador de paquetes de *Node*. Con él se pueden instalar cualquier paquete *Node.js* que se necesiten. En este caso la versión instalada es la `5.6.0`.
- ***PM2***: Gestor de proyecto es producción. Se utiliza para lanzar la aplicación en el servidor como un servicio y poder gestionarlo.

6.2.1 Sistema de carga

En primer lugar, será necesario tener preparada la base de datos sobre la que los programas van a trabajar. Para ello, bastará con que el administrador ejecute el script `DDL_radiometros.sql` sobre la base de datos. La forma más sencilla es desde el terminal de la siguiente manera:

```
$ mysql -u "nombre usuario" -p caelis < DDL_radiometros.sql
```

Ilustración 86: Lanzar script DDL_radiometros.sql

Una vez que ya están las tablas correspondientes creadas, se pasa a la configuración de los programas y librerías:

1) Crear el fichero de configuración de *MySQL* `rad_config.cnf` que almacene el nombre de usuario, la contraseña y el host que se utilizarán para establecer conexión con la base de datos. Lo recomendable es que se almacene en un directorio seguro con permisos limitados al resto de usuarios del sistema. En este caso la ruta ha sido `/etc/mysql/rad_config.cnf`.

```
[client]
user =
password =
host = localhost
```

Ilustración 87: archivo rad_config.cnf

2) Desde el script `conection_db.sh` situado dentro del directorio `ProgramsServer/sources_rad_manage`, se le asignará a la variable `conf_file_path` la ruta donde se encuentre situado el fichero de configuración *mysql* del punto anterior (en este caso `/etc/mysql/rad_config.cnf`).

3) Acceder a la librería `conection.h` situada en `ProgramsServer/libs` y asignar valores a las constantes que almacenan las credenciales para conectarse a la base de datos desde lo programas `C` (`SERVER – DATABASE – USER – PASSWORD`)

4) Acceder a la librería `read_parameters.h` situada en `ProgramsServer/libs` y dar valor a la constante `PATH` con la ruta absoluta del directorio del servidor desde el que se leen ficheros que, en este caso, es `/rad_upload` (`/home/angel/Caelis_radiometros/Files/rad_upload/`).

5) Acceder al script `rad_manage.sh`, `rad_couslt_status.sh`, `rad_activate.sh` y `rad_desactivate.sh` y establecer a la constante `RUTA_RECURSOS` el directorio raíz del proyecto (`/home/angel/Caelis_radiometros`).

Cuando ya están configurados los programas del sistema de carga, es necesario compilar los programas en *C*. Para ello, ejecutar los siguientes comandos desde el directorio raíz del proyecto:

- `gcc -o rad_filter ./ProgramsServer/filter/rad_filter.c -lmysqlclient`
- `gcc -o rad_measures ./ProgramsServer/insert/rad_measures.c -lmysqlclient`
- `gcc -o rad_getId ./ProgramsServer/sources_rad_manage/rad_getId.c`

Una vez compilados los programas, ya no sería necesario seguir almacenando los ficheros *.c* en esas rutas ya que queremos prevenir que algún usuario pueda acceder al código fuente de los programas. Bastaría con almacenar los ejecutables generados en sus correspondientes rutas.

Por otro lado, los scripts *.sh* se almacenan en texto plano en el servidor. Por eso, es necesario almacenar parámetros como contraseñas o información de conexiones en ficheros de configuración del servidor (`/etc/mysql/rad_config.cnf`). De esta manera verificamos que únicamente podrá ver la información sensible aquel usuario con permisos.

El resultado final de directorios en el servidor sería el siguiente:

```
.:
Files  ProgramsServer

./Files:
rad_achieve  rad_upload

./Files/rad_achieve:
correct  quarentine

./ProgramsServer:
filter  insert  rad_activation_control  rad_manage.sh  README.txt  sources_rad_manage

./ProgramsServer/filter:
rad_filter

./ProgramsServer/insert:
rad_measures

./ProgramsServer/rad_activation_control:
rad_activate.sh  rad_consult_status.sh  rad_desactivate.sh

./ProgramsServer/sources_rad_manage:
conection_db.sh  rad_getId
```

Ilustración 88: Despliegue de archivos en el servidor para el sistema de carga

Finalmente, solo quedaría crear el servicio que hace que el script `rad_manage.sh` se ejecute de forma periódica, en este caso cada 5 minutos. Para ello se debe ejecutar el `crontab -e`. Este se encarga de abrir en modo edición el fichero cron que posee las tareas que se ejecutarán automáticamente. Ahora solo queda añadir la siguiente línea al final del fichero:

```
#Run rad_manage.sh every 5 minutes.
*/5 * * * * /home/angel/Caelis_radiometros/Programs/rad_manage.sh
```

Ilustración 89: Línea que activa la ejecución automática de rad_manage.sh cada 5 minutos

6.2.2 Sistema de visualización

Partiendo del contenido que se encuentra en el *Capítulo 5: Preparación de los sistemas*, el de visualización ya se encontraría unificado y, por lo tanto, solo quedaría lanzar como un servicio el `app.js` de la aplicación *Node*. Para ello se utiliza el gestor de proyectos *PM2* y solo quedaría ejecutar el siguiente comando que inicia la aplicación:

```
$ pm2 start "ruta/a/app.js"
```

Ilustración 90: Iniciar app.js de Node como servicio

6.2 Manual de administrador

En este caso, el administrador va a tener un rol relevante sobre todo en el sistema de carga ya que una vez que ha instalado el sistema, es el encargado de interactuar con él. También tendrá funciones en el sistema de visualización.

6.2.1 Sistema de carga

- **Gestionar el estado del procesamiento de carga**

El administrador es el encargado de lanzar o parar el procesamiento de ficheros para su carga en la base de datos. En este caso, también tiene la posibilidad de comprobar si este estado se encuentra activado o en su defecto desactivado. Para ello, nos vamos a conectar al servidor mediante *SSH* (se ha utilizado el cliente *SSH Putty*) y mediante comandos accedemos al directorio del proyecto `/Caelis_radiometros` y, dentro de él, vamos al que contiene los programas, `/ProgramsServer`. A continuación, entramos en el directorio `/rad_activation_control` y donde se encuentran los siguientes scripts:

```
angel@goacf:~/Caelis_radiometros/ProgramsServer/rad_activation_control$ ls -l
total 12
-rwx------ 1 angel users  870 abr 16 20:03 rad_activate.sh
-rwx------ 1 angel users 1288 abr 16 20:03 rad_consult_status.sh
-rwx------ 1 angel users  871 abr 16 20:03 rad_desactivate.sh
```

Ilustración 91: Recursos del directorio rad_activation_control

- **`rad_activate.sh`:** pone a activo el estado del procesamiento de ficheros.
- **`rad_desactivate.sh`:** desactiva el procesamiento de ficheros.
- **`rad_consult_status.sh`:** muestra el estado del procesamiento de fichero, pudiendo estar activado o desactivado.

Para ejecutar cualquiera de los siguientes scripts usar:

```
$ ./"ruta hasta el script"/"script".sh
```

Ilustración 92: Ejecutar scripts del directorio rad_activation_control

A continuación, se muestra la salida que ofrece la consola al ejecutar cualquiera de los scripts.

```
|-------------------------------------------------------------|
|                    rad_consult_status.sh                    |
|-------------------------------------------------------------|
|                                                             |
|  El estado del sistema de carga está ACTIVADO               |
| *Si deseas desactivarlo ejecuta el script ./rad_desactivate.sh |
|                                                             |
|-------------------------------------------------------------|
```

Ilustración 93: Resultado de la ejecución de rad_consult.status.sh

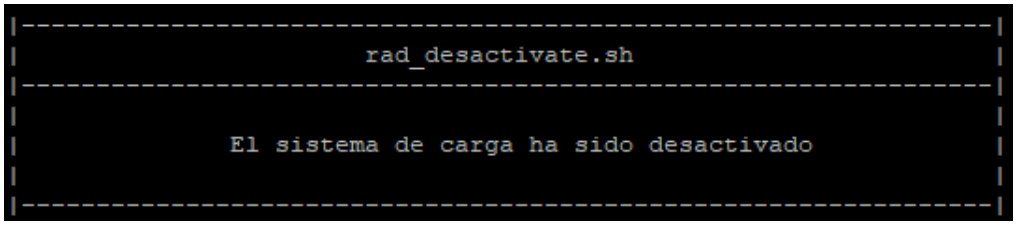


Ilustración 94: Resultado de la ejecución de rad_desactivate.sh

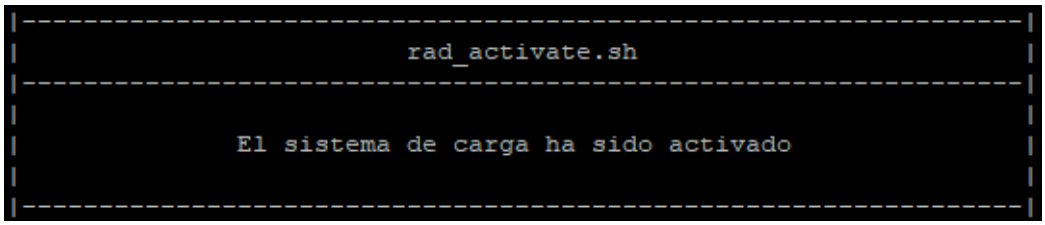


Ilustración 95: Resultado de la ejecución de rad_activate.sh

- **Cambiar periodo de ejecución del procesamiento de carga**

Desde un inicio, el servicio que ejecuta el script automáticamente, lo hace con una frecuencia de 5 minutos, pero si el administrador lo desea, esta frecuencia puede ser variar. Para ello ejecutar el comando `crontab -e` y modificar la línea del fichero en la que se activa el servicio que ejecuta el script correspondiente.

```
#Run rad_manage.sh every 5 minutes.
*/5 * * * * /home/angel/Caelis_radiometros/ProgramsServer/rad_manage.sh
```

Ilustración 96: Línea que activa la ejecución automática de rad_manage.sh cada 5 minutos

La estructura del comando consta de 5 asteriscos y el comando que se quiere ejecutar:

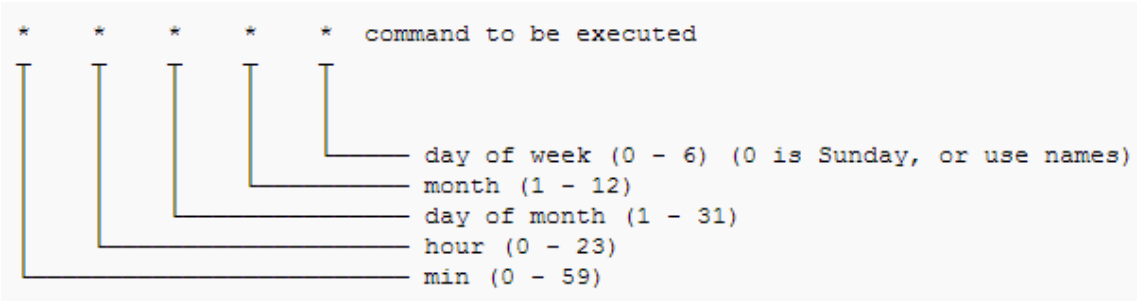


Ilustración 97: Estructura del comando crontab

En este caso en los minutos se ha introducido un `*/5`. Con esto se indica que en vez de que se ejecute cada minuto 5 de cada hora, de cada día… se ejecuta cada 5 minutos.

- **Añadir ficheros manualmente al directorio del que se procesan los mismos.**

Inicialmente el directorio del que se van a recoger los ficheros *.rad* lo irá poblando un script externo a este sistema, pero existe la posibilidad de que el administrador quiera incluir algún fichero *.rad* de forma manual. El origen de este, puede ser un dispositivo cliente, por ejemplo, un ordenador del administrador, o desde un origen interno del servidor. En el primer caso se puede recurrir a una conexión *FTP* con el servidor (*Filezilla*) y cargar los ficheros en el directorio correspondiente. En el segundo caso simplemente usar el comando:

```
$ mv "origen_fichero".rad  "ruta_a_rad_upload"
```

Ilustración 98: Comando para mover ficheros manualmente

6.2.1 Sistema de visualización

En cuanto al sistema de visualización, el administrados será capaz de parar o reiniciar el servicio que lanza la aplicación web. También podrá monitorizar en todo momento el estado de este. Todo esto gracias al gestor de procesos de producción ***PM2*** para aplicaciones *Node*.

- `$ pm2 list`

Lista todos los servicios *pm2*, cada uno con su respectivo identificador.

- `$ pm2 stop "id_aplicación_node"`

Para el servicio correspondiente a la aplicación *Node* con el id pasado por parámetro.

- `$ pm2 restart "id_aplicación_node"`

Reinicia el servicio correspondiente a la aplicación *Node* con el id pasado por parámetro.

6.3 Manual de usuario

En este apartado se hace una breve demostración en la que se indican todas las funcionalidades a las que el usuario tiene acceso respecto a la aplicación web creada. El usuario podrá acceder a la misma desde cualquier navegador que tenga soporte para ***ES6*** y ***TypeScript*** (*Chrome*, *Firefox*, *Safari*…).

La estructura de la aplicación se basa en una parte estática formada por la cabecera, con icono y nombre de la aplicación *Ilustración 99*, un menú general a través del cual se puede acceder a las distintas funcionalidades *Ilustración 100* y finalmente un pie de página *Ilustración 101*.

Ilustración 99: Cabecera de la aplicación

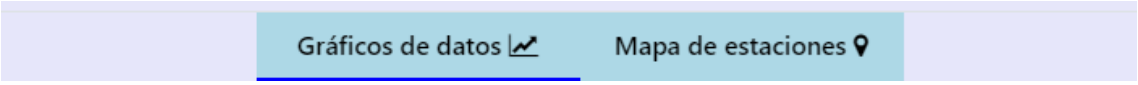


Ilustración 100: Menú general de la aplicación

Ilustración 101: Pie de página de la aplicación

Respecto a la parte dinámica de la aplicación, existen dos vistas diferentes que se cargan en función de la opción del menú que el usuario selecciona. Como se puede ver en la *Ilustración 100,* en el momento de la captura de la imagen, la opción seleccionada es la de "*Gráfico de datos*" pudiendo ver como su borde inferior tiene un borde azul oscuro.

- **Opción "*Gráfico de datos*" seleccionada**

La vista ofrece una interfaz formada por un componente (1) que permite hacer un filtrado de datos teniendo en cuenta los ítems que lo forman. Se puede elegir el identificador del radiómetro del que traer lo datos, también el tipo de medición y la fecha concreta o rango de tiempo entre dos fechas. También se ofrece la posibilidad de ver la información acerca del tipo de medición que queda marcada. Para ello se pulsa sobre el icono  situado a la derecha del ítem "*Tipo*" y se despliega un nuevo componente (2) con esta información. Se podrá cerrar haciendo clic en el icono  .

En la parte inferior se encuentra la gráfica (3) que representa los valores de las medidas a lo largo del tiempo. En el caso de la *Ilustración 102* cuando se ha pulsado al botón "*Graficar Datos*", sí que existen medidas para los valores introducidos en el filtro. En caso contrario la gráfica quedaría vacía.

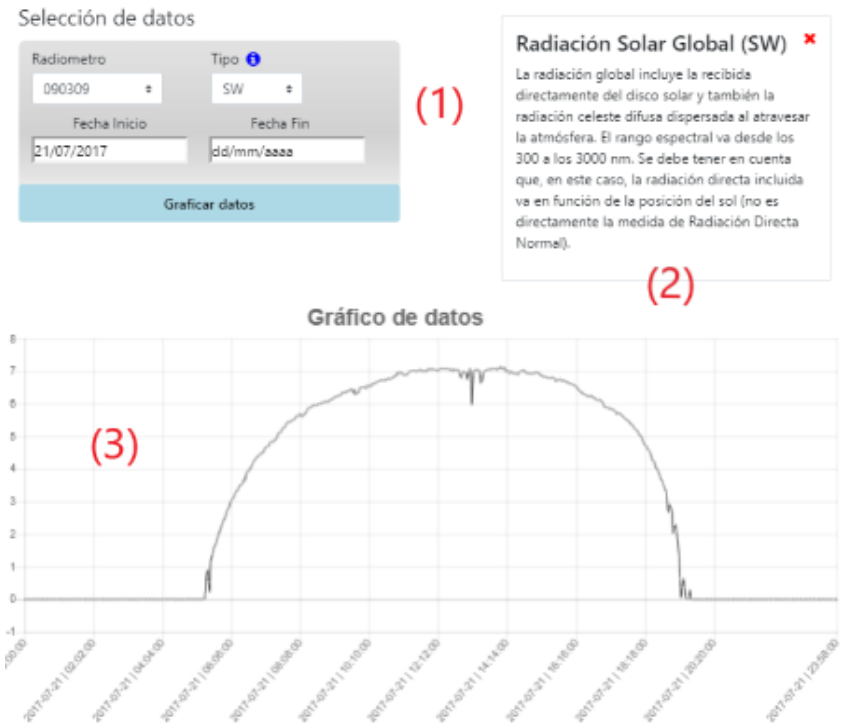


Ilustración 102: Vista de Gráfico de datos

Sobre los datos de la gráfica se pueden consultar los valores exactos de un determinado punto en concreto situando el cursor sobre el mismo (*Ilustración 103*).

Ilustración 103: Detalle de una medida

- **Opción "Mapa de estaciones" seleccionada**

Cuando el usuario hace clic sobre la opción del menú "*Mapa de estaciones*", se carga una vista que muestra un mapa en el que quedan marcadas las localizaciones exactas de las distintas estaciones radiométricas que hay registradas en la base de datos.

Ilustración 104: Vista de mapa de estaciones

Además de esto, el usuario puede hacer clic sobre cada una de las marcas que corresponden a una determinada estación. En ese momento se despliega un *tooltip* que muestra la información concreta de la misma: Nombre de la estación, coordenadas y radiómetros que tiene instalados en ella.

Ilustración 105: Información de una estación concreta

Finalmente, en esta vista también se permite hacer zoom (4) para acercar o alejar los detalles del mapa:

Ilustración 106: Mapa ampliado

Capítulo 7
Conclusiones

Una vez que el proyecto está acabado, se genera una conclusión final en la que se hace referencia a si los objetivos establecidos en el análisis inicial se han cumplido satisfactoriamente o no, a los nuevos conocimientos adquiridos a partir de la realización del proyecto y, por último, al trabajo futuro que se pretende llevar a cabo.

7.1 Objetivos logrados

El **sistema de carga** va a ayudar de forma activa al crecimiento de la red de dispositivos radiométricos gestionada por el GOA puesto que alivia en gran medida la carga de trabajo sobre ella, tanto en los trabajos de carga de datos al sistema, como en su mantenimiento y control.

Otro de los objetivos era tener toda la información centralizada en una base de datos con el fin de poder trabajar sobre ellos y poder elaborar nuevas aplicaciones a partir de estos. En este aspecto, considero que el objetivo se ha cumplido, la información se encuentra en la base de datos *CAELIS* perteneciente al GOA y, además para argumentarlo, se ha construido una aplicación web que utiliza estos datos para ser representados gráficamente.

Respecto al **sistema de visualización**, como acabo de comentar, se ha desarrollado una aplicación que representa los valores de las mediciones. Estos datos son graficados, lo que hace que sea más visual y entendible toda esta información con el fin de que la comunidad científica pueda realizar estudios o sacar conclusiones.

7.2 Aprendizaje

La realización de este proyecto ha sido realmente enriquecedora debido a las numerosas tecnologías que se han utilizado. Por un lado, ha servido para poner en práctica y profundizar más en tecnologías vistas a lo largo de la carrera, así como el lenguaje *C* para manipular la información de los ficheros de datos y poder cargarla en la base de datos. El uso de *Bash Script* para gestionar y automatizar la ejecución de los programas. Diseñar la estructura de tablas correspondientes para incorporar a la base de datos *CAELIS* utilizando *MySQL*.

Por otro lado, la novedad ha sido trabajar con tecnologías tan punteras en el desarrollo web como son *Angular5* para el desarrollo front y *NodeJS* y *ExpressJS* para el desarrollo back. Todo esto ha supuesto una larga curva de aprendizaje.

7.3 Mejoras futuras

Las aplicaciones resultantes de este proyecto son completamente funcionales y cumplen con los requisitos establecidos inicialmente, sin embargo, mi intención es seguir colaborando con el GOA en tareas de mantenimiento o en posibles nuevas propuestas que impliquen un nuevo alcance en este proyecto relacionado con dispositivos radiométricos.

Capítulo 8
Bibliografía

- Cerezo, Alejandro. “Angular Camp”. *Módulo WebApps con Angular5*. KeepCoding, 30 de enero de 2018.
- Grande, Pilar. “Ficheros”. *Fundamentos de la Programación*. Universidad de Valladolid, 2014

- «¿Qué factores determinan el clima de la Tierra?». https://www.ipcc.ch/publications_and_data/ar4/wg1/es/faq-1-1.html. [Último acceso el 25/06/2018].

- «La radiación solar» http://www.aemet.es/documentos/es/eltiempo/observacion/radiacion/Radiacion_Solar.pdf . [Último acceso el 25/06/2018].

- «Radiación Infrarroja» http://legacy.spitzer.caltech.edu/espanol/edu/ask/radiation.html . [Último acceso el 25/06/2018].

- «Instalar y configurar Ubuntu Server y Plesk con VirtualBox». https://www.adictosaltrabajo.com/tutoriales/instalar-y-configurar-ubuntu-server-y-plesk-con-virtualbox/. [Último acceso el 30/01/2018].

- «¿Cuándo usar Bash y Cuándo usar Perl/Python/Ruby?». https://www.enmimaquinafunciona.com/pregunta/27215/cuando-usar-bash-y-cuando-usar-perlpythonruby. [Último acceso el 25/06/2018].

- «Estructuras de Datos Complejas» http://www.fisica.ru/dfmg/teacher/archivos/9._VECTORES.pdf [Último acceso el 25/06/2018].

- «Conectar C con MySQL». http://www.aprendeaprogramar.com/mod/forum/discuss.php?d=626. [Último acceso el 25/06/2018].

- «MYSQL C connector with MinGW on Code::Blocks». https://cppdevblog.wordpress.com/2015/10/11/mysql-c-connector-with-mingw-on-codeblocks/. [Último acceso el 25/06/2018].

- «Programación Shell-Script».
  http://persoal.citius.usc.es/tf.pena/ASR/Tema_2html/node20.html.
  [Último acceso el 25/06/2018].

- «Cómo utilizar Cron y Crontab en Linux para programar tareas».
  https://www.redeszone.net/2017/01/09/utilizar-cron-crontab-linux-programar-tareas/.
  [Último acceso el 25/06/2018].

- «Express. Database integration».
  https://expressjs.com/en/guide/database-integration.html.
  [Último acceso el 25/06/2018].

- «Node.js, MySQL and promises».
  https://codeburst.io/node-js-mysql-and-promises-4c3be599909b.
  [Último acceso el 25/06/2018].

- «Programación Asíncrona en Node JS».
  https://es.slideshare.net/jvelez77/presentacion-35264918.
  [Último acceso el 25/06/2018].

- «Using Bootstrap with Angular».
  https://medium.com/codingthesmartway-com-blog/using-bootstrap-with-angular-c83c3cee3f4a.
  [Último acceso el 25/06/2018].

- «Chart.js>>.
  http://www.chartjs.org/docs/latest/.
  [Último acceso el 25/06/2018].

- «Advanced, production process manager for Node.js».
  http://pm2.keymetrics.io/.
  [Último acceso el 25/06/2018].