TRABAJO DE FIN DE MÁSTER

Titulación: Máster Universitario en Ingeniería Informática
Especialidad: Tecnologías Web y Negocios Digitales

Aplicación web de procesado, almacenamiento y visualización de datos oceanográficos tipo perfiles verticales

José Orlando Díaz Cueva

Tutor académico: José Fortes Gálvez
Tutora de empresa: Tania Morales Morales
Fecha de presentación: Julio de 2017

Contenido

Índice de Figuras

Índice de Tablas

Glosario: Listado de Acrónimos

Acrónimo	Significado
AOT	Ahead-of-Time Compilation
API	Application Programming Interface
BODC	British Oceanographic Data Center
CF	Climate and Forecast Metadata Convention
CORS	Cross-origin resource sharing
CSS	Cascading Stylesheets
CSV	Comma separated values text format
DSG	Discrete Sampling Geometries
EMODNET	The European Marine Observation and Data Network
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
ICTS	Infraestructura Científico-Técnica de Carácter Singular
IDE	Integrated development environment
IMOS	Integrated Marine Observing System
IOOS	Integrated Ocean Observing System
JIT	Just-in-Time Compilation
JSON	JavaScript Object Notation
MVT	Model View Template
netCDF	Network Common Data Form
NOAA	National Oceanic and Atmospheric Administration
NODB	National Oceanographic *Database*
OGC	Open Geospatial Consortium
OpeNDap	Open-source Project for a Network Data Access Protocol
ORDBMS	Object-relational database management system
ORM	Object Relational Mapping
PLOCAN	Plataforma Oceánica de Canarias
RDBMS	Relational database management system
REST API	Representational State Transfer

RUP	Rational Unified Process
SSL	Secure Socket Layer
THREDDS	Thematic Real-time Environmental Distributed Data Services
WCS	Web Coverage Services
WMS	Web Map Services
XML	Extensible Markup Language
XSS	Cross-site scripting

1. Introducción

El océano es una de las regiones menos estudiadas y observadas del planeta. Actualmente prestigiosas organizaciones a nivel mundial concentran sus esfuerzos para mejorar nuestra comprensión del océano. Un ejemplo lo constituye la Red Europea de Observación e Información del Mar (EMODNET)[1]. Esta organización se dedica a reunir información marítima procedente de distintas fuentes, pertenecientes a sus socios dentro de la Unión Europea, con el objetivo de:

- Ayudar al sector, las autoridades públicas y los investigadores a encontrar información y utilizarla de forma más eficaz para desarrollar nuevos productos y servicios.
- Mejorar nuestro conocimiento del comportamiento del mar.

En las páginas de EMODNET, los ingenieros y científicos pueden ver qué datos existen sobre una cuenca marítima determinada y descargar observaciones originales y datos elaborados, como por ejemplo modelos digitales del terreno, distribuciones de sedimentos y hábitats marinos. El desarrollo de sistemas regionales y subregionales de observación de las zonas costeras proporciona mejores oportunidades para un ***mayor acceso a los datos meteorológicos y oceanográficos***. Las ventajas del acceso a este tipo de datos contribuyen a la conservación de los ecosistemas marinos y al aprovechamiento correcto de los recursos para la implantación de una economía azul.

Canarias no escapa a esta tendencia y en fecha tan reciente como Julio del 2016 el presidente del Cabildo de Gran Canaria, Antonio Morales, y el director del consorcio Plataforma Oceánica de Canarias (PLOCAN)[1], Octavio Llinás, firmaron un convenio para aunar esfuerzos a través de la Sociedad de Promoción Económica de Gran Canaria y potenciar la capacidad de crecimiento de la economía azul y de la biotecnología marina [2]. Recientemente a principios del 2017 el consorcio público PLOCAN, dentro de la cual se enmarca este proyecto, se convirtió en socio de EMODNET [3].

En este marco se propone el desarrollo del presente sistema, con el objetivo de cubrir la necesidad en PLOCAN de automatizar la gestión de datos de los perfiles verticales tomados en distintas campañas oceanográficas realizadas en el área reservada conocida como Banco de Ensayo. El sistema contará con una interfaz de administración que podrá ser usada por oceanógrafos y científicos en general para subir los datos de los perfiles verticales recolectados. Dichos datos serán puestos a disposición de la comunidad científica mediante el acceso a una web que permite filtrarlos y graficarlos.

Se trata de implantar una solución web Open Source que permita procesar, almacenar y visualizar los perfiles verticales procedentes de las estaciones de medida de la citada área. Se utilizan tecnologías y estándares compatibles con la organización Open Geospatial Consortium (OGC)[4] y la convención Climate and Forecast Metadata Conventions (CF)[5]. Entre los componentes más relevantes de la arquitectura a diseñar se encuentra el servidor de datos científicos ERDDAP, responsable de indexar y distribuir los datos tipo perfiles verticales a través de ficheros binarios con formato Network Common Data Form (netCDF) [6].

[1] http://plocan.eu/index.php/es/

El sistema cuenta con una interfaz de administración desarrollada en Django para la gestión de los perfiles y sus metadatos asociados como estaciones de observación, campañas oceanográficas, sensores y variables medidas. La parte pública de la aplicación permite realizar consultas sobre los datos de los perfiles verticales, visualizarlos y compararlos mediante gráficas. Permite obtener todos los perfiles verticales de una variable específica para una estación en particular y los resultados se organizan por campañas. Igualmente permite obtener los perfiles verticales de una variable específica para una determinada campaña y los resultados se muestran organizados por estaciones.

2. Contexto y Motivación del Proyecto

Los recientes avances en el diseño de los sensores y en las técnicas de análisis de datos han contribuido a disminuir los costes de adquirir sistemas remotos de monitoreo para el estudio de los cambios producidos por el hombre en los ecosistemas costeros [7]. Por otro lado, los datos oceanográficos que se generan son muy diversos, abarcando una amplia gama de escalas espaciales tales como trayectorias, series temporales, perfiles verticales y abarcando mediciones de satélite, mediciones in situ y simulaciones numéricas.

El volumen y la complejidad de estos datos también están aumentando exponencialmente con el despliegue de nuevos sistemas de observación del océano, el aumento en la resolución de modelos y el advenimiento de nuevas técnicas de modelado [8]. Sin embargo, este auge necesita de la correspondiente **mejora en el acceso y disponibilidad de los datos** a la comunidad científica.

En este contexto se inserta PLOCAN, que constituye una Infraestructura Científico-Técnica de Carácter Singular (ICTS) del Estado español, dedicada a la innovación tecnológica marina así como a la movilización económica de su entorno [9]. Como parte de su actividad despliega una gran cantidad de estaciones de observación marinas que recopilan diferentes tipos de datos científicos como pueden ser series temporales o perfiles verticales.

Este proyecto se integrará a un proyecto interno más amplio de carácter científico-técnico. ***El foco del Trabajo de Fin de Máster se centrará específicamente en el desarrollo de una aplicación web que permita procesar, almacenar y visualizar datos oceanográficos tipo perfiles verticales procedentes de las estaciones de medida pertenecientes al Banco de Ensayos de PLOCAN***. En lo referente a la visualización de los datos se trabajará con librerías JavaScript para la creación de mapas interactivos así como el análisis a través de gráficas interactivas 2D. Los datos oceanográficos provienen de estaciones en el mar equipadas con sensores que captan datos dependientes de la profundidad como la temperatura, salinidad, conductividad, pH, etc. Ver Figura 1.

En concreto, en PLOCAN se pretende cubrir la necesidad de automatizar la gestión de datos de los perfiles verticales tomados en distintas campañas. A nivel tanto canario como mundial, esta herramienta podrá ser utilizada por cualquier tipo de institución que lleve a cabo campañas oceanográficas ya que las necesidades y tipos de soluciones requeridas son similares. Los oceanógrafos y científicos podrán utilizar el sistema para subir los datos recolectados durante las campañas y estos podrán ser posteriormente filtrados y visualizados por la comunidad científica en general.

En la CF se define un perfil como un conjunto ordenado de puntos de datos a lo largo de una línea vertical en una posición horizontal fija y un tiempo fijo. Ver las geometrías de muestreo discreto (DSG) definidas en la convención CF [10].

```
Pres Temp Cond Sal O2Sat% O2ppm pH Chl(a) Turb. Time Date
0.37 24.972 0.000 0.012 109.8 9.05 3.30 0.03 -0.04 16:40:30.72 30/07/2012
1.08 22.845 53.150 36.805 101.3 7.03 8.07 -0.01 1.44 16:42:48.33 30/07/2012
2.06 22.847 53.130 36.788 102.8 7.13 8.05 0.04 1.49 16:42:53.05 30/07/2012
3.04 22.846 53.125 36.783 103.0 7.14 8.04 0.04 1.46 16:42:54.11 30/07/2012
4.07 22.849 53.134 36.788 103.7 7.19 8.04 0.06 1.40 16:42:55.84 30/07/2012
5.06 22.848 53.130 36.785 104.3 7.23 8.04 0.05 1.46 16:42:57.33 30/07/2012
6.16 22.843 53.131 36.790 104.7 7.26 8.04 0.06 1.40 16:42:58.57 30/07/2012
7.17 22.810 53.094 36.788 105.0 7.29 8.04 0.06 1.41 16:42:59.62 30/07/2012
8.17 22.764 53.098 36.830 105.4 7.32 8.04 0.06 1.41 16:43:00.86 30/07/2012
9.14 22.740 53.043 36.807 105.6 7.34 8.04 0.06 1.42 16:43:02.10 30/07/2012
10.16 22.731 53.021 36.797 105.8 7.35 8.05 0.07 1.41 16:43:03.59 30/07/2012
11.10 22.722 53.009 36.794 106.0 7.36 8.04 0.07 1.41 16:43:04.60 30/07/2012
```

Figura 1: Ejemplo de fichero de mediciones de perfiles verticales de una estación

En la Figura 1 se muestra un pequeño ejemplo de fichero de texto que contiene las mediciones tomadas en una estación de observación fija en el mar, durante una campaña. Se puede apreciar que la primera fila define los nombres de las variables medidas en dicha estación. Cada columna constituye entonces un perfil vertical perteneciente a su respectiva variable. La primera columna contiene los datos de la presión, que representan las distintas profundidades a las que se miden las observaciones de los perfiles. La columna tiempo contiene pequeñas variaciones del tiempo en que se realizaron las mediciones, no obstante según el concepto de perfil vertical se define que el tiempo de todas las observaciones de un perfil es el mismo: el primer valor de la columna tiempo.

3. Objetivos

A continuación se definen los objetivos del proyecto:

- Aplicar la metodología de ingeniería de software para documentar el proceso de desarrollo del proyecto, especificando el análisis, diseño, implementación, pruebas y validación del sistema.
- Identificar las mejores prácticas, tecnologías y herramientas que utilizan centros de datos marinos de referencia mundial para el tratamiento de datos oceanográficos.
- Llevar a cabo el desarrollo con herramientas de software libre o de código abierto.
- En el sistema a desarrollar, integrar un servidor de datos científicos que sirva de interfaz de acceso y distribución a los datos oceanográficos.
- Crear una aplicación web que sirva de soporte a los administradores del sistema, permitiendo captar los datos oceanográficos, gestionar sus metadatos y generar ficheros netCDF tipo perfiles verticales utilizando una librería propia creada en PLOCAN.
- Crear una aplicación web específica para la:
    - Visualización de las estaciones de observación en un mapa
    - Visualización en gráficas 2D de los datos tipo perfil vertical
    - Realizar consultas por estaciones y por campañas oceanográficas.

4. Estado del Arte

En este apartado se mostrará de manera general qué se está haciendo actualmente en relación a la gestión de los datos oceanográficos, en tres centros de datos marinos de referencia mundial:

- Sistema de Observación Marina Integrada de Australia (IMOS)[2].
- Centro de Datos Oceanográfico Británico (BODC)[3].
- Administración Nacional Oceánica y Atmosférica de los Estados Unidos (NOAA)[4].

Además se verá durante el transcurso de ese análisis la principal solución técnica que existe para almacenar los datos oceanográficos:

- Ficheros en formato netCDF: Es un formato binario de datos auto descriptivo, independiente del tipo de sistema operativo y soporta la creación, el acceso y el intercambio de datos científicos multidimensionales. Cuando se menciona los términos "datos multidimensionales" se refiere a datos que dependen de varias dimensiones como por ejemplo datos geolocalizados que dependen de la latitud y la longitud o datos temporales que dependen del tiempo.

Finalmente y siguiendo la línea de los ficheros netCDF, se muestran las características de dos populares servidores de datos científicos:

- THREDDS, por sus siglas en inglés de Thematic Real-time Environmental Distributed Data Services[5].
- ERDDAP[6].

Estos servidores trabajan sobre ficheros netCDF y permiten indexarlos, compartirlos y facilitar la posterior consulta y análisis de la información contenida en ellos.

4.1. Centros de datos marinos de referencia mundial

4.1.1. Sistema de Observación Marina Integrada de Australia

En [11] se describe la infraestructura y los flujos de trabajo desarrollados para administrar y distribuir los datos al público de IMOS. Se destacan las normas existentes y el software de código abierto que han adoptado, y las contribuciones hechas.

[2] http://imos.org.au/home/
[3] https://www.bodc.ac.uk/
[4] http://www.noaa.gov/oceans-coasts
[5] http://www.unidata.ucar.edu/software/thredds/current/tds/TDS.html
[6] https://coastwatch.pfeg.noaa.gov/erddap/index.html

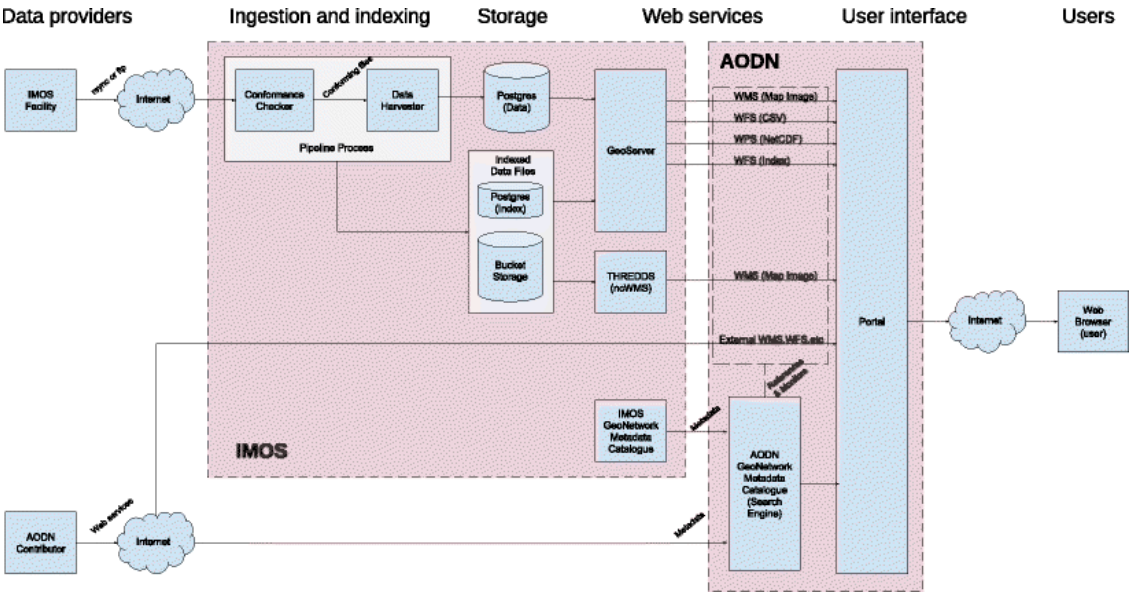

Figura 2: Componentes de la infraestructura de información de IMOS [11]

En la Figura 2 se muestran los principales componentes de la infraestructura de información de IMOS. Ilustra cómo los datos de fuentes externas a IMOS transitan a la Red de Datos del Océano Australiano (AODN) [11]. Se puede apreciar que partiendo de los datos científicos aportados por proveedores se procede a continuación a la fase de captación e indexado, a su posterior almacenamiento y finalmente son distribuidos gracias a una serie de servicios web implementados por servidores de datos como THREDDS y GeoServer[7].

En cuanto a los formatos y convenciones, según se describe en [11], IMOS ha adoptado el formato netCDF como el formato primario para la transferencia y almacenamiento de datos y su entrega a los usuarios. Estos autores destacan que este formato tiene muchas ventajas, es auto descriptivo (es decir, almacena datos y metadatos juntos), flexible, capaz de almacenar grandes volúmenes de datos multidimensionales y puede ser leído por herramientas comunes de análisis de datos.

Para la interoperabilidad con las herramientas de análisis disponibles en el mercado, los archivos netCDF generados por IMOS son estructurados y documentados de acuerdo con la convención CF. Esta convención es comúnmente utilizada en oceanografía y está diseñada para promover el procesamiento y el intercambio de archivos netCDF.

Para el procesamiento de los datos emplean una herramienta propia creada en Matlab con la cual llevan a cabo los siguientes pasos para asegurar la calidad de los datos y el formato:

1. Leer datos en varios formatos específicos de instrumentos.
2. Leer los metadatos asociados de una base de datos.
3. Realizar conversiones, correcciones y calcular las variables derivadas.

[7] http://geoserver.org/

4. Aplicar pruebas de control de calidad.
5. Escribir archivos netCDF de acuerdo con las convenciones requeridas.

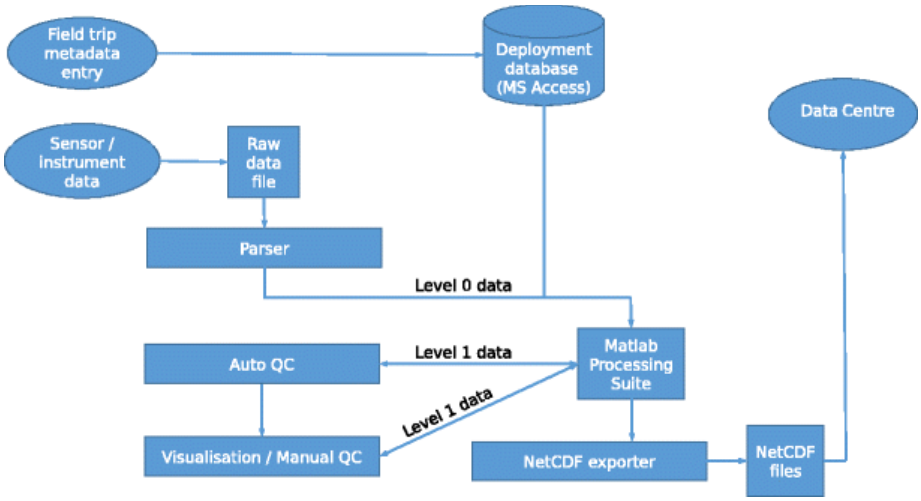

Figura 3: Flujo de los datos provenientes del sensor [11]

En la Figura 3 se muestra el flujo de los datos provenientes del sensor. En el nivel 0 se encuentran los datos sin procesar, en el nivel 1 se asegura la consistencia de los datos al pasar un control de calidad (QC). IMOS despliega muchos sensores que cuentan con diferentes instrumentos, de ahí la necesidad que han tenido de implementar una amplia gama de diferentes intérpretes para cada formato específico.

Los metadatos asociados (por ejemplo, la extensión espacial y temporal, los detalles de la estación y los instrumentos, las unidades de medida y la información sobre la calidad de los datos) se almacenan en cada archivo netCDF. Estos metadatos se extraen de todos los archivos netCDF y son publicados en una base de datos PostgreSQL/PostGIS. Almacenar esta información en una base de datos les permite el uso de consultas complejas para el filtrado de recopilación y la generación de informes sobre la disponibilidad de datos. Por ejemplo, para seleccionar sólo los archivos que contienen datos dentro de un área geográfica especificada y un intervalo de tiempo.

4.1.2. Centro de Datos Oceanográfico Británico

En la sección de Procesamiento de datos de Plataformas o Instrumentos Estacionarios de la web del BODC [12], se describe el paso a paso del procesamiento de los datos provenientes de este tipo de estaciones de medida. Una vez procesados son almacenados en la Base de Datos Oceanográfica Nacional (NODB). En la Figura 4 se muestra el flujo de este procesamiento.

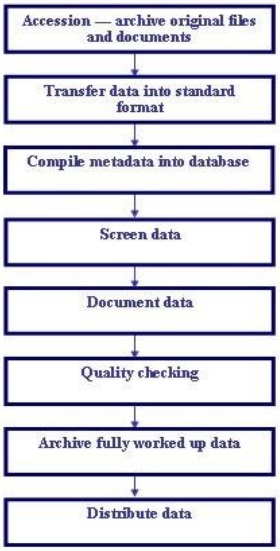

Figura 4: Procesamiento de los datos provenientes de instrumentos estacionarios [12]

Archivar los datos originales: Cuando los datos son recibidos por primera vez, pasan por el procedimiento de captación y se archivan de forma segura en su forma original junto con cualquier documentación asociada.

Transferencia de datos al formato estándar: Los datos llegan en varios formatos y se transfieren al formato estándar de la BODC. Este formato binario es el netCDF. Se utiliza para la transformación una caja de herramientas para el trabajo con netCDF que viene integrada en Matlab.

Compilar metadatos: Los metadatos (datos sobre datos), por ejemplo la fecha y hora de recogida, la posición de la estación de observación, el tipo de instrumento, la profundidad del instrumento y la profundidad del fondo marino, se cargan en las tablas de la base de datos Oracle. Éstos se comprueban cuidadosamente para saber si hay errores y la consistencia con los datos. Se contactará con los proveedores de los datos si no se pueden resolver claramente los problemas.

Visualización de Datos: utilizan el software de visualización interno de BODC. Este software se puede utilizar para:

- Mostrar un mapa de las posiciones de las estaciones
- Graficar series temporales y perfiles de los datos recopilados, comparando los datos de diferentes instrumentos de medición.

Documentación de todos los datasets o conjuntos de datos: para asegurarse de que pueden ser utilizados en el futuro sin ambigüedad o incertidumbre. La documentación completa se compila utilizando la información proporcionada por el proveedor del dato.

Comprobación de la calidad de los datos: antes de cargarlos en la base de datos, los archivos netCDF y los metadatos se comprueban minuciosamente utilizando el software Matlab para asegurar que se ajustan a los estrictos estándares de la BODC.

Distribución y entrega de datos: Algunos datos están disponibles previa solicitud, mientras otros están disponibles a través de su sitio web.

En la sección [13] de la web de la BODC se especifica que sus ficheros netCDF generados cumplen con las normas establecidas por la convención CF, cumpliendo así con las guías y recomendaciones de cómo estructurar los metadatos dentro de un fichero netCDF.

4.1.3. Administración Nacional Oceánica y Atmosférica de los Estados Unidos

Dentro de la agencia norteamericana NOAA existe un programa conocido como Sistema Integrado de Observación Oceánica (IOOS)[14] encargado de liderar la integración de las capacidades de observación oceánica, costera y de los Grandes Lagos. IOOS tiene como misión maximizar el acceso a los datos y la generación de productos de información, para la toma de decisiones y para promover beneficios económicos, ambientales y sociales.

A continuación se exponen las principales tecnologías empleadas en este programa al analizar su Modelo de Diseño de Interoperabilidad explicado en [15].

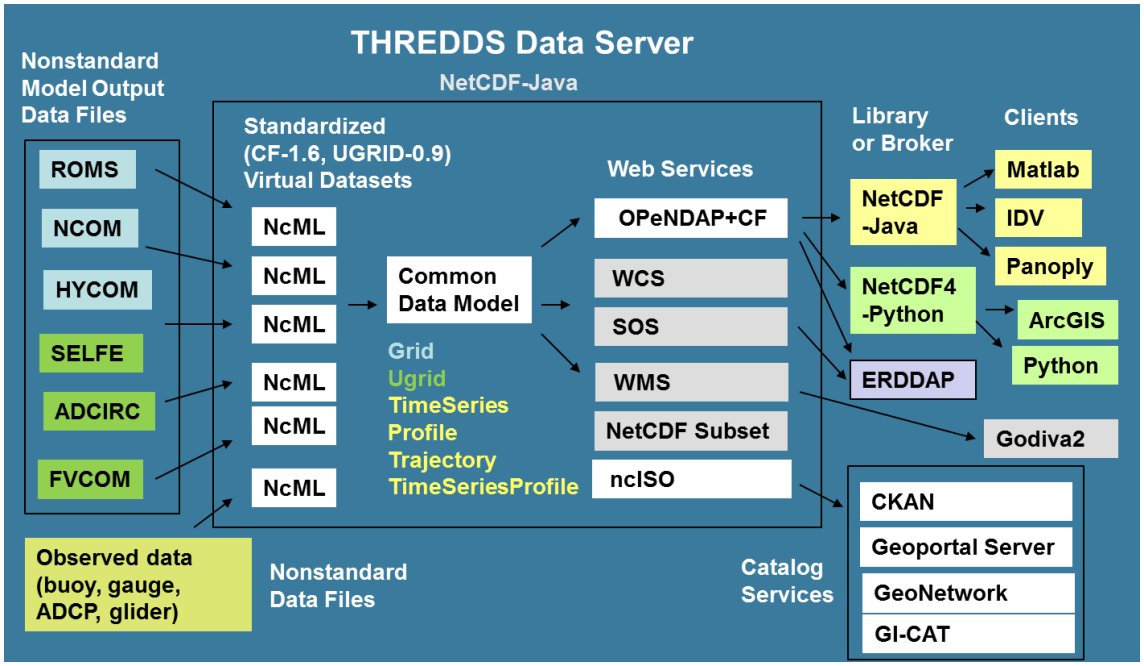

Figura 5: Tecnologías del programa IOOS [15]

En la Figura 5 se puede apreciar que los ficheros de datos que no cumplen un estándar se convierten en conjuntos de datos virtuales estandarizados que utilizan el lenguaje de marcado NetCDF (NcML)[8] y que constituye una capa ligera de XML. Se desarrolla una plantilla NcML

[8] http://www.unidata.ucar.edu/software/thredds/current/netcdf-java/ncml/

personalizada para cada tipo de modelo no estándar. Una vez que los datos hayan sido normalizados al tipo estándar mediante el uso de la convención CF, son distribuidos por servicios web y consumidos por clientes basados en esos estándares. De esta manera se proporciona interoperabilidad entre los datos para el usuario cliente. En la Figura 5 se muestra que para el caso particular de THREDDS existen una variedad de protocolos de acceso a datos remotos como son: OPeNDAP, WMS y WCS.

4.2. Servidores de datos científicos

Este tipo de servidores se caracteriza por permitir el almacenamiento y la distribución de grandes volúmenes de datos científicos. Constituyen una herramienta de conectividad entre proveedores de datos científicos y posibles usuarios finales ya sean personas u otras aplicaciones.

Según se expresa en [16], a lo largo del tiempo diferentes comunidades científicas han desarrollado diferentes tipos de servidores de datos. Cada una especializada en su campo, siendo difícil obtener datos de diferentes tipos de servidores:

- Diferentes servidores de datos necesitan la petición de los mismos en diferentes formatos.
- Diferentes servidores de datos devuelven datos en diferentes formatos.
- Diferentes colecciones de datos utilizan diferentes formatos para los datos de tiempo, por lo que los resultados son difíciles de comparar.

4.2.1. THREDDS

Según se describe en [17] , constituye un servidor web basado en Java que proporciona acceso a metadatos y colecciones de datos científicos, utilizando una gran variedad de protocolos de acceso a datos remotos como son: OPeNDAP, OGC WMS, WCS y HTTP. Estos protocolos de acceso a datos remotos están diseñados para ser consumidos desde otras aplicaciones para las cuales THREDDS sería un intermediario de los datos que almacena.

THREDDS también nos proporciona diferentes opciones de servicios de descarga que van desde descargar directamente el fichero a otras opciones que permiten seleccionar, mediante los parámetros de la URL, pequeñas subsecciones dentro del mismo. El formato de salida en estos casos puede ser netCDF3, CSV y XML.

4.2.2. ERDDAP

Es un servidor web basado en Java y creado por la NOAA, que proporciona una forma sencilla y consistente de descargar datos científicos en formatos comunes de archivo así como crear gráficas y mapas. Posee múltiples ventajas, enumeradas en [16]:

- Unifica los diferentes tipos de servidores de datos para que se tenga una forma consistente de obtenerlos, en el formato que se desee.
- Actúa como intermediario entre el usuario y varios servidores de datos remotos. Puede comunicarse con otros servidores ERDDAP o THREDDS por ejemplo.
- Cuando se hace una petición a ERDDAP, se formatea en el formato requerido por el servidor remoto, envía la solicitud al servidor remoto, obtiene los datos, formatea los

datos en el formato que se solicitó originalmente y los envía. De esta manera no es necesario ir a diferentes servidores para obtener datos de diferentes colecciones de datos.

- Ofrece una forma fácil y consistente de usar para solicitar datos: a través del estándar OPeNDAP o a través del servicio de mapas web (WMS).
- Da la posibilidad de elegir el formato de salida de los datos solicitados aportando una gran variedad como son: .html, ESRI .asc, .csv, Google Earth .kml, OPeNDAP binario, .mat, .nc, ODV .txt, .tsv, .json y .xhtml. También permite devolver una imagen .png o .pdf con un gráfico o mapa personalizado.
- Otra de las ventajas es que estandariza las fechas en los resultados, eliminando así las incongruencias que se presentan en los datos de otros servidores de datos, cuyos datos son difíciles de comparar pues a menudo se expresan en diferentes formatos.
- Para tiempos en formato cadena (string), ERDDAP utiliza siempre el formato estándar ISO 8601: 2004 (E), por ejemplo, 1985-01-02T00: 00: 00Z. Para tiempos numéricos, ERDDAP siempre utiliza "segundos desde 1970-01-01T00: 00: 00Z".
- Utiliza siempre la zona horaria Zulu (UTC, GMT) para eliminar las dificultades de trabajar con diferentes husos horarios y el tiempo estándar frente al horario de verano. Posee un servicio para convertir un tiempo numérico a/desde un tiempo de cadena.
- Tiene una interfaz de administración web para los administradores con navegadores y servicios web RESTful para software. Esto permite usar los servicios web RESTful de ERDDAP (por ejemplo, para buscar conjuntos de datos, descargar datos, hacer mapas) directamente desde cualquier programa de computadora (por ejemplo, Matlab) o incluso desde páginas web (a través de JavaScript).

4.3. Tecnologías de Desarrollo Web

A continuación se presenta la típica arquitectura cliente–servidor de tres capas y por cada una de las capas se describen algunas de las tecnologías de desarrollo web más populares a elegir dentro del mercado.

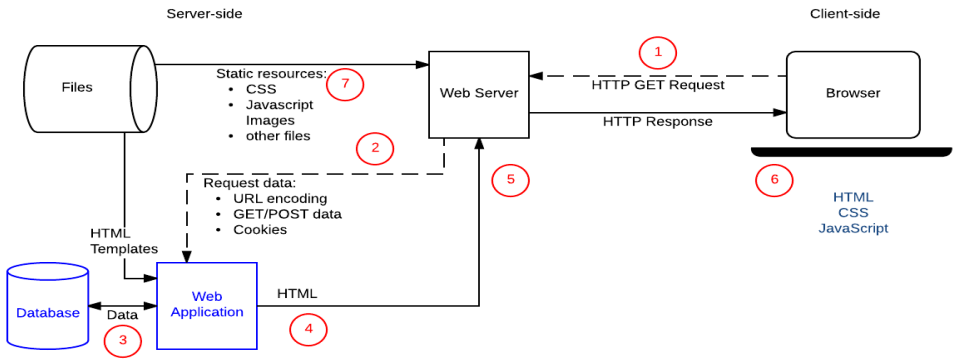

Figura 6: Elementos de una aplicación web dinámica [18]

La Figura 6 muestra los elementos principales de una aplicación web dinámica [18].

Capa del Cliente: el navegador representa al equipo que solicita los recursos y contiene el código JavaScript que se ejecuta del lado de cliente. Como tecnologías candidatas del lado del cliente tenemos:

- **Frameworks JavaScript**: Alternativas populares de este tipo de frameworks son: Angular[9] y React.
- **Librerías JavaScript de Mapas**: OpenLayers y Leaflet [10].
- **Librerías JavaScript de gráficas**: Google Charts, D3.js y HighCharts[11].

Capa del Servidor web: gestiona los recursos solicitados y atiende las peticiones de los usuarios.

- **Servidor web**: Entre las opciones más populares al escoger un servidor web se encuentran Apache y Nginx.
- **Frameworks web del lado del servidor**: Los frameworks como ASP.NET, Symfony, Express, Ruby on Rails y Django[12] representan algunos de los más populares disponibles. Todos ellos tienen lo necesario para ser productivos: son de código abierto, están en desarrollo activo, tienen comunidades entusiastas generando documentación y ayudando a los usuarios en los foros de discusión, y se utilizan en un gran número de sitios web de alto perfil.

Capa de Datos: gestiona, almacena y proporciona al servidor web los datos que solicite el cliente.

- **Servidores de base de datos**: MySQL y Postgres[13].
- **Servidores de Datos científicos:** THREDDS y ERDDAP.

En el caso de Angular y React, cada uno tiene sus propias fortalezas y debilidades. Al analizar las características que cada framework incorpora para facilitar el desarrollo de aplicaciones web del lado del cliente, Angular sobresale[14]. Entre las características que incorpora Angular se encuentran: inyección de dependencias, plantillas, sistema de rutas, peticiones AJAX, formularios, encapsulamiento de componentes, protección contra ataques Cross-site scripting (XSS) y componentes para realizar pruebas unitarias. En el caso de REACT, no proporciona inyección de dependencias, en vez de utilizar plantillas utiliza un lenguaje basado en XML que se convierte a JavaScript, proporciona protección XSS y soporte de pruebas unitarias.

A la hora de escoger una librería de mapas los dos principales contendientes son Leaflet y OpenLayers. Leaflet cuenta con una mejor documentación de su API y proporciona una curva de aprendizaje más rápida que OpenLayers. Su ecosistema de plugins es mucho más extenso[15]

[9] https://angular.io/
[10] http://leafletjs.com/
[11] https://www.highcharts.com/
[12] https://www.djangoproject.com/
[13] https://www.postgresql.org/
[14] https://www.sitepoint.com/react-vs-angular/
[15] http://leafletjs.com/plugins.html

pues cuenta con cientos de estos mientras que OpenLayers apenas llega a la docena[16]. Comparando el peso de cada una comprimida, Leaflet v1.1.0 ocupa unos 39KB mientras que OpenLayers v4.2.0 es mucho más pesada, con unos 150KB.

Existen numerosas alternativas de librerías JavaScript para la representación de datos mediante gráficas. Google Chart es una solución muy popular sin embargo presenta un importante inconveniente: de acuerdo a los Términos de servicio de Google[17], el código de la librería no se puede almacenar en nuestro propio servidor sino en los servidores de Google. D3.js es otra opción disponible, es una solución muy completa sin embargo tiene una curva de aprendizaje elevada. HighCharts por su parte utiliza el ampliamente conocido formato JSON para definir cada gráfica y cuenta con una excelente API, bien documentada. La Tabla 1 resume las características generales de estas librería [19].

Del lado de los servidores web se presentan algunas consideraciones prácticas sobre Nginx y Apache recogidas en [20]. Nginx ha crecido en popularidad en los últimos años debido a su eficiencia de recursos hardware y su capacidad de escalar fácilmente cuando estos son limitados. Sobrepasa a Apache a la hora de servir contenido estático y en el manejo de conexiones concurrentes. También lo supera en la velocidad de procesamiento de las peticiones y en temas de seguridad pues en ambos casos no depende de los conocidos ficheros de configuración a nivel de directorio de Apache, los ficheros .htaccess.

Tabla 1: Comparativa de librerías gráficas [19]

	Google Charts	D3.js	HighCharts
Compatibilidad con el navegador y los dispositivos	Compatible con todos los navegadores web y móviles modernos, ofreciendo además soporte a las versiones más antiguas de IE	Compatible con todos los navegadores web modernos. Ofrece soporte para IE9+	Compatible con todos los navegadores modernos, incluyendo sus versiones móviles. Brinda también soporte a navegadores más antiguos como IE6
Formato de entrada de datos	JS API	JSON y XML	JSON
Precio y términos de licencia	Gratis para todos los usos	BSD-3	Gratis para uso no comercial
Tipos de gráfico disponibles	13 tipos de gráfico	Sin gráficos pre-construidos, pero con una biblioteca de más de 200 ejemplos	Más de 25 tipos de gráfico
Posibilidad de exportar gráficos como JPG, PNG o PDF	No	No	Sí, además de SVG, CSV y XLS
Diseño e interactividad	Bueno	Excelente	Muy Bueno
Curva de aprendizaje	Baja	Elevada	Baja

[16] http://openlayers.org/3rd-party/
[17] https://developers.google.com/chart/interactive/faq?csw=1#offline

Por la parte de los frameworks web del lado del servidor, todas las alternativas son igualmente potentes. Presentan características que permiten desarrollar aplicaciones web con relativa facilidad y permiten al desarrollador contar con herramientas como una capa de acceso a datos, una capa para las rutas de la aplicación y un sistema de plantillas que permiten no tener que reinventar la rueda a cada paso. Sin embargo Django destaca por su fácil curva de aprendizaje, buen soporte de su comunidad, extensa documentación y por su productividad [21].

En el caso de los servidores de base de datos, Postgres de conjunto con su extensión espacial Postgis[18], constituye la solución más popular en el mercado de las base de datos geoespaciales [22].

Tabla 2: Comparativa entre MySQL y Postgres [22]

MySQL	PostgreSQL
RDBMS de código abierto escrito en C, C++	ORDBMS de código abierto escrito en C
Mejor para el almacenamiento simple	Mejor para el almacenamiento espacial
Enfocado en la velocidad de lectura	Enfocado en la fiabilidad e integridad referencial

4.4. Conclusiones

Como se pudo observar en la sección 4.1, los 3 grandes centros analizados tienen en común que utilizan el estándar netCDF como formato de almacenamiento para las mediciones de los datos oceanográficos. Estos ficheros binarios los generan apegándose a la convención CF en su versión 1.6. Los metadatos los almacenan en bases de datos tradicionales como Postgres en el caso del IMOS u Oracle en el caso del BODC. Emplean servidores científicos como THREDDS o ERDDAP para indexar, consultar y distribuir sus datos oceanográficos.

Para este proyecto se decide trabajar con ERDDAP al existir ya un servidor THREDDS en PLOCAN, siendo necesario entonces investigar esta nueva tecnología y conocer sus potencialidades. Además, ERDDAP tiene varios beneficios agregados al compararlo con THREDDS:

- **Más opciones de búsqueda**: Ofrece múltiples maneras de buscar datasets dentro de su colección de datos, en lugar de sólo navegar por una lista de árbol (jerárquica) de datasets.
- **Más formatos de archivo**: Da la posibilidad de descargar subconjuntos de los datasets en muchos formatos de archivo comunes (por ejemplo, .html, ESRI .asc, Google Earth .kml, .mat, .nc, OPeNDAP .asc y .dods, .csv, .tsv, .json y .xhtml) en lugar de sólo los formatos netCDF3, c.sv y .xml de THREDDS.
- Esta ventaja se muestra particularmente útil, pues al utilizar un framework JavaScript para la aplicación del frontend en este proyecto, las consultas a la API REST del servidor ERDDAP se realizarán solicitando que el formato de la respuesta sea JSON, el cual está contemplado en ERDDAP.

[18] http://postgis.net/

- **Un Formato de Tiempo Consistente**: En ERDDAP, el tiempo cuando está formateado como un número está siempre en "segundos desde 1970-01-01T00: 00: 00Z" y esto hace que sea fácil especificar restricciones de tiempo en las peticiones a ERDDAP, sin tener que preocuparse por el formato de tiempo (evitando los posibles formatos y zonas horarias). Los resultados de diferentes fuentes de datos son fáciles de comparar.
- **Gráficas y mapas personalizados:** Ofrece la posibilidad de crear gráficos y mapas personalizados a partir de los conjuntos de datos almacenados.

A continuación se presentan las tecnologías que se utilizan en el proyecto.

- **Angular**
- **Leaflet**
- **Highcharts**
- **Django**
- **Postgres**
- **Nginx**

Su elección ha estado condicionada en la mayoría de los casos por la experiencia previa de los desarrolladores que le darán mantenimiento a este sistema en PLOCAN. No obstante, es de destacar que estas tecnologías sobresalen al compararlas con sus respectivas contrapartes según se explicó en la sección 4.3.

En el caso de Angular, se escoge por la experiencia previa del autor de este trabajo con dicho framework en su versión 1, aunque en el momento de realizarse el presente proyecto la versión actual de Angular es la 4, cuya versión estable ha sido liberada en marzo de 2017. El framework ha sufrido un cambio de paradigmas y es como volver a aprenderlo aunque no desde cero. Afortunadamente cuenta con una excelente documentación y comunidad de usuarios. Además, se ha podido observar al compararla con opciones como REACT, que Angular trae muchas características que aceleran el desarrollo y la productividad.

Angular 2 fue anunciado por primera vez en 2014 para hacerle frente al surgimiento de nuevas tecnologías que su versión predecesora no podía incorporar. Por ejemplo el estándar ECMAScript v6 de JavaScript, también conocido como ES2015 o ES6 [23], fue aprobado en 2015 y ha impulsado la popularidad de JavaScript. Además, Angular 1 no fue diseñado para el uso de estándares web emergentes como Web Components, pero sí las nuevas versiones de Angular.

Entre las ventajas de este framework es que nos proporciona una interfaz fluida y una buena experiencia de usuario, en la que no se nota la comunicación entre cliente y servidor, al no haber necesidad de recargar la página para realizar una petición al servidor.

Para hacer una distinción entre las diferentes versiones de Angular, de ahora en adelante cuando se hable de Angular se estará refiriendo a la nueva versión sin importar si es Angular 2, 2.x o 4.x.

5. Justificación de las competencias específicas

En este apartado se describen cómo son cubiertas las competencias específicas del TFM [24] más directamente relacionadas con este proyecto.

En el caso de las competencias C08 y DG01, son cubiertas mediante el presente trabajo pues ha sido necesario aplicar los conocimientos adquiridos en el Máster para resolver problemas nuevos o poco conocidos dentro de la especialidad como son los servidores web ERDDAP. Este tipo de tecnologías son propias de contextos más generales como el contexto científico y ha sido necesario abordarla para su integración en el sistema. Su uso como herramienta para la importación, indexado y distribución de datos científicos sustituye a las tradicionales bases de datos para el tipo de datos señalado. También ha sido necesario asimilar contenidos nuevos como los ficheros binarios en formato netCDF, para el almacenamiento de los datos científicos tipo perfiles verticales.

Las competencias TI01 y TI02 son puestas en práctica al analizar el diseño del sistema a través del diagrama de despliegue. Este diagrama modela y resume los elementos más importantes de la arquitectura. Muestra los diferentes componentes que lo forman y que se necesitan gestionar, administrar y mantener como las aplicaciones Django, ERDDAP, Angular. También muestra los diferentes tipos de servicios web que interactúan entre sí como Tomcat y Nginx, incluyendo Postgres como servidor de base de datos. Se detalla el uso de la arquitectura cliente-servidor y el uso de formatos de datos y protocolos de uso estándar en Internet como JSON y HTTP para la comunicación entre los componentes participantes.

La competencia TI05 se pone de manifiesto a lo largo de toda la etapa de desarrollo de software. Desde un inicio se analizaron las necesidades de información de PLOCAN y se formalizó una lista de requisitos funcionales y no funcionales. A partir de estos se lleva a cabo en todas sus etapas el proceso de construcción del software, desde el diseño de la base de datos hasta la implementación y posterior validación y pruebas.

Los requisitos del sistema definidos en este trabajo, el diseño de los casos de uso acompañados de las respectivas interfaces web y toda la fase de desarrollo ayudan a cubrir la competencia TI11, pues ayudan a conceptualizar, diseñar y desarrollar la interacción persona–ordenador del sistema.

En cualquier caso es necesario decir que se puso de manifiesto el autoaprendizaje como solución, trabajando de manera autónoma e investigando sobre la abundante literatura en Internet y la documentación de las tecnologías utilizadas.

6. Aportaciones

En el mundo globalizado en que vivimos donde la interdependencia entre las actividades humanas es total, la introducción de mejoras en una actividad específica del hombre puede derivar inmediatamente en numerosas aplicaciones para sectores relacionados.

Visto desde un contexto científico-técnico, el sistema implementado contribuye a aumentar la productividad de aquellas tareas relacionadas con la recolección, gestión y visualización de datos oceanográficos tipo perfiles verticales. Fomenta a través del análisis de estos datos, la posterior innovación y la reducción de la incertidumbre en materia marina.

Desde el punto de vista socio-económico, mejores datos científicos marinos a su vez coadyuvan a obtener mejores modelos de predicción oceánica y meteorológica, lo que a su vez mejora la gestión y la eficiencia operativa en los sectores económicos dependientes del océano. Una mayor eficiencia produce beneficios económicos en términos de productos de mayor valor [25].

Un mejor acceso a datos marinos que se han normalizado y armonizado de conformidad con estándares como la OGC, la convención CF y normas de calidad preestablecidas estimula la inversión en actividades costeras y oceánicas abiertas y sostenibles. Además, una mayor disponibilidad de estos datos permite establecer políticas más eficientes de protección ambiental y reducir los riesgos asociados con la economía azul, a mantener la salud del medio ambiente marino [3]. En Gran Canaria todo lo anterior adquiere una notable importancia, al ser una isla donde sectores relacionados con el océano como el pesquero, las energías renovables y el turismo han adquirido gran peso en la economía. Se hace necesario entonces potenciar cada vez más el estudio del medio ambiente marino.

El software implementado ayuda a PLOCAN a continuar consolidándose a su vez como entidad científica productora de datos marinos de interés para la economía y la comunidad científica.

7. Fundamentos Teóricos

En esta sección se introducen algunos conceptos claves que ponen en contexto el tema de los ficheros netCDF y que influyen en la estructura que tendrá la base de datos. Se explica la estructura que tendrán los ficheros netCDF generados en este proyecto.

7.1. Estructura de un fichero netCDF de datos tipo perfil vertical

En la convención CF se describen los diferentes tipos de características (features) geométricas o Geometrías de Muestreo Discreto que existen para los datos científicos [10]. Existen los puntos, las series temporales, las trayectorias, los perfiles, los perfiles de series temporales y los perfiles de trayectorias. Ver Tabla 3.

Tabla 3: Geometrías de Muestreo Discreto según la convención CF

Tipo de característica geométrica	**Descripción de una característica con este tipo de geometría de muestreo discreto**	
	Forma de una variable de datos que contiene los valores definidos en una colección de estas características	**Coordenadas espacio-tiempo obligatorias para una colección de estas características**
Punto	Un solo punto de datos (que no tiene ninguna coordenada relacionada con otros puntos)	
	data(i)	x(i) y(i) t(i)
Series temporales	Una serie de puntos de datos en la misma ubicación espacial con tiempos monótonamente creciente	
	data(i,o)	x(i) y(i) t(i,o)
Trayectoria	Una serie de puntos de datos a lo largo de un camino a través del espacio con tiempos monótonamente creciente	
	data(i,o)	x(i,o) y(i,o) t(i,o)
Perfil	Un conjunto ordenado de puntos de datos a lo largo de una línea vertical en una posición horizontal fija y tiempo fijo	
	data(i,o)	x(i) y(i) z(i, o) t(i)
Perfil de Serie de Tiempo	Una serie de características del perfil en la misma posición horizontal con tiempos monótonamente creciente	
	data(i,p,o)	x(i) y(i) z(i,p,o) t(i,p)
Perfil de Trayectoria	Una serie de características del perfil situadas en puntos ordenados a lo largo de una trayectoria	
	data(i,p,o)	x (i, p) y (i, p) z (i, p, o) t (i, p)

En la Tabla 3 la dimensión con el subíndice i identifica una característica particular dentro de una colección de características. Se llama la dimensión de la instancia. Las variables unidimensionales en un archivo CF de geometría discreta, que tienen sólo esta dimensión (como x(i), y(i) y z(i) para una serie temporal) son variables de instancia.

Los subíndices o y p distinguen los elementos de datos que componen una sola característica.

- Por ejemplo, en una colección de características tipo serie temporal, cada instancia de serie temporal, i, tiene valores de datos en varias ocasiones, o.
- En una colección de características de perfil, el subíndice, o, proporciona la posición de índice a lo largo del eje vertical de cada instancia de perfil.

Nos referimos a valores de datos en una característica como sus elementos, y a las dimensiones de o y p como dimensiones de elemento. Cada característica puede tener su propio conjunto de subíndices de elementos o y p. Por ejemplo, en una colección de características de serie temporal, cada serie individual puede tener su propio conjunto de tiempos. La notación t(i, o) significa que hay un conjunto de tiempos con subíndices o para los elementos de cada característica i.

El caso de interés de este trabajo como se ha mencionado anteriormente son los perfiles verticales, que no son más que una serie de observaciones conectadas a lo largo de una línea vertical, como las realizadas por una sonda atmosférica u oceánica. Para cada perfil, hay una única latitud y longitud.

Se muestra a continuación en la Tabla 4 un ejemplo de la estructura que debe tener un fichero netCDF que almacena múltiples perfiles.

Tabla 4: Múltiples perfiles de sonda atmosférica para un conjunto común de coordenadas verticales almacenadas en representación de matriz ortogonal multidimensional

```
dimensions:
    z = 42 ;
    profile = 142 ;

variables:
    int profile(profile) ;
        profile:cf_role = "profile_id";
    double time(profile);
       time:standard_name = "time";
       time:long_name = "time" ;
       time:units = "days since 1970-01-01 00:00:00" ;
    float lon(profile);
       lon:standard_name = "longitude";
       lon:long_name = "longitude" ;
       lon:units = "degrees_east" ;
    float lat(profile);
       lat:standard_name = "latitude";
       lat:long_name = "latitude" ;
       lat:units = "degrees_north" ;

    float z(z) ;
```

```
        z:standard_name = “altitude”;
        z:long_name = "height above mean sea level" ;
        z:units = "km" ;
        z:positive = "up" ;
        z:axis = "Z" ;

    float pressure(profile, z) ;
        pressure:standard_name = "air_pressure" ;
        pressure:long_name = "pressure level" ;
        pressure:units = "hPa" ;
        pressure:coordinates = "time lon lat altz" ;

    float temperature(profile, z) ;
        temperature:standard_name = "surface_temperature" ;
        temperature:long_name = "skin temperature" ;
        temperature:units = "Celsius" ;
        temperature:coordinates = "time lon lat altz" ;

    float humidity(profile, z) ;
        humidity:standard_name = "relative_humidity" ;
        humidity:long_name = "relative humidity" ;
        humidity:units = "%" ;
        humidity:coordinates = "time lon lat altz" ;

  attributes:
    :featureType = "profile";
```

Se puede observar que se han definido tanto dimensiones como variables. En este ejemplo existen dos tipos de dimensiones: z (la profundidad) y *profile* (el perfil). La dimensión profundidad tiene un valor de 42, o sea, se han realizado observaciones en 42 profundidades diferentes. La dimensión perfil tiene un valor de 142, lo que significa que se han realizado 142 perfiles. Relacionando ambas dimensiones significa que por cada perfil realizado existen 42 observaciones, correspondientes a cada una de las profundidades.

En realidad este es un caso particular definido en la CF y se conoce como Representación de matriz ortogonal multidimensional de los perfiles, pues asume que todos los perfiles contienen mediciones realizadas en exactamente los mismos valores de profundidades.

Las variables *profile*, *time*, *lat* y *lon* dependen solamente de la dimensión *profile* (en este caso el perfil se pudiera interpretar como el identificador de la estación donde se toman las mediciones). El resto de las variables son dependientes de las dimensiones *z* y *profile*.

Se puede ver el ejemplo de la variable *temperature*, que depende de ambas dimensiones. En la Tabla 5 se muestran dos perfiles. El primero contiene mediciones de temperatura tomadas a 42 profundidades diferentes y en la estación de medida de identificador 8. El segundo perfil se ha realizado en la estación número 11 y contiene los valores de temperatura correspondientes a los mismos valores de profundidad del perfil anterior.

Tabla 5: Ejemplo de dos perfiles verticales de temperatura

Observación	Profundidad(m)	Temperatura(C)	
		Perfil #8	Perfil #11
1	100	20.7	20.5
2	200	20.2	20.3
3	300	19.7	19.5
4	400	19.2	19.3
5	500	19.0	19.1
6	600	18.9	18.7
7	700	18.8	18.6
...	...	...	...
42	...	...	...

Finalmente se presenta la estructura que se emplea en este proyecto para almacenar ficheros netCDF. Cuando un solo perfil se almacena en un único archivo, no hay necesidad de la dimensión del perfil. Las matrices de datos son unidimensionales. Este es un caso especial de la representación de matriz multidimensional ortogonal. Ver un ejemplo en la Tabla 6.

Tabla 6: Datos de un único perfil de sonda atmosférica

```
dimensions:
    z = 42 ;

  variables:
    int profile ;
       profile:cf_role = "profile_id";

    double time;
       time:standard_name = "time";
       time:long_name = "time" ;
       time:units = "days since 1970-01-01 00:00:00" ;
    float lon;
       lon:standard_name = "longitude";
       lon:long_name = "longitude" ;
       lon:units = "degrees_east" ;
    float lat;
       lat:standard_name = "latitude";
       lat:long_name = "latitude" ;
       lat:units = "degrees_north" ;

    float z(z) ;
       z:standard_name = “altitude”;
```

```
    z:long_name = "height above mean sea level" ;
    z:units = "km" ;
    z:positive = "up" ;
    z:axis = "Z" ;

  float pressure(z) ;
    pressure:standard_name = "air_pressure" ;
    pressure:long_name = "pressure level" ;
    pressure:units = "hPa" ;
    pressure:coordinates = "time lon lat z" ;

  float temperature(z) ;
    temperature:standard_name = "surface_temperature" ;
    temperature:long_name = "skin temperature" ;
    temperature:units = "Celsius" ;
    temperature:coordinates = "time lon lat z" ;

  float humidity(z) ;
    humidity:standard_name = "relative_humidity" ;
    humidity:long_name = "relative humidity" ;
    humidity:units = "%" ;
    humidity:coordinates = "time lon lat z" ;

attributes:
  :featureType = "profile";
```

En realidad esta representación es la que se necesita, pues la librería que se utilizará para generar los ficheros netCDF tiene limitaciones en relación a la cantidad de dimensiones del fichero netCDF a generar. Por cada fichero de mediciones que se suba a la aplicación web se generará un único fichero netCDF. Más adelante se profundizará en este tema al explicar el diseño físico de la base de datos.

En resumen, la estructura que sigue un fichero netCDF es parecida a la de cualquier otro fichero en formato binario en el sentido que primero contiene un encabezado y después la sección de los datos propiamente dichos de los perfiles verticales. El encabezado contiene metadatos de la sección de los datos: una serie de atributos globales que describen al conjunto de datos en su totalidad, las dimensiones y las variables. Cada variable a su vez contiene un conjunto de atributos que la describen. En este proyecto se utilizará una librería que genera un fichero netCDF a partir de los siguientes parámetros:

- fichero que describe los metadatos del mismo.
- fichero que contiene los datos, es decir, los valores numéricos de las mediciones de los perfiles.

8. Desarrollo

En este apartado se presentan las etapas por las que ha transitado el proyecto durante el proceso de desarrollo del software. Se describen de manera detallada las tareas realizadas en cada fase.

8.1. Metodología empleada. Justificación

En el desarrollo del proyecto no se siguió una metodología de desarrollo de software en específico sino que en sus puntos más relevantes combina aspectos de metodologías ágiles [26] y de la metodología Rational Unified Process (RUP)[27]. A continuación se describen los principios seguidos.

Se mantuvo *una* ***constante retroalimentación con el tutor de PLOCAN*** y el tutor de la universidad a lo largo de todo el proyecto. De manera frecuente, el cliente pudo ir corrigiendo, agregando o excluyendo elementos a medida que se iba desarrollando el software.

Inicialmente como punto de partida se realizó una ***lista de las funcionalidades a implementar*** así como el diseño de la base de datos, que constituye el cimiento de toda la información de la aplicación. Sin embargo, posteriormente durante la implementación fueron apareciendo nuevos requisitos. En varias ocasiones se identificaron puntos en donde la implementación no se correspondía con la visión de PLOCAN y se realizaron las modificaciones requeridas. Esto constituye un ejemplo del ***principio de respuesta ante el cambio*** y aceptar cambios de requisitos.

Se ***identificaron los componentes del sistema que presentaban mayor riesgo*** y se priorizó su implementación. En ese sentido se comenzó el desarrollo por la puesta a punto del servidor ERDDAP, seguidamente la administración realizada en Django y finalmente la aplicación Angular. Dentro de cada componente del sistema se ***dividió el desarrollo por módulos*** identificando siempre aquellos que presentaban dificultades técnicas. Dentro de Django, este fue el caso del desarrollo de la API REST y de la gestión de las mediciones de los perfiles. Para el caso de Angular, se identificaron como elementos con dificultad técnica la integración de las librerías externas para los mapas interactivos y la generación de gráficas.

Antes de comenzar la implementación de cada funcionalidad siempre se coordinó una reunión previa con el cliente, donde se hizo ***uso de prototipos de interfaz*** de usuario para visualizar la manera en que debería quedar y alcanzar así un consenso.

El desarrollo fue iterativo e incremental pues el método está basado en la mejora continua. Este es un punto característico de las metodologías ágiles y de RUP. El desarrollo se fue realizando a través de ***pequeñas entregas***, módulo a módulo, en base a los casos de uso identificados. Por cada entrega, se realizaba una revisión conjunta con el cliente. Para algunos módulos del sistema fue necesario realizar iteraciones adicionales, refactorizando su código y agregando nuevas funcionalidades que obligaban a declarar nuevos requisitos del sistema, realizar cambios en el diseño de la base de datos y en la implementación.

Se generaron algunos ***artefactos importantes*** y característicos de RUP como el diagrama del diseño físico de la base de datos, el artefacto de los casos de uso y el diagrama de

despliegue [28]. En general, la división del proceso de desarrollo de software siguió las etapas mencionadas en RUP: Análisis, Diseño, Implementación y Pruebas/Validación, de manera iterativa e incremental.

8.2. Análisis

En este apartado se describen los requisitos funcionales y no funcionales del software después de analizar las necesidades del cliente en relación a la información a gestionar y las tecnologías a emplear.

8.2.1. Requisitos funcionales del sistema

Esta sección establece el acuerdo logrado con el cliente en cuanto al comportamiento esperado de la aplicación, funcionalidades que proporciona y su interacción con los usuarios y otros sistemas.

Aplicación web Django

Funcionalidades del sistema de administración a implementar:

- Gestionar la información de las estaciones de medida localizadas en el Banco de Ensayos de PLOCAN, área marina al este de Gran Canaria, de aproximadamente 23 kilómetros cuadrados, reservada para el ensayo y demostración de nuevas tecnologías marinas.
- Gestionar la información de las campañas oceanográficas realizadas en las estaciones del Banco de Ensayos. Al definir una campaña debe ser posible asociarle tantas mediciones realizadas en las estaciones participantes en ella como se desee, para facilitar la usabilidad de la interfaz de usuario.
- Los ficheros que contienen las mediciones de los perfiles verticales deben subirse mediante la interfaz web.
- Permitir definir mediante la interfaz web los metadatos necesarios para crear los netCDF: atributos globales, las variables asociadas y por cada una poder especificar tantos atributos como sean necesarios.
- A partir de los ficheros de perfiles verticales subidos al sistema, almacenar su información en ficheros netCDF. Se deben generar los ficheros netCDF desde la interfaz web. Para poder generar un netCDF se utiliza una librería que tiene como entradas dos ficheros: el fichero de perfiles y un fichero intermedio de metadatos que la aplicación genera dinámicamente.
- Dar la posibilidad de gestionar la información de los sensores asociados a las mediciones.
- Para cumplir con lo planteado por la convención CF en cuanto al nombre estándar de cada variable, dar la opción de gestionarlos. En la convención CF existe toda una página dedicada a llevar el registro de estos nombres. Tener en cuenta que sin estos nombres estándar cada científico pondría el nombre que considerara más conveniente y sería muy difícil comparar la misma variable desde una aplicación si tienen diferentes nombres.
- Mantener un repositorio organizado en tres directorios, según los tres tipos de ficheros con los que se trabaja:

- Directorio de ficheros de perfiles verticales subidos a sistema.
- Directorio de ficheros de metadatos necesarios para generar los netCDF a partir de los ficheros de los perfiles verticales. Ver librería NetCDFGenerator en la sección 8.2.2.
- Directorio de ficheros netCDF.

Por cada tipo de directorio se debe mantener una estructura que contenga en el primer nivel directorios con los nombres de las estaciones. El segundo nivel debe contener directorios con los nombres de las campañas en que ha participado la estación. Finalmente en el último nivel el fichero, con el nombre de la estación a la que pertenece y el tiempo de la medición de los datos que contiene o está relacionado, concatenados como nombre.

Aplicación ERDDAP

Aun cuando ERDDAP es una aplicación desarrollada por terceros y no forma parte de las funcionalidades a implementar, se incluye aquí dado que incorpora la importante funcionalidad de distribuir los datos oceanográficos que indexa mediante la API REST que incorpora. Además, es un requisito del sistema, que el administrador del mismo pueda interactuar con la interfaz de administración de ERDDAP.

Aplicación web Angular

Los siguientes puntos recogen los requisitos funcionales de la aplicación en cuanto a la visualización y consulta de los datos tipo perfiles verticales:

- Debe poder visualizarse en un mapa la localización de las estaciones de medida.
- Permitir realizar búsquedas por estación y por campaña para una variable en concreto, dígase temperatura, humedad, salinidad, etc. En el primer caso se debe mostrar en los resultados todos los perfiles realizados en las distintas campañas por esa estación y para esa variable. En el segundo caso se debe mostrar en los resultados todos los perfiles realizados en las distintas estaciones para esa campaña y la variable seleccionada.
- Al visualizar en una gráfica los perfiles resultantes, debe darse la posibilidad de compararlos, es decir, mostrarlos todos en una misma gráfica. También debe permitirse mostrarlos de manera separada, cada uno en su propia gráfica.

8.2.1.1. Casos de Uso

Los requisitos enumerados se documentan a través de la definición de los casos de uso. Por cada uno se define una plantilla en forma de tabla con sus aspectos relevantes [29]. Por cuestiones prácticas el listado de plantillas donde se documentan se puede encontrar en el Anexo A. Antes de continuar se recomienda leer al Anexo A, para una mejor comprensión del resto del documento.

La Figura 7 muestra el diagrama de los casos de uso de la administración y resume las funcionalidades de la aplicación Django, explicadas en el Anexo A. La Figura 8 igualmente sintetiza los casos de uso descritos en el Anexo A para la aplicación Angular.

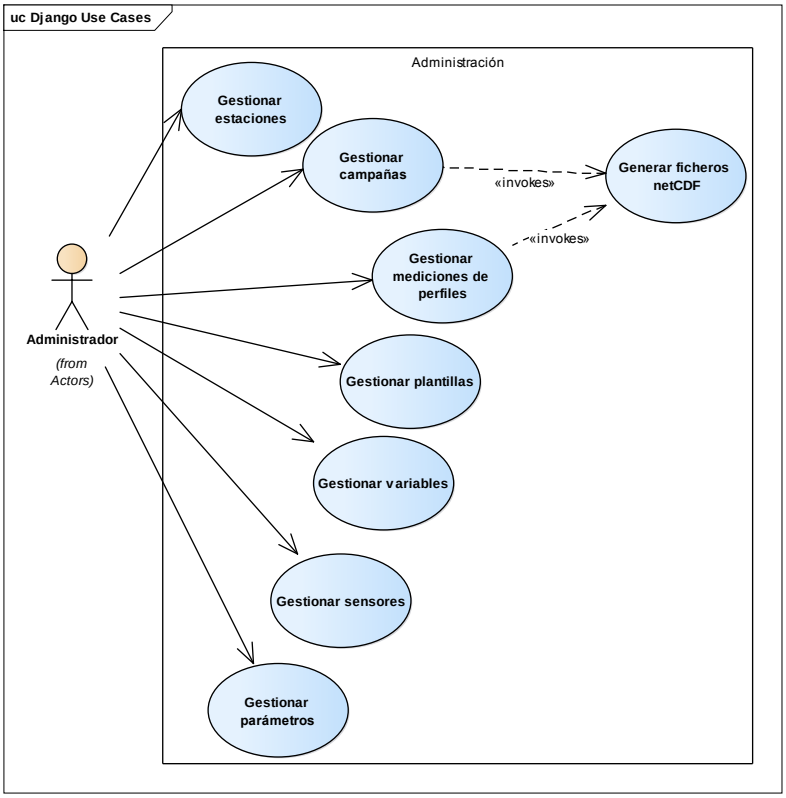

Figura 7: Casos de uso de la aplicación Django

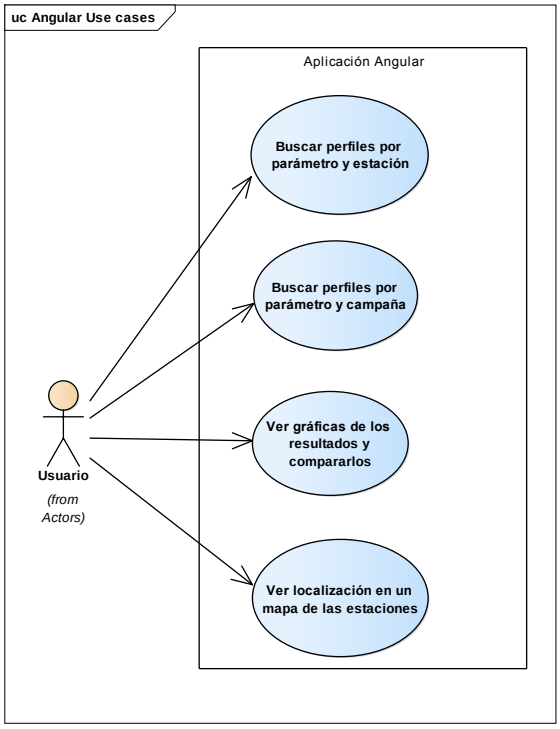

Figura 8: Casos de uso de la aplicación Angular

8.2.2. Requisitos no funcionales

Los requisitos no funcionales Imponen restricciones en el producto desarrollado. Especifican cómo debe realizarse el software.

En este proyecto se necesita que el tratamiento de los datos oceanográficos y las herramientas empleadas en la implementación se apegue a los estándares de la CF, de la convención OceanSites[19] y de la organización OGC. Para generar los ficheros netCDF se debe utilizar la librería NetCDFGenerator, desarrollada en PLOCAN.

En cuanto las tecnologías de desarrollo web, se imponen como condiciones utilizar:

- Postgres como servidor de base de datos.
- Django como aplicación web del lado del servidor.
- Leaflet como librería JavaScript de mapas.
- HighCharts como librería JavaScript para la generación de gráficas.
- Nginx como servidor web.

8.2.3. Requisitos inversos

"Este tipo de requisitos especifica lo que la aplicación no hace (son infinitos). Pueden aclarar posibles malentendidos y el alcance del sistema"[30].

En ocasiones puede darse el caso que 2 ficheros de texto de perfiles verticales contengan la misma variable pero con diferentes denominaciones. Los responsables de estos ficheros no han seguido un estándar a la hora de crearlos. La aplicación no se encarga de darle solución a esta situación pero puede encargarse de sus consecuencias.

Por ejemplo, supóngase que el administrador del sistema desea subir datos de una campaña en la que 11 ficheros contienen perfiles verticales de una variable llamada *Turb*, pues la aplicación permite crear una variable que haga referencia al nombre estándar *sea_water_turbidity* y que mapea al nombre *Turb*. Si posteriormente se suben datos de otra campaña donde 15 ficheros vienen con perfiles verticales con el nombre *turb*. pues el administrador puede crear otra variable que haga referencia al mismo nombre estándar *sea_water_turbidity*. De esta manera se pueden solventar las incongruencias derivadas.

La aplicación no automatiza la importación de los ficheros netCDF a ERDDAP.

8.3. Diseño

Para una mejor comprensión de los requisitos vistos hasta ahora y cómo se integran de manera general en el diseño a realizar, se presentan primero los componentes de alto nivel y el flujo de información dentro del sistema. A continuación se explicará el diseño de la base de datos. Este paso es fundamental antes de comenzar la fase de implementación, pues constituye el cimiento sobre el que se almacena la información y se organiza todo.

[19] http://www.oceansites.org/

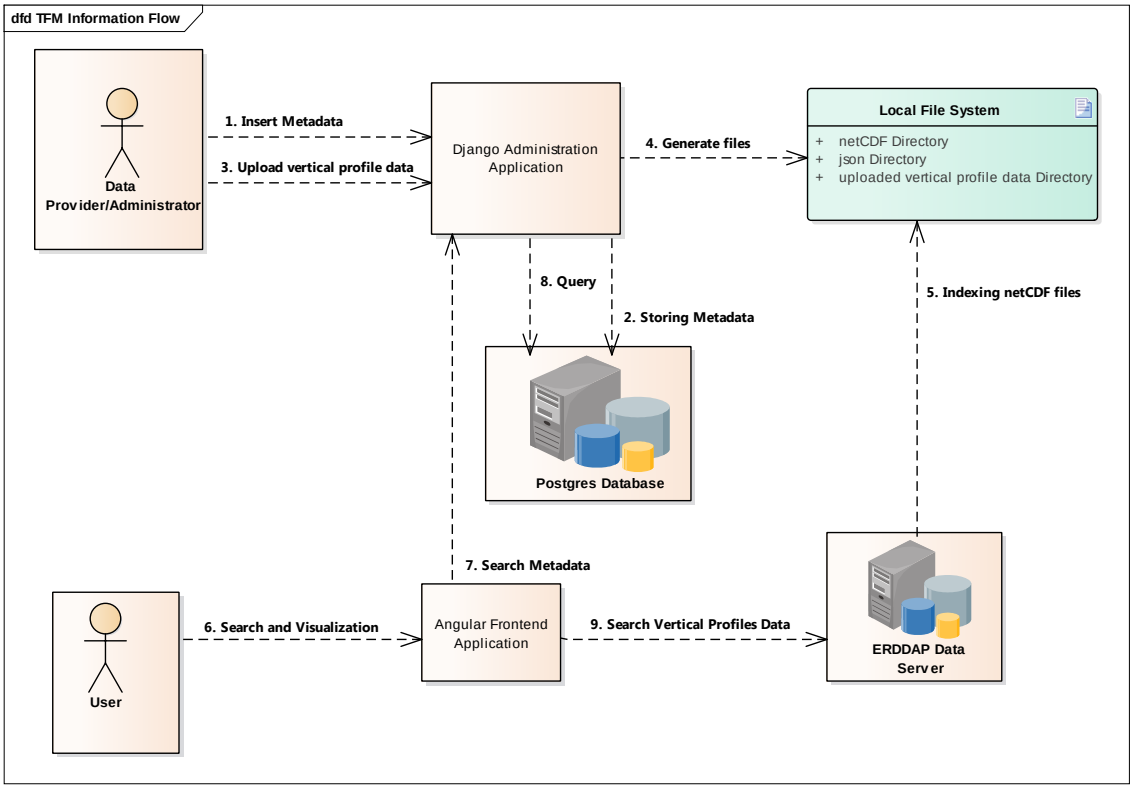

Figura 9: Componentes a alto nivel y flujo de información dentro del sistema

En la Figura 9 el flujo de información comienza por un usuario con acceso a la administración del sistema, responsable de definir los metadatos (1): sensores, variables, plantillas, nombres estándar, estaciones y campañas. Esta información es almacenada en Postgres (2). Posteriormente este usuario procede a subir los ficheros que contienen los perfiles verticales (3). La aplicación de administración entonces salva los ficheros generados dentro de un repositorio del sistema local de archivos (4). Este repositorio es indexado por ERDDAP (5). Por otra parte, la aplicación Angular que constituye la parte pública del sistema, recibe una petición de consulta por parte de un usuario que desea visualizar los perfiles verticales de PLOCAN (6). La aplicación Angular a partir del filtro escogido por el usuario, interroga al backend del sistema en búsqueda de los metadatos necesarios (7). El backend realiza una consulta a la base de datos Postgres (8) y devuelve la respuesta al frontend. Con los metadatos resultantes de la búsqueda en la base de datos, el frontend tiene toda la información que necesita para recuperar de ERDDAP los perfiles verticales requeridos por el usuario (9).

8.3.1. Diseño Lógico y físico de la base de datos

El diseño del modelo lógico de la base de datos es un paso previo del diseño del modelo físico. Ayuda a refinar la estructura de las entidades que participan en el dominio desde un nivel de abstracción más alto que la base de datos. Una ventaja de este tipo de modelo de datos lógicos es que proporciona una base sobre la cual basar el modelo físico y la posterior

implementación de la base de datos. Debe tenerse en cuenta que este modelo es independiente de la plataforma de base de datos que se vaya a utilizar. Contiene las relaciones entre las entidades pero tal cual son. Por ejemplo para representar una relación de muchos a muchos (m:m) no es necesario indicar una tabla intermedia, que sería ya una implementación específica de la base de datos [31]. En la Figura 10 se presentan las entidades del dominio del presente proyecto.

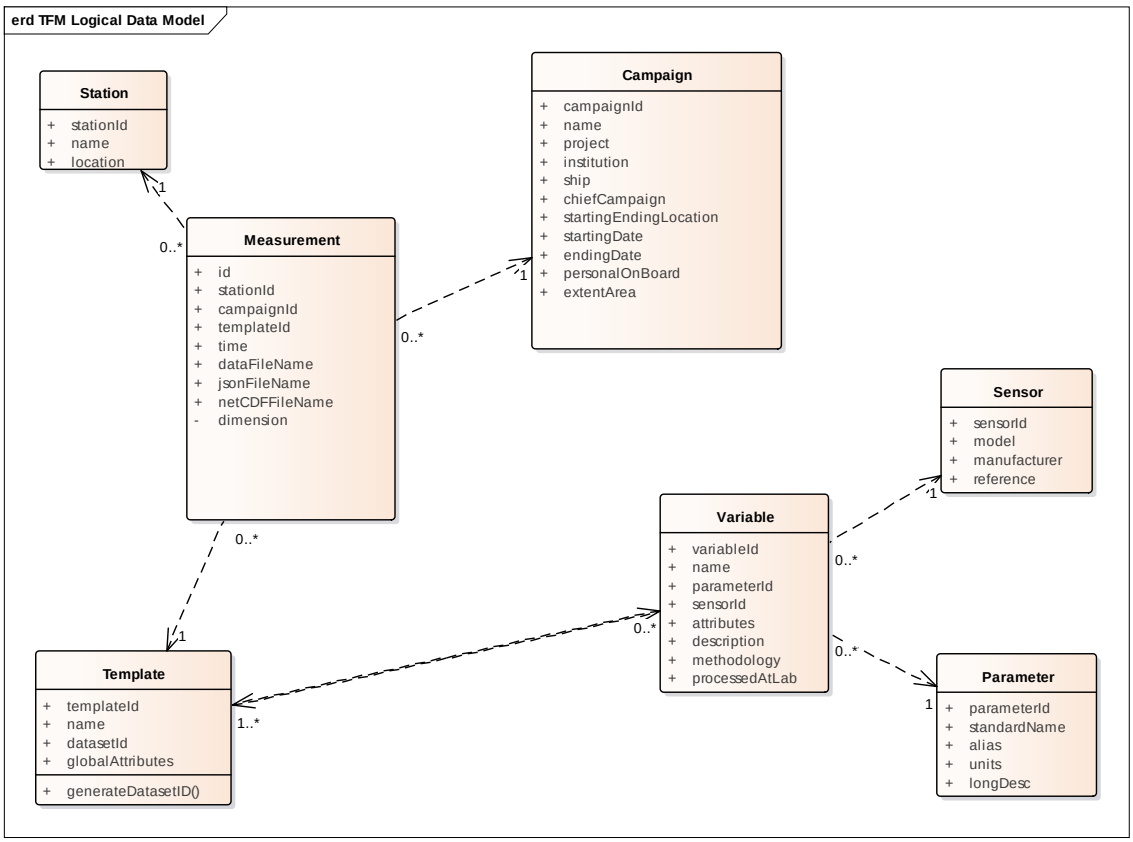

Figura 10: Modelo lógico de la base de datos

En cambio el modelo físico representa el diseño real de una base de datos relacional. Representa cómo deben estructurarse y relacionarse los datos en una base de datos específica, por lo que es importante considerar la convención y las restricciones que nos puede imponer. Esto significa que se necesita un uso preciso del tipo de datos para las columnas de cada entidad y se debe evitar el uso de palabras reservadas en las entidades de nombres y columnas. Además, contiene claves primarias, claves externas y restricciones al diseño.

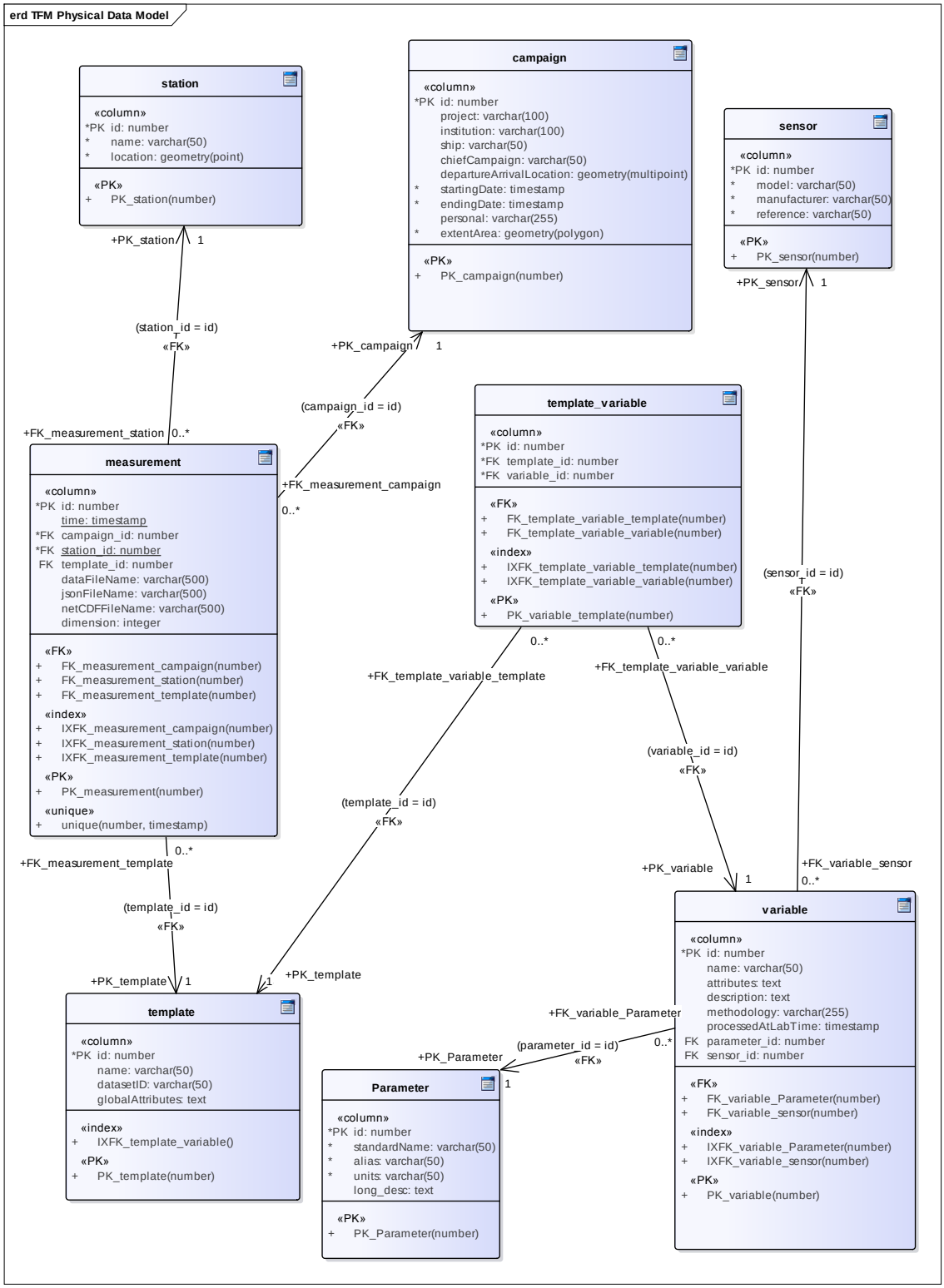

Figura 11: Modelo físico de la base de datos

Expliquemos a partir del esquema de la Figura 11, de la base de datos almacenada en Postgres, qué representan cada una de las tablas y las relaciones que se establecen entre ellas. Todas las tablas contienen como llave primaria un campo de tipo secuencia nombrado id. En el caso de las tablas intermedias de las relaciones mucho a mucho, este campo id se mantiene (Es el comportamiento del framework Django cuando genera las tablas intermedias de las relaciones de muchos a muchos a partir de las entidades del dominio. La unicidad de los registros Django la garantiza, utilizando una sentencia unique propia de Postgres, sobre el conjunto de campos que no deben estar duplicados en los registros de una tabla).

station: representa una estación de medida. Contiene los siguientes campos:

- **name**: el nombre de dicha estación.
- **location**: campo geométrico de tipo Point, con la localización en el mapa.

campaign: representa una campaña oceanográfica. Sus campos son:

- **name**: el nombre de la campaña.
- **project**: proyecto al que pertenece.
- **institution**: institución.
- **ship**: nombre del barco, en caso de haber sido utilizado durante la campaña.
- **chiefCampaign**: jefe de campaña.
- **StartingEndingLocation**: campo geométrico de tipo MultiPoint, que contiene el puerto desde donde zarpa el barco y puerto donde termina su recorrido.
- **startingDate**: fecha de comienzo de la campaña.
- **endingDate**: fecha del fin de la campaña.
- **personalOnBord**: personal a bordo del barco.
- **extentArea**: campo geométrico de tipo Polygon, con el área de extensión de la campaña.

measurement: representa los datos de una medición (que contiene múltiples perfiles verticales) realizada por una estación en una campaña específica. Esta es la tabla intermedia de la relación de muchos a muchos entre **station** y **campaign**. Una estación participa en muchas campañas y en una campaña participan muchas estaciones. En relación a las restricciones, en una campaña pueden existir una o más mediciones de una misma estación pero está prohibido que una misma estación tenga asociada dos mediciones realizadas en el mismo tiempo. Sus campos son:

- **station_id**: identificador de la estación donde se realiza la medición.
- **campaign_id**: identificador de la campaña donde se realiza la medición.
- **time**: tiempo específico en el que se realizó la medición.
- **dataFileName**: ruta del fichero de texto que contiene los datos de la medición.
- **jsonFileName**: ruta del fichero de metadatos utilizado para convertir los datos de la medición a formato netCDF.
- **netCDFFileName**: ruta del fichero binario con los datos de la medición en formato netCDF.
- **dimension**: número de observaciones realizadas a diferentes profundidades en la medición.

- **template_id**: identificador de la plantilla que describe los datos medidos.

template: especifica las variables utilizadas en una medición (a través de su relación con la tabla *variable***)**. Sus campos son:

- **name**: nombre de la plantilla.
- **datasetID**: identificador del dataset o colección de datos de ERDDAP donde se almacenarán las mediciones asociadas a esta plantilla, en formato netCDF.

  ERDDAP define un dataset como una colección de datos que puede apuntar a datos remotos o locales. En este proyecto los datasets apuntan a datos almacenados localmente en el sistema de archivos. Cada dataset apuntará a un directorio raíz de nombre único reflejado por el campo **datasetID** donde se encuentran los ficheros netCDF. Las restricciones que impone ERDDAP es que todos los ficheros netCDF pertenecientes a un dataset deben ser del mismo tipo y contener el mismo conjunto de variable ya sea en cuanto a su nombre o número de ellas. De no ser así, el dataset no es cargado ni puede ser accedido por ERDDAP.
- **globalAttributes**: campo JSON con los atributos globales de los datos de la medición según el estándar de la convención CF y OceanSites.

variable: almacena la información referente a las variables:

- **name**: nombre de la variable.
- **attributes**: campo JSON que sigue el estándar de la convención CF y OceanSites para describir una variable de datos tipo perfil vertical, de un fichero netCDF.
- **description**: descripción del propósito de la variable.
- **processedAtLab**: especifica si la variable fue procesada en el laboratorio.
- **methodology**: metodología empleada en caso de haber sido procesada en el laboratorio.
- **parameter_id**: identificador del parámetro de la variable.
- **sensor_id**: identificador del sensor asociado a la variable.

template_variable: recoge la información sobre las variables que pertenecen a una plantilla. Representa la tabla intermedia de una relación de muchos a muchos entre **template** y **variable**. Una variable pertenece a muchas plantillas y una plantilla contiene muchas variables.

- **template_id**: identificador de la plantilla.
- **variable_id**: identificador de la variable.

sensor: declara la información referente a los tipos de sensores del sistema:

- **model**: modelo del sensor
- **manufacturer**: nombre del fabricante del sensor
- **reference**: identificador de referencia del sensor

parameter: contiene información base relacionada a una variable. Expone las normas de cómo debe identificarse una variable, según la convención CF:

- **standardName**: nombre estándar de la variable según la convención CF.
- **alias**: versión del campo anterior más apropiada para ser recordada por las personas.
- **units**: unidad de medida estándar.
- **long_desc**: descripción estándar del propósito de la variable.

8.4. Implementación

8.4.1. Diagrama de despliegue

La Figura 12 presenta los componentes principales del sistema desarrollado.

8.4.1.1. Servidores

Del lado del servidor se tiene la aplicación implementada en Django y que se ejecuta en un servidor Nginx. En ese mismo servidor Nginx (aun cuando se muestra de manera separada en el diagrama) se encuentra alojada la aplicación implementada en Angular pero ejecutándose en un puerto diferente. También se muestra el servidor de base de datos Postgres y el servidor de datos ERDDAP. Este último se ejecuta sobre el servidor web Tomcat. Resumiendo las responsabilidades de cada servidor:

- **Nginx**: gestiona las peticiones HTTP de las aplicaciones implementadas en Django y Angular respectivamente.
- **Postgres**: contiene la información referente a los metadatos de los perfiles verticales. Dígase estaciones de observación, campañas, variables, sensores y demás tablas recogidas en la Figura 11.
- **Tomcat**: gestiona las peticiones HTTP a la aplicación ERDDAP.
- **ERDDAP**: aplicación que permite interactuar con los datos tipo perfiles verticales almacenados en formato netCDF.

En cuanto a los servidores proveedores de servicios WMS remotos, representan a servidores remotos especializados en dar servicios de mapas. Un servicio WMS devuelve una imagen con información geográfica, pero esta solo contiene la propia información visual para que el cliente pueda mostrarla [32].

Es importante destacar en esta fase el tema del Control de Acceso HTTP, debido a que existen tres aplicaciones web con distintos dominios y que interactúan entre sí mediante llamadas a API. Por razones de seguridad, los navegadores web restringen las solicitudes HTTP de origen cruzado iniciadas dentro de un script. Por lo que, una aplicación usando XMLHttpRequest solo puede hacer solicitudes HTTP a su propio dominio [33].

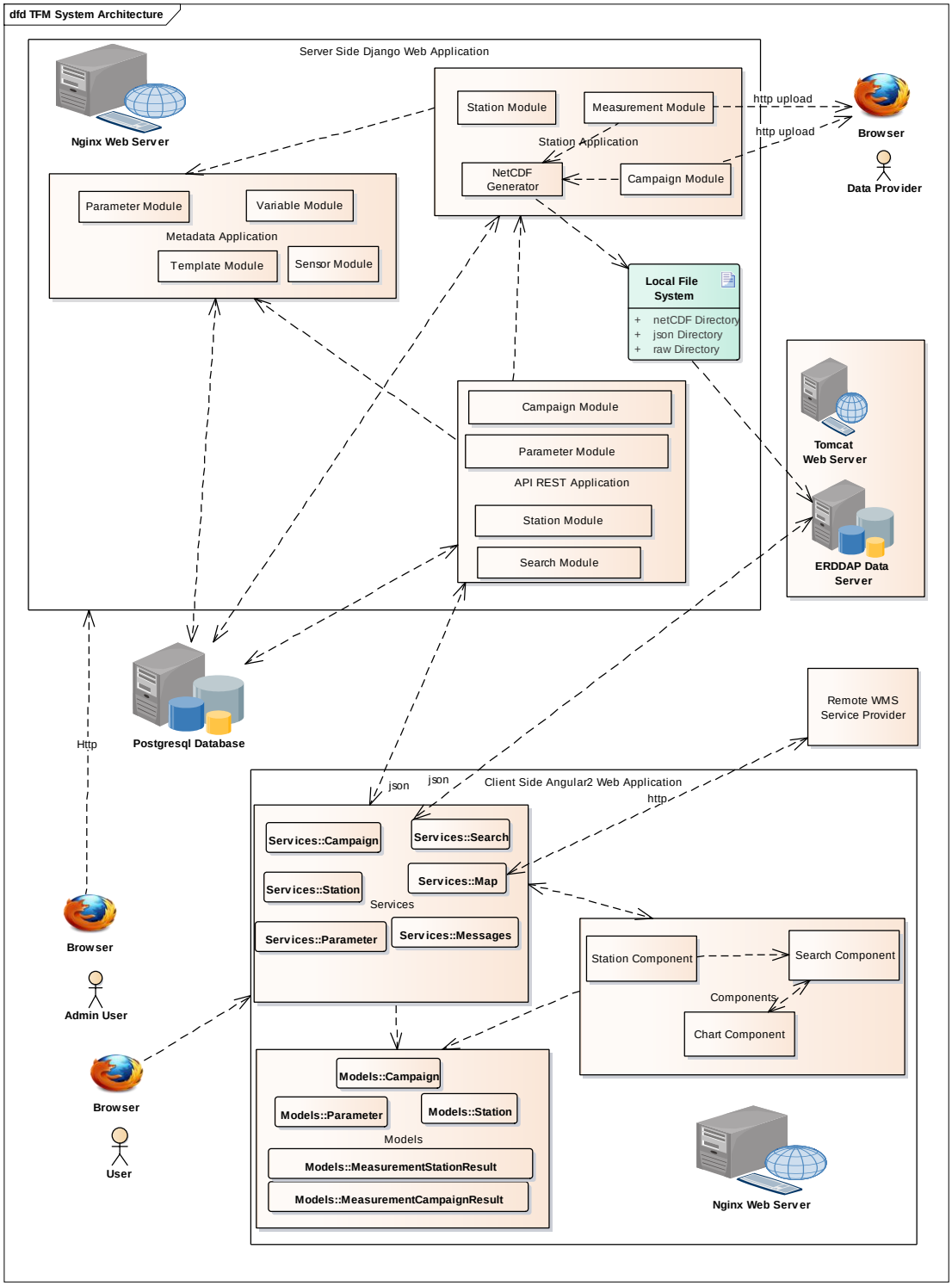

Figura 12: Diagrama de despliegue

Es necesario utilizar el mecanismo de Intercambio de Recursos de Origen Cruzado (CORS, por sus siglas en inglés). CORS permite realizar controles de acceso a dominios cruzados para servidores web, lo que garantiza la transferencia segura de datos en dominios cruzados. Esta configuración se debe realizar del lado del servidor. Por lo tanto fue necesario habilitar el soporte a peticiones CORS en los servidores Tomcat donde se ejecuta la API REST de ERDDAP y a nivel de la aplicación web Django, donde se encuentra la otra API REST.

8.4.1.2. Aplicaciones web

En el diagrama se muestran tres tipos de actores potenciales que podrían interactuar con el sistema. El administrador que tendría acceso a la aplicación hecha en Django. Los usuarios proveedores de datos oceanográficos como científicos e investigadores que interactúan con los módulos Measurement y Campaign para subir dichos datos. El tercer actor que se muestra en el diagrama es el típico usuario que navega por internet y desea acceder a la parte pública implementada en Angular.

Seguidamente se explican las interacciones y responsabilidades más significativas de los elementos del diagrama. La administración de la aplicación desarrollada en Django se encuentra dividida lógicamente en tres aplicaciones:

1. Aplicación **Metadatos**: agrupa los módulos responsables de la gestión de los metadatos asociados a un fichero netCDF. Si se recuerda, un fichero netCDF declara una serie de variables pero también contiene unos atributos globales a todas las variables.

- Módulo **Variable:** gestiona la información relacionada a cada variable.
- Módulo **Sensor**: gestiona la información de cada sensor asociado a una variable.
- Módulo **Parameter**: gestiona la información relacionada con los nombres estándar asociados a cada variable y definidos en la CF.
- Módulo **Template**: declara las variables que contiene un fichero netCDF así como los atributos globales del mismo.

2. Aplicación **Station**: agrupa los módulos relacionados directamente a una estación.

- Módulo **Station**: responsable del manejo de la información que define a una estación de medida: su localización y nombre.
- Módulo **Campaign**: encargado de manejar la información de las campañas donde participan las estaciones. Para cada campaña permite administrar los campos definidos en la tabla campaign de la base de datos. Permite igualmente asociar múltiples mediciones de determinada estación a la campaña.
- Módulo **Measurement**: Administra los registros de las mediciones de las estaciones que se han subido al sistema y sus correspondientes ficheros JSON y netCDF. Este módulo al igual que el módulo Campaign, interactúa con la librería NetCDFGenerator para la creación de los ficheros netCDF.
- Librería **NetCDFGenerator**: Esta librería es capaz de generar ficheros netCDF, aunque inicialmente solo se había probado para trabajar con datos tipo series temporales. Fue necesario hacerle algunos ajustes en su implementación para poder utilizar datos tipo perfiles verticales de una única dimensión. Genera los ficheros netCDF en un directorio

del sistema local de archivos donde se encuentra Django. A este directorio debe tener acceso ERDDAP para poder indexarlos.

3. Aplicación **API REST**: actúa como intermediaria entre Postgres y la aplicación Angular. Ante las peticiones que recibe de la aplicación Angular, se encarga de acceder a la base de datos, realizar la consulta pertinente y devolver la respuesta JSON.

- Módulo **Campaign**: encapsula la lógica de acceder a la base de datos y recuperar la información de las campañas, serializar esa información y devolver el JSON pertinente.
- Módulo **Parameter**: responsable de la lógica de acceder a la base de datos y recuperar la información de los parámetros, serializar esa información y devolver la respuesta JSON correspondiente.
- Módulo **Station**: encargado de acceder a la base de datos y recuperar la información de las estaciones, serializar esa información y devolver el JSON.
- Módulo **Search**: este es el módulo que atiende las solicitudes de búsqueda que realiza la aplicación Angular. Ante los dos tipos de búsquedas mencionadas en los requisitos de la aplicación Angular, realiza la consulta pertinente y devuelve la respuesta serializada.

La aplicación desarrollada en Angular se encuentra dividida lógicamente en tres partes:

1. **Services**: este paquete agrupa los servicios de la aplicación. En Angular los servicios son clases especiales encargadas de encapsular algún tipo de funcionalidad y brindarla como un servicio al resto de la aplicación gracias al patrón de diseño conocido como inyección de dependencias[20]. Por lo general aunque no exclusivamente, se usan para el acceso a datos remotos o locales.

- Servicio **Campaign**: encapsula el acceso a la API REST de Django para recuperar las campañas existentes.
- Servicio **Station**: encapsula el acceso a la API REST de Django para obtener las estaciones existentes.
- Servicio **Parameter**: accede a la API REST de Django para obtener los parámetros definidos en la base de datos.
- Servicio **Map**: interactúa con servidores remotos WMS para recuperar las capas base que serán utilizadas en el mapa.
- Servicio **Search**: interactúa con la API REST para el acceso a los datos de Postgres, enviando los parámetros de búsqueda y una vez obtenida la respuesta, pasa el resultado al componente Search. Este servicio también se utiliza para interactuar con la API REST de ERDDAP y realizar las correspondientes búsquedas.
- Servicio **Messages**: Encargado de la comunicación entre los componentes de la aplicación. Más adelante se verán los componentes definidos pero se puede adelantar por ejemplo que el componente Search al obtener los resultados de la búsqueda de la API REST de Django, utiliza este servicio para pasar los resultados al componente Charts.

[20] https://angular.io/docs/ts/latest/guide/dependency-injection.html

2. **Models**: este paquete contiene los modelos de la aplicación. Los modelos son clases que definen los atributos que contiene determinada entidad. Es en esta capa donde se guardan los datos de la aplicación. También ayudan a mapear las respuestas de las API RESTs a objetos en JavaScript.

- Modelo **Campaign**: representa el tipo de las instancias devueltas por el servicio Campaign.
- Modelo **Station**: representa el tipo de las instancias devueltas por el servicio Station.
- Modelo **Parameter**: representa el tipo de las instancias devueltas por el servicio Parameter.
- Modelo **MeasurementStationResult**: representa el tipo de las instancias devueltas por el servicio Search, cuando se buscan las estaciones que participaron en determinada campaña.
- Modelo **MeasurementCampaignResult**: representa el tipo de las instancias devueltas por el servicio Search, cuando se buscan las campañas en las que participó una determinada estación.

3. **Components**: este paquete agrupa los componentes de la aplicación. Constituyen el componente más básico de una interfaz de usuario en una aplicación Angular. Una aplicación Angular es un árbol de componentes. Los componentes siempre tienen una plantilla.

- Componente **Station**: representa el componente que muestra el mapa en la interfaz. Este componente utiliza al servicio Maps para obtener las capas que mostrará en el mapa. También utiliza el servicio Stations para mostrar las estaciones sobre el mapa. Se comunica con el componente Search, pues al pasar con el ratón sobre un marcador de una estación en el mapa, la estación se selecciona inmediatamente en la lista desplegable del formulario de búsqueda.
- Componente **Search**: representa el componente que contiene el formulario de búsqueda. Interactúa con el servicio Search para obtener los resultados de la búsqueda, los que a continuación pasa al componente Charts. También utiliza los servicios Station, Parameter y Campaign para la información a mostrar en las listas desplegables de los formularios de búsqueda.
- Componente **Chart**: este componente es el encargado de mostrar las gráficas de la aplicación. Una vez que el componente Search interroga a la API REST de Django, con dicha respuesta el componente Chart realiza la correspondiente búsqueda en ERDDAP gracias al servicio Search.

8.4.2. Herramientas de trabajo y plataforma de trabajo

Pycharm: es un IDE muy completo, creado por JetBrains [34] y en este proyecto ha sido utilizado para desarrollar la aplicación Django. Proporciona completamiento inteligente e inspección del código, resaltado dinámico de errores junto con refactorizaciones. Ofrece amplias opciones para depurar el código y establecer puntos de interrupción dentro del editor y en las plantillas de Django. Incorpora soporte de bases de datos como Postgres, de frameworks de desarrollo web como Django. Incluye completamiento de código para etiquetas en tecnologías como HTML5, JavaScript y CSS. Se integra con herramientas de control de código fuente como Git.

Visual Code: es un IDE desarrollado por Microsoft que se utilizó en este trabajo para la implementación de la aplicación Angular. Es multiplataforma (Windows, Linux y Mac) y gratuito, con soporte integrado para control de versiones, auto-detección, coloreado de sintaxis y autocompletado de código [35]. Soporte para depurar el código. Proporciona extensiones que facilitan el desarrollo con Angular y genera fragmentos de código para las tareas más básicas a desarrollar en Angular como componentes y servicios [36].

Google Chrome Developer Tool: es un conjunto de herramientas de creación web y depuración integrado en Google Chrome. Ofrece el panel Console para registrar información de diagnóstico durante el desarrollo, un panel Sources para depurar el código JavaScript con puntos de interrupción y un panel Network para obtener información sobre recursos solicitados y descargados, y optimizar el rendimiento de carga de la página [37].

Ubuntu: Como plataforma de desarrollo se utilizó Ubuntu 16.06 pues coincidirá con el entorno de producción. De esta manera se puede imitar exactamente el entorno de producción, los mismos permisos del sistema de archivos, la misma versión de Python y el mismo servidor web. Esto hace que el despliegue sea un proceso más sencillo, más predecible y sin errores.

GitLab: Para el control de versiones del código fuente se utiliza la versión en la nube de GitLab[21], herramienta basada en Git que proporciona un control completo sobre los repositorios o proyectos. Permite escoger de manera gratuita si son públicos o privados.

8.4.3. Aspectos relevantes de la implementación

8.4.3.1. Librería netCDFGenerator

El propósito de esta librería es convertir los datos tipo perfil vertical que se encuentran en un fichero de texto al formato netCDF. Para ello la librería necesita un fichero JSON que describe la estructura del fichero de texto. Estos dos ficheros son las entradas de la librería. Con estas entradas la librería es capaz de generar el fichero netCDF.

A continuación se describen las dependencias que son necesarias instalar para poder utilizar la librería netCDFGenerator[38].

- NetCDF4 v1.2.6: paquete de Python que permite leer y escribir archivos tanto en el nuevo formato netCDF 4 como en el antiguo formato netCDF 3, y puede crear archivos que sean legibles por los clientes HDF5 [39].
- HDF5 v2.6.0: El paquete h5py es una interfaz de Python para el formato de datos binarios HDF5. Permite almacenar grandes cantidades de datos numéricos, y manipular fácilmente los datos NumPy. Por ejemplo, puede dividir conjuntos de datos de varios terabytes almacenados en disco, como si fueran verdaderos arrays NumPy. Miles de conjuntos de datos se pueden almacenar en un solo archivo, categorizados y etiquetados como se desee [40].
- Numpy v1.12: Librería encargada de añadir toda la capacidad matemática y vectorial a Python haciendo posible operar con cualquier dato numérico o array [41].

[21] https://about.gitlab.com/

- Pandas v0.19.1: Permite leer y escribir datos entre estructuras de datos en memoria y diferentes formatos: archivos CSV y texto, Microsoft Excel y el formato HDF5 [42]. Proporciona estructuras de datos de alto rendimiento y es fácil de usar.

Fue necesario realizar algunas modificaciones dentro de la librería, con el fin de preparar los datos contenidos en las mediciones antes de su conversión a netCDF:

- Los datos de la fecha y el tiempo vienen separados, por lo que se implementó una rutina en Python y utilizando la librería Pandas que además de unificarlos, los convierte a segundos tomando como referencia la época 1970-01-01 00:00:00.
- Se agregó soporte para los diferentes formatos de fecha que pueden llegar en las mediciones.
- Soporte para leer diferentes tipos de ficheros de texto

8.4.3.2. Aplicación web implementada en Django

Para la implementación de la aplicación se ha utilizado el framework web Django en su versión 1.10 de conjunto con Python 2.7.

Django usa el modelo de programación Modelo Vista Plantilla o MVT por sus siglas en inglés. Ver Figura 13 [43]. El modelo representa la capa de acceso a datos y constituye la fuente única y definitiva de los datos. Contiene los campos esenciales y el comportamiento de los datos que está almacenando. Generalmente, cada modelo representa a una única tabla de la base de datos [44].

- Cada modelo es una clase Python, subclase de *django.db.models.Model*.
- Cada atributo del modelo representa un campo de la respectiva tabla de la base de datos.
- Con todo esto, Django ofrece una API de acceso a base de datos generada automáticamente.

La plantilla es la capa de presentación y define decisiones relacionadas con la presentación: cómo se debe mostrar la información en una página web. La vista representa la capa de lógica del negocio y declara la lógica que accede al modelo y carga la plantilla apropiada o devuelve una respuesta en un tipo de contenido específico como el formato JSON por ejemplo.

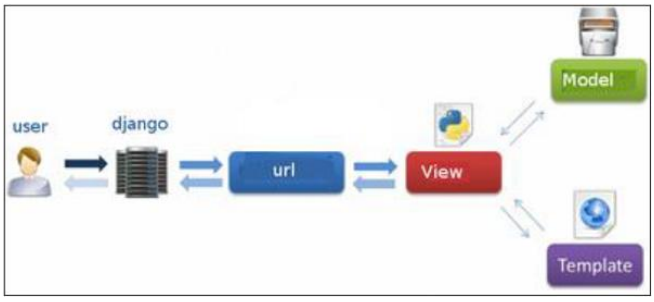

Figura 13: Componentes del patrón de diseño MVT de Django [43]

El código de la Figura 14 muestra en un único fichero definiciones de clases que residen en distintas aplicaciones, pero solo para poder presentar el siguiente ejemplo de mapeo objeto-relacional. Primeramente se observa que dichas clases implementan las especificaciones de la fase de diseño de la base de datos vistas en la Figura 10. En la línea 36 se define una relación de llave foránea entre Measurement y Template. En esa misma línea se declara la restricción de integridad referencial con la instrucción *on_delete = models.PROTECT*, esto evita que se elimine de la base de datos una plantilla que tenga asociada mediciones. A su vez en la línea 26 se define que existe una relación de muchos a muchos entre Template y Variable. Las características de integridad de datos son definidas como parámetros del atributo: valores nulos, valores únicos, longitud y precisión de los datos.

```
1   class Variable(models.Model):
2
3       name = models.CharField(max_length=50, blank=False, null=False,default="temp")
4       parameter = models.ForeignKey(Parameter,null=True, blank=True, on_delete=models.PROTECT)
5       sensor = models.ForeignKey(Sensor, on_delete=models.PROTECT)
6       processedAtLab = models.DateTimeField(null=True, blank=True)
7       methodology = models.TextField(null=True, blank=True)
8       description = models.TextField(null=True, blank=True)
9       attributes = JSONField(null=True, blank=True, default=json.dumps(getJsonAttributesTemplate()))
10
11      ...
12
13  class Parameter(models.Model):
14
15      standardName = models.CharField(max_length=100)
16      alias = models.CharField(max_length=100)
17      units = models.CharField(max_length=50)
18      long_desc = models.TextField(null=True)#refer to long_name
19
20      ...
21
22  class Template(models.Model):
23
24      name = models.CharField(max_length=20, blank=True)
25      dataSetID = models.CharField(max_length=30,unique=True)
26      variables =  models.ManyToManyField(Variable, null=True, blank=True)
27      attributes = JSONField(null=True, blank=True, default=getJsonGlobalAttributesTemplate())
28
29      ...
30
31  class Measurement(models.Model):
32      objects = MeasurementQuerySet.as_manager()
33      station = models.ForeignKey(Station, on_delete=models.PROTECT)
34      campaign = models.ForeignKey(Campaign, on_delete=models.PROTECT)
35      time = models.DateTimeField(null = True, blank=True)
36      template = models.ForeignKey(Template, on_delete=models.PROTECT)
37      dataFileName = models.FileField(upload_to=generate_path,  storage=MeasurementFileSystemStorage(), null=True, blank=True, max_length=500)
38      jsonFileName = models.FilePathField(path=settings.JSON_TEMPLATE_FILE_DIRECTORY,match="\.json", null=True, blank=True, max_length=500)
39      netCDFFileName = models.FilePathField(path=settings.NETCDF_FILE_DIRECTORY, match="\.nc", null=True, blank=True, max_length=500)
40      dimension = models.IntegerField()
41
42      ...
```

Figura 14: Ejemplo de definición de las clases de los modelos en Django

Con el propósito de demostrar la potencia de la API que genera Django automáticamente gracias al mapeo objeto-relacional, en la Figura 15 se muestra una vista definida en la aplicación de la API REST. La vista MeasurementStationViewSet es la encargada de buscar todas las estaciones que hayan participado en la campaña con identificador campaignId y que hayan realizado mediciones del parámetro parameterId.

En las líneas 15 y 16 se leen los parámetros que vienen por la URL desde la petición en Angular. En la línea 19 se define la consulta a la tabla measurement haciendo un join con la tabla station pues es necesario devolver en la respuesta información relativa a ambas. En la línea 20 se filtran los resultados del join anterior de acuerdo a la campaña y parámetro de interés.

```python
from station.models import Station, Measurement

from rest_framework.viewsets import ReadOnlyModelViewSet
from rest_app.serializers import MeasurementStationSerializer

class MeasurementStationViewSet(ReadOnlyModelViewSet):
    """
    API endpoint that allows search all measurements of parameterId from a given campaignId.
    """
    serializer_class = MeasurementStationSerializer

    def get_queryset(self):


        campaignId = self.request.query_params.get('campaign',None)
        parameterId = self.request.query_params.get('parameter',None)

        if campaignId and parameterId:
            queryset = Measurement.objects.select_related('station').select_related('template')
            queryset = queryset.filter(campaign__id=campaignId, template__variables__parameter=parameterId)
        else:
            return Measurement.objects.none()

        return queryset
```

Figura 15: Ejemplo de una clase View en Django

Interfaz de Administración de Django

Una de las partes más potentes de Django es el paquete Admin que viene por defecto en la instalación. Es capaz de generar de manera automática la interfaz de administración [45]. Lee los metadatos de los modelos para proporcionar una interfaz rápida centrada en el modelo en la que los usuarios pueden administrar el contenido de su sitio. Se genera de manera automática las páginas de listar los registros del modelo, editarlos y crearlos.

Si se necesita personalizar la interfaz de la administración, el módulo Admin da la posibilidad de extenderlo. En este proyecto fue necesario extender la plantilla de la administración que muestra el listado de estaciones, de tal manera que se pudiera ver, además de la tabla de estaciones, un mapa con sus posiciones. En varias ocasiones fue necesario igualmente extender las clases de ModelForm [46] y ModelAdmin [47] de Django, para poder ejecutar lógica propia del negocio de los diferentes modelos. La clase ModelForm permite crear una clase formulario a partir de un modelo en Django y la clase ModelAdmin es la representación de un modelo en la interfaz de administración. Extendiendo ambas clases para el modelo de interés, se puede personalizar las páginas relacionadas con la edición de cualquier modelo desde la administración.

GeoDjango

GeoDjango [48] es un paquete incluido para Django que añade funcionalidades específicas que permiten almacenar y manipular datos geográficos. GeoDjango se esfuerza por simplificar la creación de aplicaciones web geográficas, como los servicios basados en la ubicación. Sus características incluyen:

- Campos de modelos de Django para geometrías OGC y datos ráster.
- Extensiones al ORM de Django para consultar y manipular datos espaciales.

- Interfaces de Python de alto nivel acopladas de forma flexible para la geometría de Sistemas de Información Geográficos y operaciones de ráster y manipulación de datos en diferentes formatos.
- Da soporte para la edición de campos de geometría de un modelo desde la administración.

```python
from django.contrib.gis.db import models

class Station(models.Model):
    name = models.CharField(max_length=50)
    location = models.PointField(srid=4326,  blank=True, null=True)

    ...

class Campaign(models.Model):
    name = models.CharField(max_length=100, default='-')
    project = models.CharField(max_length=100)
    institution = models.CharField(max_length=100)
    ship = models.CharField(max_length=100)
    chiefCampaign = models.CharField(max_length=100)
    startingEndingLocation = models.MultiPointField(blank=False, null = False)
    startingDate = models.DateTimeField()
    endingDate = models.DateTimeField()
    personal =  models.TextField()
    extentArea = models.PolygonField()
    stations = models.ManyToManyField(
        Station,
        through='Measurement'
    )

    ...
```

Figura 16: Ejemplo de modelos en Django con campos geográficos asociados

En la Figura 16 se muestra la definición de varios campos geográficos de diferentes tipos. En el modelo Station se define un campo *location* de tipo *PointField*. En el caso de la campaña se define el campo *startingEndingLocation* como *MultiPointField*, necesario para almacenar el puerto de partida y de destino, en caso de utilizarse un barco en la campaña. Finalmente en el *extentArea* se declara un campo *PolygonField*, que contendrá las coordenadas del rectángulo que delimita la extensión de la campaña.

Django-Leaflet

Este paquete proporciona una manera de incorporar Leaflet muy fácilmente en un proyecto de Django [49]. En el caso específico de las páginas de edición y creación de un modelo, si dicho modelo posee un campo geográfico entonces Django-Leaflet se encarga de insertar el mapa y los componentes adecuados para editar los campos tipo geometrías del modelo. El widget para editar geometrías se llama Leaflet Draw [50] y se puede apreciar en la barra de herramientas a la izquierda de la Figura 17. Leaflet Draw incorpora más funcionalidades que el widget que incorpora GeoDjango para edición de geometrías, una de las razones que motivó la elección de Django-Leaflet. Además, el módulo es altamente configurable a través de settings.py, el fichero de configuración de Django.

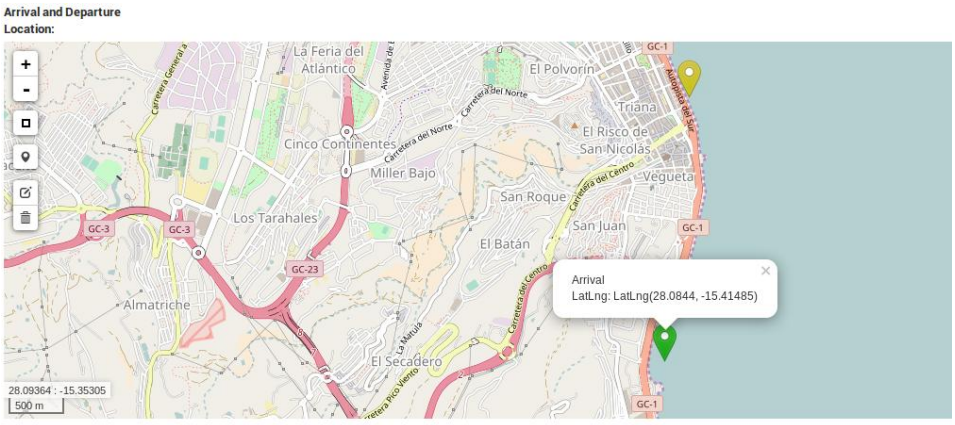

Figura 17: Barra de herramientas que permite editar puntos

En el caso la edición de las campañas, fue necesario modificar el comportamiento del widget de Leaflet para permitir insertar como máximo dos marcadores que representan los puertos de partida y llegada. Ver Figura 17. Deben poder distinguirse entre sí por colores. Se utilizó los eventos JavaScript que tiene definido el widget para realizar la personalización.

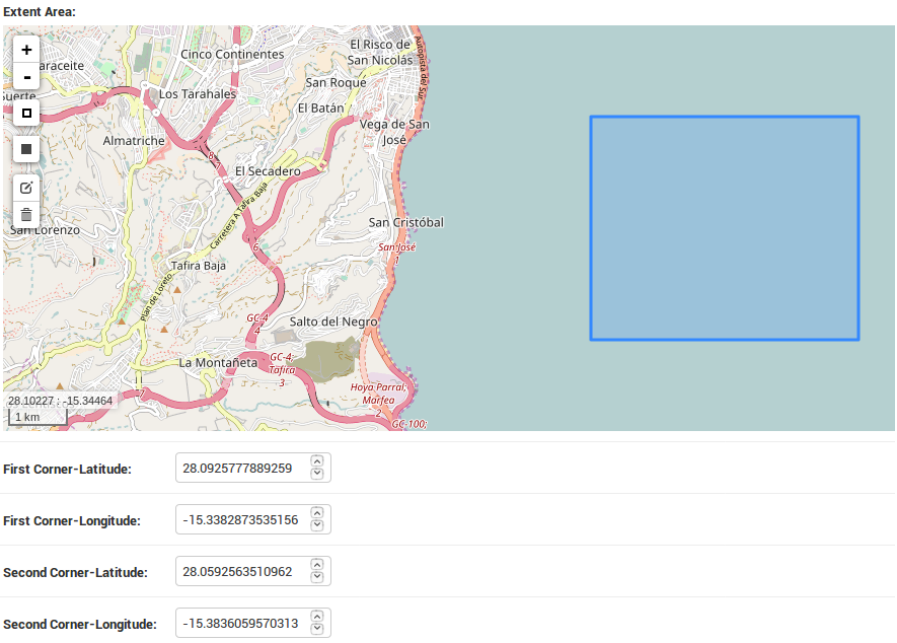

Figura 18: Barra de herramientas para la edición de rectángulos

También en la edición de las campañas, específicamente en el campo que define su extensión, se utilizó de nuevo Leaflet Draw. En este caso el comportamiento por defecto del widget es permitir que se dibuje un rectángulo pero una vez salvados los cambios en la base de datos, al tratar de editar el rectángulo en el mapa, el widget lo trata como un polígono. Esto trae como desventaja que al editar la capa para mover el polígono de lugar si se desea, pues no está permitido. Solamente mover los puntos que lo definen, lo que rápidamente deforma el rectángulo. Para solucionar esto se modifica el comportamiento del widget utilizando los eventos que trae definido. El polígono se convierte a rectángulo al cargarse el mapa. También

se agregó la posibilidad de insertar las coordenadas de las esquinas noreste y suroeste del rectángulo, para tener un control más preciso. En todos los mapas de este sistema se utiliza el plugin Leaflet.MousePosition[22], que muestra en la esquina inferior izquierda del mapa las coordenadas latitud y longitud del ratón al moverse sobre el mismo. Ver Figura 18.

Django API REST

Django REST Framework [51] es un paquete de Django que permite construir APIs web. Incluye una interfaz administrativa desde la cual es posible realizar pruebas sobre la API definida mediante los distintos tipos de peticiones HTTP. También incluye políticas de autenticación y autorización para regular el acceso a los recursos de la API. El siguiente código muestra la configuración global de los permisos de la API dentro del fichero de configuración global de Django, settings.py. En este caso se permite el acceso anónimo de solo lectura.

```
REST_FRAMEWORK = {
  'DEFAULT_PERMISSION_CLASSES': [
     'rest_framework.permissions.DjangoModelPermissionsOrAnonReadOnly'
  ]
}
```

También puede personalizarse estos permisos al nivel de las vistas definidas en la API REST. La Figura 15, muestra una de estas vistas, MeasurementStationViewSet. En la línea 6 se declara que hereda de ReadOnlyModelViewSet, de esta manera se especifica que realizará operaciones de solo lectura. En cuanto a la serialización de los objetos, admite fuentes de datos ORM y no ORM, o sea, que pueden o no estar asociados a modelos en Django. Este paquete de Django cuenta con amplia documentación y gran apoyo de la comunidad.

Los serializadores permiten que los datos complejos como los resultados de consultas y las instancias de modelo se conviertan en JSON u otros tipos de contenido. Los serializadores también proporcionan deserialización, permitiendo que los datos analizados se conviertan de nuevo en tipos complejos, después de validar primero los datos entrantes. En la Figura 19 se implementa el serializador utilizado en la búsqueda de las estaciones que participaron en una campaña. En la clase Meta de la línea 12 se declara el modelo sobre el cual devolverá los resultados. Los campos a devolver se declaran en la línea 14. Este serializador se utiliza de conjunto con la vista definida en la Figura 15, en la línea 10.

```
1
2
3   from metadata.models import Parameter
4   from station.models import Station, Campaign, Measurement
5   from rest_framework import serializers
6
7   class MeasurementStationSerializer(serializers.ModelSerializer):
8       stationId = serializers.ReadOnlyField()
9       stationName = serializers.ReadOnlyField()
10      dataSetId = serializers.ReadOnlyField()
11
12      class Meta:
13          model = Measurement
14          fields = ('stationId', 'time', 'stationName', 'dataSetId', 'dimension')
```

Figura 19: Ejemplo de clase que define un serializador

[22] https://github.com/ardhi/Leaflet.MousePosition

En la Figura 20 se observa el uso de la interfaz de administración del framework Django REST. En este caso se presentan los resultados de la prueba con la búsqueda de las estaciones que participan en una campaña para un parámetro dado. El formato de los campos del resultado son los especificados en el serializador MeasurementStationSerializer.

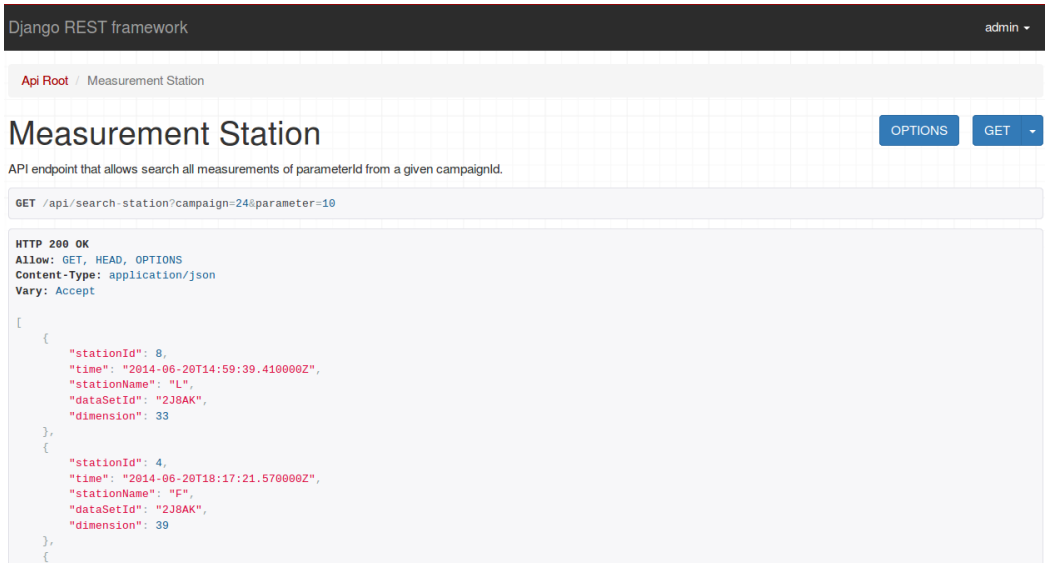

Figura 20: Herramienta web de Django REST Framework

Los recursos que expone la API REST son:

Tabla 7: Recursos que expone la API REST implementada en Django

http://<dominio>/api/parameters/	*Listado de los parámetros*
http://<dominio>/api/campaigns/	*Listado de las campañas*
http://<dominio>/api/campaigns/<id>	*Devuelve la campaña con el identificador especificado*
http://<dominio>/api/stations/	*Listado de estaciones*
http://<dominio>/api/stations/<id>	*Devuelve la estación con el id especificado*
http://<dominio>/api/search-station?campaign=<campaignid>&parameter=<parameterid>	*Busca todas las estaciones que hayan medido el parámetro especificado y que hayan participado en la campaña dada.*
http://<dominio>/api/search-campaign?station=<stationid>&parameter=<parameterid>	*Busca todas las campañas en que se haya medido el parámetro especificado y en que haya participado la estación especificada.*

Django-Cors-Headers

Este paquete de Django permite que la API REST desarrollada pueda ser consumida desde dominios distintos al de la aplicación Django. Permite definir desde el fichero de configuración global de Django, una lista de hosts que tienen permitido realizar peticiones CORS. Con la directiva *CORS_ORIGIN_WHITELIST* se declara el dominio del host que tiene permitido acceder a la API. Permite igualmente especificar los métodos HTTP autorizados para este tipo de peticiones [52]. Solo se permiten peticiones GET y OPTIONS en el caso de este proyecto. La

solicitud con el método OPTIONS es una forma de solicitar un permiso previo a la realización de la solicitud GET [53]. Las siguientes líneas muestran el código de configuración:

Tabla 8: Código de configuración de Django-Cors-Header

```
CORS_ORIGIN_WHITELIST = (
   'www.plocan-perfiles.com',
)

CORS_ALLOW_METHODS = (
   'GET',
   'OPTIONS'
)
```

JSON Editor

JSON Editor es una librería JavaScript para ver, editar, formatear y validar JSON [54]. Tiene varios modos, como un editor de árbol, un editor de código y un editor de texto sin formato. Es utilizada dentro de la administración de Django para editar los campos JSON de las plantillas y de las variables. Para ello fue necesario redefinir la clase ModelForm asociada a ambos modelos y además crear una clase widget personalizada para este tipo de campo.

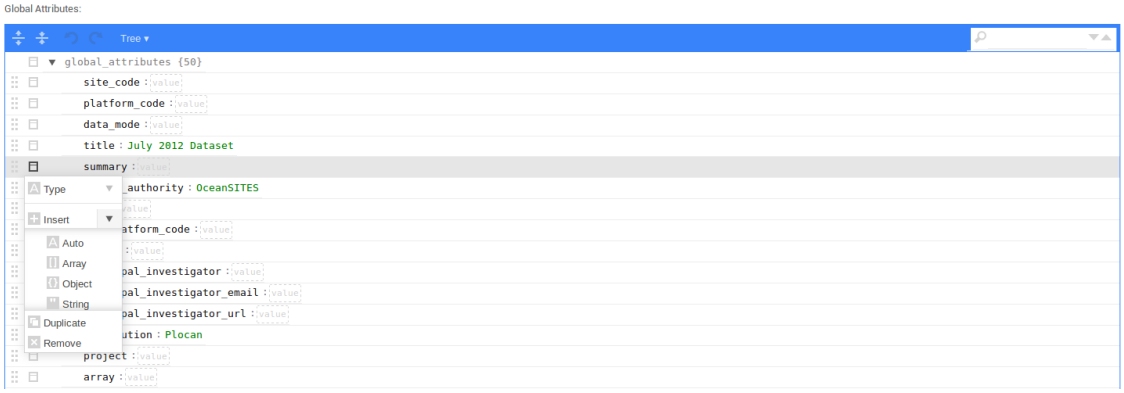

Figura 21: Herramienta que permite editar campos JSON

8.4.3.3. ERDDAP

ERDDAP se utiliza en este proyecto para crear un servicio que nos permita distribuir los datos almacenados localmente en formato netCDF. En este apartado se describen los pasos necesarios para su implantación y la posterior configuración de los conjuntos de datos o datasets. Aunque no forma parte del alcance de este proyecto explicar el uso que se le dará a todas las funcionalidades que incorpora la interfaz de administración de ERDDAP, en principio se puede decir que su uso se limitará a un usuario administrador.

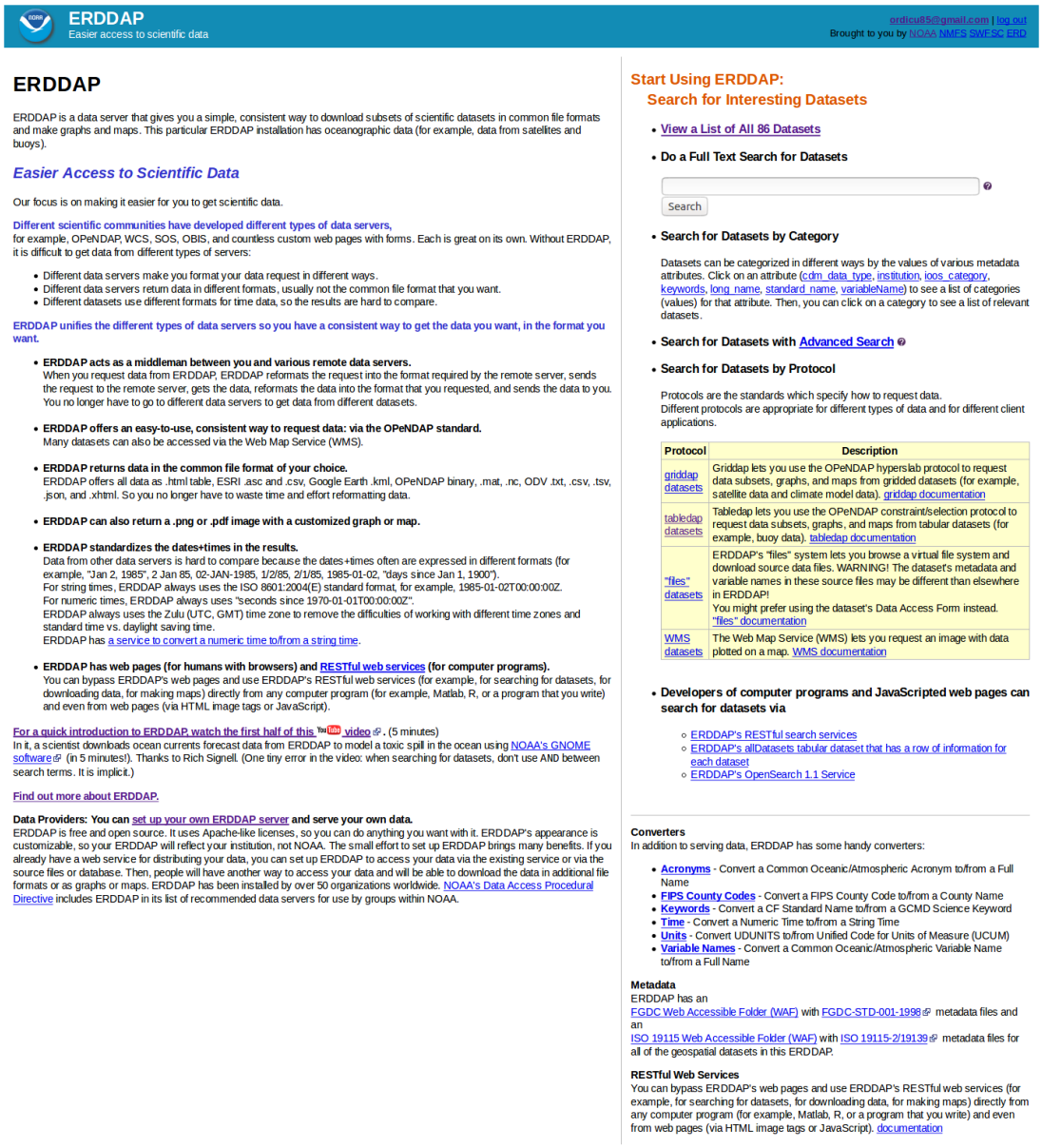

Figura 22: Página de inicio de ERDDAP

Instalación y configuración

Las instrucciones para instalar ERDDAP en nuestro propio servidor se pueden encontrar en [55]. Es necesario instalar primero Java y a continuación Tomcat. En este último caso se debe crear un usuario Tomcat y asignar una serie de permisos sobre sus directorios de instalación así como configurar variables de entornos de Tomcat. Posteriormente se debe descargar una serie de ficheros para configurar ERDDAP. A través del fichero setup.xml se le especifican algunos parámetros como:

Ruta absoluta del directorio bigParentDirectory: El usuario que ejecuta Tomcat debe tener privilegios de lectura/escritura para este directorio. Los subdirectorios que serán creados por ERDDAP dentro de bigParentDirectory son:

- DatasetInfo (algunos datasets lo utilizan para almacenar su información y acelerar su recarga en el futuro).
- Flag: donde se puede poner un archivo con el identificador de un dataset (**datasetID)** para forzar la recarga del mismo
- Caché: ERDDAP lo utiliza para mantener los archivos de datos en caché.
- El archivo de registro ERDDAP (log.txt) se colocará en este directorio: es útil en caso de que algunos datasets no se carguen en ERDDAP o este no se ejecute.

Dirección Base: Declara la dirección pública de ERDDAP. En el proyecto se configuró para utilizar el protocolo https, siendo necesario configurar Tomcat para dar soporte al uso de SSL a través de la creación de una llave pública y la generación de un certificado auto firmado dentro del servidor Tomcat.

Autenticación: Se utiliza en caso que se desee restringir el acceso a un dataset que se declare privado. En este proyecto se configuró el método de autenticación google, que requiere crearse una cuenta en el sitio de Google para desarrolladores y obtener un identificador de cliente de Google.

Una vez que se ha instalado y configurado Java y Tomcat y se terminó de configurar ERDDAP, se procede a realizar el despliegue de la aplicación ERDDAP dentro del directorio de las aplicaciones web de Tomcat.

Configuración de los datasets

Después de instalar y configurar ERDDAP se debe editar el archivo datasets.xml, localizado dentro de los archivos de configuración de ERDDAP, para describir los datasets que se servirán al público. En la Figura 22 se puede ver que se encuentran cargados 86 datasets, 83 que trae por defecto y 3 que son de creación propia en este proyecto.

Tabla 9: Estructura básica del fichero de configuración de datasets en ERDDAP

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<erddapDatasets>
    <convertToPublicSourceUrl /> <!-- 0 o más -->
    <requestBlacklist>...</requestBlacklist> <!-- 0 o 1 -->
    <subscriptionEmailBlacklist>...</subscriptionEmailBlacklist> <!—0 o 1 -->
    <user username="..." password="..." roles="..." /> <!-- 0 o más -->
    <dataset>...</dataset> <!-- 1 o más-->
</erddapDatasets>
```

El código anterior muestra la estructura básica del fichero datasets.xml [56]. Este fichero contiene la descripción que le dice a ERDDAP cómo debe ser accedido cada dataset. Antes de utilizar este fichero se debe establecer el tipo del dataset que mejor representa los datos tipo perfil vertical de las estaciones de PLOCAN. ERDDAP trae dos tipos de dataset:

- EDDGrid, se utiliza para las estructuras de datos en forma de rejilla como datos provenientes de satélites y de modelos.
- EDDTable, utilizado para datos provenientes de boyas estáticas en el mar, estaciones y datos de trayectorias. Este último tipo representa datos tabulares similares a los almacenados en las tablas de una base de datos con filas y columnas. Ver Figura 23.

Back

profile	time	latitude	longitude	temperature	depth	conductivity	salinity	o2_concentration	oxygen	ph	chlorophyll	turbidity
	UTC	degrees_north	degrees_east	K	m	S m-1	PSU	percent	uMol	1	kg m-3	1
1	2013-04-24T01:17:42Z	28.058779	-15.405903	19.612	3.5			79.2	7.24	8.465	0.1	0.1
1	2013-04-24T01:17:42Z	28.058779	-15.405903	20.716	10.1	50.92	36.847	77.3	5.57	8.137	0.6	0.2
1	2013-04-24T01:17:42Z	28.058779	-15.405903	20.723	20.6	50.948	36.864	78.0	5.62	8.152	0.6	0.8
1	2013-04-24T01:17:42Z	28.058779	-15.405903	20.728	30.2	50.947	36.858	78.5	5.65	8.158	0.7	0.2
1	2013-04-24T01:17:42Z	28.058779	-15.405903	20.705	40.3	50.924	36.859	79.2	5.7	8.195	0.8	0.2
1	2013-04-24T01:17:42Z	28.058779	-15.405903	20.713	50.4	50.932	36.859	79.3	5.71	8.192	0.7	0.1
1	2013-04-24T01:17:42Z	28.058779	-15.405903	20.713	60.1	50.937	36.862	79.8	5.75	8.191	0.7	0.1
1	2013-04-24T01:17:42Z	28.058779	-15.405903	20.719	70.6	50.955	36.871	80.1	5.77	8.19	0.7	0.1
1	2013-04-24T01:17:42Z	28.058779	-15.405903	20.731	80.6	50.961	36.866	80.4	5.79	8.189	0.7	0.1
1	2013-04-24T01:17:42Z	28.058779	-15.405903	20.732	90.4	50.965	36.867	80.9	5.82	8.189	0.7	0.1
1	2013-04-24T01:17:42Z	28.058779	-15.405903	20.719	100.5	50.941	36.859	81.3	5.86	8.188	0.8	0.2
1	2013-04-24T01:17:42Z	28.058779	-15.405903	20.672	110.6	50.898	36.864	81.6	5.88	8.189	0.7	0.1
1	2013-04-24T01:17:42Z	28.058779	-15.405903	20.51	120.6	50.721	36.86	81.9	5.92	8.189	0.7	0.1
1	2013-04-24T01:17:42Z	28.058779	-15.405903	20.331	130.2	50.588	36.907	82.4	5.98	8.19	0.7	0.2
1	2013-04-24T01:17:42Z	28.058779	-15.405903	20.255	140.4	50.438	36.849	82.6	6.0	8.19	0.8	0.7
1	2013-04-24T01:17:42Z	28.058779	-15.405903	20.227	150.4	50.434	36.871	82.9	6.02	8.189	0.8	0.2

Figura 23: Listado en forma de tabla del contenido de un dataset

El tipo de dataset utilizado en este proyecto sin lugar a dudas es EDDTable, debido a la naturaleza tabular de los datos tipo perfil vertical. En la web de ERDDAP se refiere que trabajar con el fichero datasets.xml no es una tarea trivial debido a la cantidad de restricciones que se deben cumplir en cuanto a la estructura del XML, codificación de los caracteres del fichero, tratamiento de variables especiales como la longitud, latitud, profundidad y tiempo, errores de sintaxis en XML, entre otras.

Una vez seleccionado EDDTable como el tipo de dataset, se debe especificar entonces el tipo de EDDTable de interés pues existen múltiples opciones [56]:

- EDDTableFromAsciiFiles
- EDDTableFromAsciiService
- EDDTableFromAsciiServiceNOS
- EDDTableFromAwsXMLFiles
- EDDTableFromCassandra
- EDDTableFromColumnarAsciiFiles
- EDDTableFromDapSequence
- EDDTableFromDatabase
- EDDTableFromEDDGrid
- EDDTableFromErddap
- EDDTableFromFileNames
- EDDTableFromFiles
- EDDTableFromHyraxFiles
- EDDTableFromMultidimNcFiles

- EDDTableFromNcFiles
- EDDTableFromNcCFFiles
- EDDTableFromNcCSVFiles
- EDDTableFromNOS
- EDDTableFromOBIS
- EDDTableFromSOS
- EDDTableFromThreddsFiles
- EDDTableFromWFSFiles
- EDDTableAggregateRows
- EDDTableCopy

Después de analizar las opciones se decide escoger el tipo EDDTableFromNcCFFiles. Según su descripción es capaz de agregar datos desde ficheros netCDF, versión 3 o 4, que utilicen uno de los formatos de archivo especificados en la DSG de la convención CF.

El siguiente paso es generar el código XML que representa al dataset de interés. En un principio no se recomienda crearlo desde cero en un editor de código pues tomaría mucho tiempo y requiere tener conocimiento extenso de cómo ERDDAP funciona.

ERDDAP proporciona dos programas de línea de comandos GenerateDatasetsXML y DasDds, que son herramientas para ayudar a crear el XML para cada conjunto de datos que se desee publicar. Son scripts de shell de Linux/Unix con la extensión .sh. Cuando se ejecuta cada programa, hace preguntas sobre la ruta de los ficheros netCDF a importar, el título del dataset entre muchas otras. Una vez completado el proceso y si no hubo errores, el programa imprime por pantalla el código XML que describe los metadatos de los ficheros netCDF de la ruta especificada. Estos programas son capaces de generar un primer borrador del XML y ERDDAP recomienda trabajar entonces a partir de ese borrador creado y no desde cero.

Durante la realización de este trabajo se utilizó fundamentalmente la herramienta GenerateDatasetsXML. Una vez generado el XML se procede a copiarlo dentro del fichero datasets.xml. Debido a que el código XML generado no es perfecto algunos errores se producen por lo que se continúa perfeccionando manualmente hasta que todo funciona bien. La retroalimentación de los errores que se van generando es gracias al fichero de log de ERDDAP, con información detallada del error producido y del dataset que no se pudo cargar.

Una vez que ya se tiene una versión estable del XML que representa un dataset, no es necesario utilizar de nuevo la herramienta GenerateDatasetsXML, pues para subsiguientes datasets del mismo tipo, solo es necesario copiar el XML que funciona, cambiar el identificador del dataset, modificar la ruta de los ficheros netCDF a los que apunta y agregar y/o eliminar variables según sea el caso.

Esta parte del proceso dentro del desarrollo del sistema es la única que se debe hacer manualmente: la generación de los XML correspondientes a cada dataset. En las conclusiones de esta memoria se propondrán mejoras futuras para automatizar esta tarea en las versiones por venir del sistema.

El esqueleto básico del dataset generado es:

Tabla 10: Código XML de configuración de un dataset en ERDDAP

```
<dataset type="EDDTableFromNcCFFiles" datasetID="<datasetID>" active="true">
    <reloadEveryNMinutes>60</reloadEveryNMinutes>
   <updateEveryNMillis>10000</updateEveryNMillis>
   <fileDir><absolute-path> /files/netCDF/<datasetID>/</fileDir>
   <fileNameRegex>.*\.nc</fileNameRegex>
   <recursive>true</recursive>
   <pathRegex>.*</pathRegex>
   <metadataFrom>last</metadataFrom>
   <fileTableInMemory>false</fileTableInMemory>
   <accessibleViaFiles>false</accessibleViaFiles>
   <addAttributes>
      <att name="cdm_profile_variables">profile, time, latitude, longitude</att>
      <att name="CF:featureType">null</att>
      <att name="CF_featureType">Profile</att>
      <att name="Conventions">CF-1.6 , COARDS, ACDD-1.3</att>
      <att name="creator_name">Plocan</att>
      <att name="summary">tt. Plocan data from a local source.</att>
      <att name="title">Plocan Vertical Profiles </att>
     <!—continuan otros atributos->
   </addAttributes>
   <dataVariable>
      <sourceName>PROFILE</sourceName>
      <destinationName>profile</destinationName>
      <dataType>int</dataType>
      <addAttributes>
         <att name="ioos_category">Identifier</att>
         <att name="long_name">Profile</att>
      </addAttributes>
   </dataVariable>
 <!—continúan otras variables-->
</dataset>
```

En este código XML se empieza por definir el tipo del dataset, su identificador, cada cuanto tiempo se actualiza y recarga, la ruta del origen de los datos, si debe buscar recursivamente dentro de la raíz por otros directorios y la extensión de los ficheros de interés. En los atributos se definen características globales a todo el fichero como el tipo de DSG que representa (en este caso es un perfil), título del dataset, institución que lo crea. A continuación de los atributos viene la definición de las variables. Para el ejemplo solo se muestra la variable PROFILE.

8.4.3.4. Aplicación web Angular

En esta sección se describen los patrones de diseño y elementos más relevantes del framework que influyen notablemente en la curva de aprendizaje de Angular y en la posterior implementación de la aplicación. Las dependencias que requiere esta aplicación son básicamente:

- Node v6.x.x [57], necesario para poder utilizar el gestor de paquetes npm.
- npm 3.x.x [58]: gestor de librerías JavaScripts.
- Webpack: librería JavaScript para cargar y empaquetar módulos. Más adelante se describe qué es un módulo.

- Typescript compiler: compilador del lenguaje Typescript

Typescript

Con la nueva versión de Angular se utiliza Typescript para crear aplicaciones aunque es posible seguir haciéndolo utilizando JavaScript. Typescript es un superconjunto de JavaScript que se compila a JavaScript y como Java, permite definir nuevos tipos y clases, desarrollando así una programación web orientada a objetos. La declaración de variables con tipos en lugar de utilizar una variable genérica abre la puerta a herramientas más productivas que las disponibles para JavaScript. TypeScript viene con un analizador de código estático y a medida que se introduce código en el IDE con soporte para este lenguaje como por ejemplo Visual Code, éste muestra una ayuda contextual que sugiere los métodos disponibles en el objeto o tipos del argumento de la función [59].

Debido a que los navegadores actuales solo entienden el código JavaScript, cualquier aplicación desarrollada en Typescript requiere una librería que transforme el código a dicho lenguaje. Su función es analizar la sintaxis que los navegadores no entienden (por ejemplo, clases, 'const', funciones de flecha [60]) y convertirlos en sintaxis que entenderán (funciones, 'var', funciones).

Librerías externas

En el caso que la aplicación Angular escrita en Typescript utilice una librería de terceros escrita en JavaScript, se puede instalar un archivo de definición de tipo (con la extensión .d.ts) que contenga declaraciones de tipo para esta librería. Las declaraciones de tipos de cientos de librerías JavaScript populares están disponibles libremente y se pueden instalar fácilmente con Typings [61], un Administrador de definiciones de Typescript.

Se utiliza el gestor de paquetes npm para instalar todas las dependencias de esta aplicación. Las dependencias externas más importantes del proyecto son:

- Ng-Bootstrap: Es la versión de Angular del framework CSS Bootstrap 4 [62]. Está desarrollada en Typescript. Se utiliza para el maquetado y los estilos del HTML de la aplicación.
- Leaflet: Librería de mapas en su versión JavaScript. Es necesario instalar el archivo de definición correspondiente @types/leaflet [63]. Se emplea en la visualización de mapas interactivos.
- Ng-highcharts: Librería de Angular que permite el uso de highcharts [64]. Genera las gráficas de la aplicación.

Patrón de Diseño Desarrollo basado en Componentes

Angular no es un framework MVC, sino un framework basado en componentes, que no son más que bloques de construcción reutilizables que encapsulan su lógica interna [59]. En Angular, una aplicación es un árbol de componentes. Por ejemplo, la Figura 24 muestra la página de la aplicación como una colección de componentes App, Maps, Search y Charts. El componente padre o raíz es App, mientras el resto son los componentes hijos.

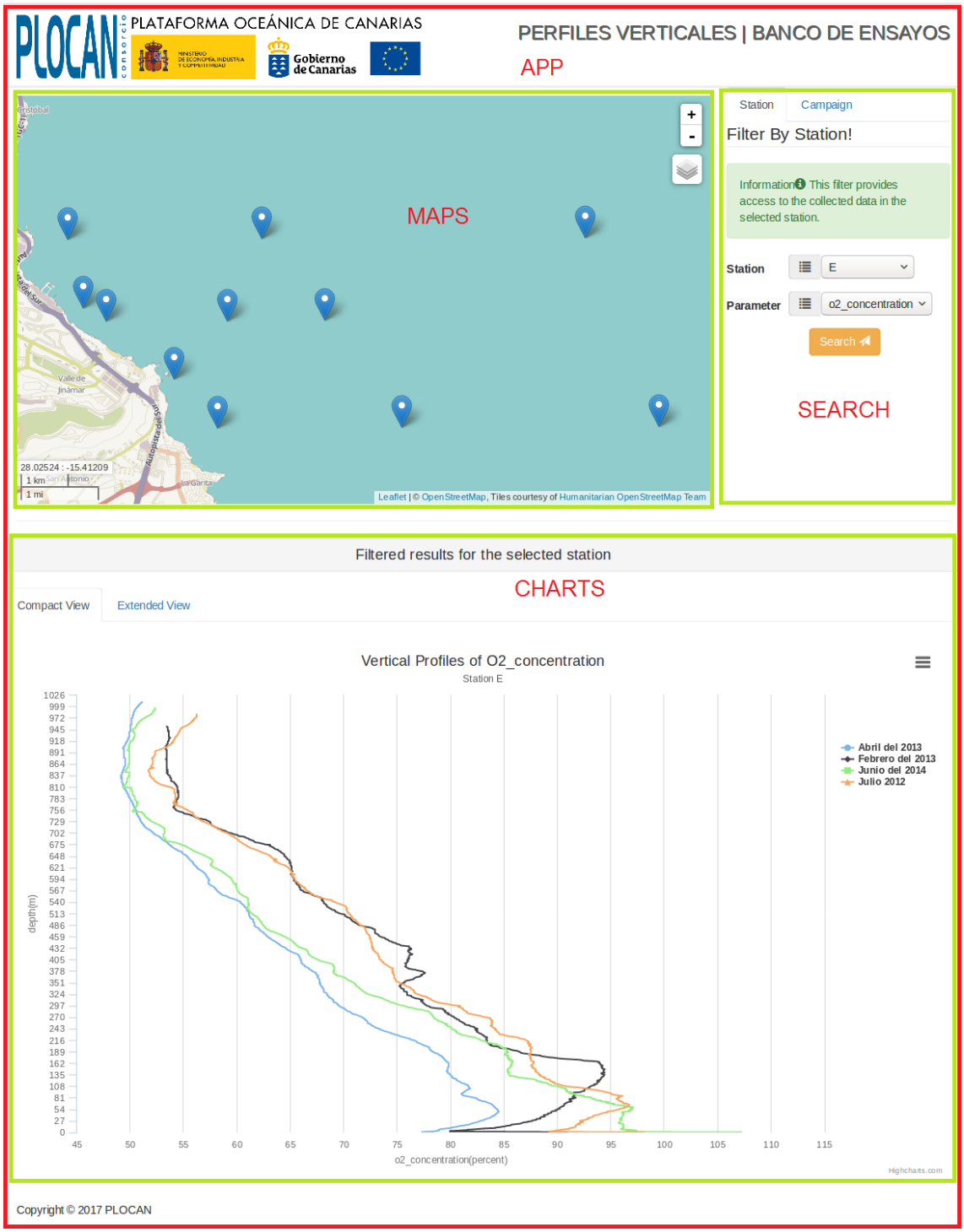

Figura 24: Componentes de la aplicación Angular

En Typescript un componente es una clase anotada con @Component, ver Figura 25.

```
1  import {
2    Component,
3    OnInit,
4    ViewEncapsulation
5  } from '@angular/core';
6
7  /*
8   * App Component
9   */
10 @Component({
11       selector: 'app',
12       encapsulation: ViewEncapsulation.None,
13       styleUrls: [
14         './app.component.scss',
15       ],
16       templateUrl: './app.component.html'
17 })
18
19 export class AppComponent {
20   /**contenido va aqui**/
21 }
```

Figura 25: Ejemplo de definición de componente

La clase AppComponent se anota con @Component en la línea 10. El valor de la propiedad selector de la línea 11 se utilizará como si se hubiera definido una nueva etiqueta HTML definida como <app ></app>. La propiedad templateURL especifica la ubicación del fichero que contiene la plantilla del componente mientras que styleUrls define el fichero de estilos del componente. Dentro de la clase AppComponent va la lógica del componente.

```
1  <html>
2  <head>
3  </head>
4  <body>
5    <app ng-version="4.0.3">
6      <header>
7      <!--aqui va contenido-->
8      </header>
9      <section>
10     <div class="container" id="main">
11         <div class="row">
12             <section class="col-md-9">
13                 <stations-map></stations-map>
14             </section>
15
16             <aside class="sidebar col-md-3">
17                 <search-form></search-form>
18             </aside>
19         </div>
20
21         <section class="row">
22             <charts></charts>
23         </section>
24         <footer>
25             <div class="row">
26                 <div class="col-lg-12">
27                     <p>Copyright © 2017 PLOCAN</p>
28                 </div>
29             </div>
30         </footer>
31     </div>
32     </section>
33   </app>
34 </body>
35 </html>
```

Figura 26: Etiquetas de los componentes dentro de la plantilla HTML

La Figura 26 muestra una versión resumida de los elementos de la página capturados por la extensión para desarrolladores de Google Chrome. Se aprecia que existen 4 nuevas etiquetas HTML representando a los 4 componentes de la aplicación en las líneas 5, 13, 17 y 22 respectivamente. La etiqueta app representa el componente padre y las etiquetas stations-map, search-form y charts representan al resto de los componentes.

En la Figura 27 se muestra una versión muy resumida de la clase ChartsComponent, que representa el componente encargado de mostrar las gráficas de los perfiles verticales. Uno de los atributos de este componente es la variable de tipo arreglo extendedChartList de la línea 16. Dentro de ChartsComponent existen métodos responsables de insertar dentro de extendedChartList la información de todos los objetos que representan a las gráficas. Por otra parte las variables globales (atributos) de un componente siempre están enlazadas a la respectiva plantilla. Cualquier cambio que ocurra en extendedChartList se refleja automáticamente en la plantilla de ChartsComponent. Ver Figura 28.

```
1  import {
2    Component,
3    OnInit,
4    ViewEncapsulation
5  } from '@angular/core';
6
7  @Component({
8
9    selector: 'charts',
10   templateUrl: './charts.component.html',
11   encapsulation: ViewEncapsulation.None,
12   styleUrls: ['charts.component.css']
13 })
14
15 export class ChartsComponent implements OnInit {
16       public extendedChartList:Array<any> = [];
17      /**contenido va aqui**/
18 }
```

Figura 27: Componente ChartsComponent

```
1  <!--aquí va contenido-->
2
3  <div id="extended" class="tab-pane fade">
4      <div class="charts-container" *ngIf='extendedChartList && extendedChartList.length > 0'>
5          <div  *ngFor="let extendedChartData of extendedChartList;trackBy:trackChartById;let last = last;" class="col-md-4">
6                  <div  [ng2-highcharts]="extendedChartData"></div>
7          </div>
8      </div>
9    </div>
```

Figura 28: Plantilla charts.component.html, del componente ChartsComponent

En la plantilla charts.component.html se define un ciclo for para iterar sobre los elementos de extendedChartList en la línea 5 y por cada elemento se llama a la librería externa ng2-highcharts para el renderizado de la gráfica.

Patrón de Diseño Inyección de dependencias

Los componentes pueden utilizar servicios para implementar lógica de acceso a datos o de negocio. Los servicios son sólo clases que Angular instancia y luego inyecta en componentes.

La clase StationService representa el servicio encargado de interactuar con la API REST de Django y recuperar el listado JSON de las estaciones. Ver Figura 29. Esta lógica se declara en el método getStations de la línea 15. Si en la Figura 30 se especifica un argumento de tipo StationService en el constructor de StationsComponent en la línea 19, Angular automáticamente instancia e inyecta el servicio dentro del componente.

```
1  import {Injectable} from "@angular/core";
2  import { Http, Response } from '@angular/http';
3  import { Observable } from 'rxjs/Observable';
4  import 'rxjs/add/operator/map';
5  import 'rxjs/add/operator/catch';
6  import { Station } from '../app.model';
7  import { django_domain } from './utils';
8
9  @Injectable()
10 export class StationService {
11
12   constructor(public http: Http) {
13   }
14
15   getStations() {
16     return this.http.get(django_domain + '/api/stations')
17                     .map((res:Response) => <Station[]>res.json())
18                     .catch(StationService.handleError);
19   }
20 }
```

Figura 29: Clase del servicio StationService

Posteriormente el componente StationsComponent se encarga de llamar al servicio inyectado para almacenar el listado de estaciones en el atributo stationList declarado en la línea 18 de la Figura 30.

```
1  import {
2    Component,
3    OnInit,
4    ViewEncapsulation
5  } from '@angular/core';
6
7  import { StationService } from '../services/station.service';
8  import { Station } from '../app.model';
9
10 @Component({
11   selector: 'stations-map',
12   templateUrl: './stations.component.html',
13   encapsulation: ViewEncapsulation.None,
14   styleUrls: ['stations.component.css']
15 })
16
17 export class StationsComponent implements OnInit {
18   public stationList:Array<Station> = [];
19   constructor(public stationService: StationService) {}
20  /** aqui va contenido **/
21 }
```

Figura 30: Clase que define el componente StationsComponent

Módulos

Los componentes son el componente básico de una aplicación Angular. A su vez se encuentran organizados en módulos, que nos permiten combinar nuestros componentes en grupos reutilizables de funcionalidad que se pueden exportar e importar a lo largo de la aplicación. Cada aplicación Angular debe tener al menos uno de estos módulos: el módulo raíz, que constituye el **punto de entrada de la aplicación**.

En la Figura 31 se muestra el módulo raíz de la aplicación, que importa librerías del núcleo de Angular entre las líneas 1 y 4. Los componentes de la aplicación son importados a continuación. En las líneas 11 y 12 se importan librerías externas para la visualización de las gráficas y del icono que se muestra durante la carga de la página. A continuación se importan los servicios. Finalmente con la anotación @NgModule de la línea 24 se declara el módulo que contiene las declaraciones de las clases que agrupa en la línea 26, los módulos externos que

importa en la línea 32 y los servicios que expone para la inyección de dependencias en la línea 39.

```
1  import { BrowserModule } from '@angular/platform-browser';
2  import { FormsModule } from '@angular/forms';
3  import { HttpModule } from '@angular/http';
4  import { NgModule,   ApplicationRef } from '@angular/core';
5
6  import { AppComponent } from './app.component';
7  import { StationsComponent } from './stations';
8  import { SearchComponent } from './search';
9  import { ChartsComponent } from './charts';
10
11 import {Ng2HighchartsModule} from 'ng2-highcharts';
12 import { LoadersCssModule } from 'angular2-loaders-css';
13
14 import { MapService } from './services/map.service';
15 import { StationService } from './services/station.service';
16 import { ParameterService } from './services/parameter.service';
17 import { CampaignService } from './services/campaign.service';
18 import { MessageService } from './services/message.service';
19 import { SearchService } from './services/search.service';
20
21 import '../styles/styles.scss';
22 import '../styles/headings.css';
23
24 @NgModule({
25   bootstrap: [ AppComponent ],
26   declarations: [
27     AppComponent,
28     StationsComponent,
29     SearchComponent,
30     ChartsComponent
31   ],
32   imports: [
33     BrowserModule,
34     FormsModule,
35     HttpModule,
36     Ng2HighchartsModule,
37     LoadersCssModule
38   ],
39   providers: [
40     ENV_PROVIDERS,
41     APP_PROVIDERS,
42     MapService,
43     StationService,
44     ParameterService,
45     CampaignService,
46     MessageService,
47     SearchService,
48   ]
49 })
50
```

Figura 31: Módulo raíz de la aplicación Angular

Carga de módulos

El estándar ES6 define una sintaxis de módulo estándar a JavaScript y una especificación para la carga de módulos. Sin embargo, actualmente no hay navegadores que puedan cargar nativamente módulos ES6 [65]. Es por ello que existen librerías que permiten al navegador importar módulos ES6. En Angular este tipo de librerías se utiliza para la carga de módulos ya sean los definidos en la propia aplicación como el módulo raíz, punto de entrada de la aplicación u otros módulos representando librerías externas. Las dos librerías JavaScripts más populares en el mercado que nos permiten agregar esta funcionalidad son SystemJS [66] y WebPack [67].

Por otra parte, al crear un proyecto en Angular se tiene la posibilidad de escribir todas las clases desde cero o utilizar proyectos conocidos como Angular Seed Projects o Angular Starters. Estos proyectos nos proporcionan un punto de partida para empezar a desarrollar

aplicaciones sin necesidad de perder tiempo en configurar el ecosistema de librerías de Angular. Los más conocidos son Angular-Cli [68] que está basado en SystemJS y AngularClass [69] que utiliza WebPack.

Al analizar el gráfico de tendencias de Google [70], que muestra la Figura 32 para comparar la popularidad de estas dos librerías, se observa la mayor popularidad de WebPack. Incluso al comparar las estrellas en GitHub que tienen estos dos proyectos, de nuevo sale favorecido WebPack que cuenta con casi 29 mil mientras que SystemJS apenas alcanza los 8317.

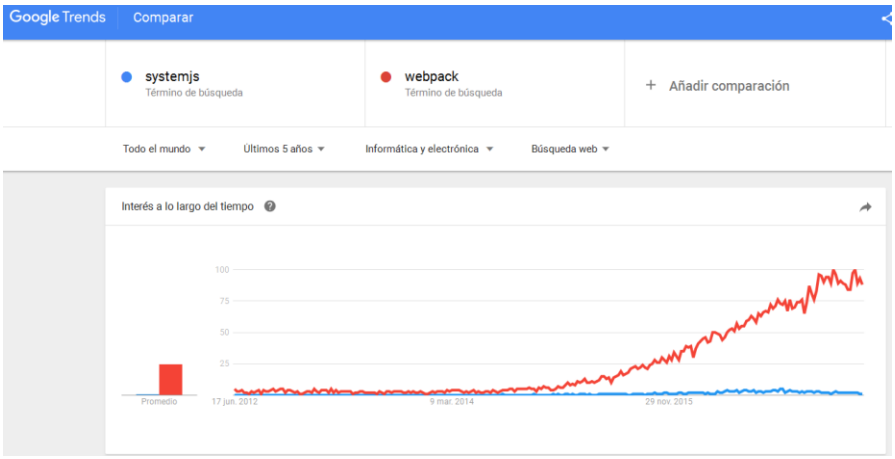

Figura 32: Comparación de Google sobre la tendencia de SystemJS y WebPack [70]

El mismo interés por WebPack también se refleja en las tendencias del conocido sitio Stackoverflow, donde millones de programadores comparten sus conocimientos. Basta con realizar una búsqueda y ver el número de veces que ha sido etiquetada cada librería en la sección etiquetas [71]. Por todo lo anterior la aplicación implementada se basa en el proyecto de AngularClass que utiliza WebPack, pues cuenta con mejor soporte de la comunidad.

Una ventaja adicional de WebPack es que además de cargar módulos permite empaquetarlos, proceso conocido como crear un paquete (bundle)[72]. Un bundle es un fichero que se devuelve como respuesta a una solicitud del navegador. Puede agrupar dentro de sí el contenido de otros ficheros JavaScripts, CSS o de cualquier otro tipo. Al configurar WebPack se le especifica donde se encuentra el punto de entrada de la aplicación. Durante el proceso de compilación de la aplicación recorre el código fuente, buscando instrucciones de importación, crea un gráfico de dependencias y emite uno o más bundles. Esta herramienta integra una serie de plugins y reglas con expresiones regulares que le permiten preprocesar y minificar diferentes archivos que sean JavaScript, TypeScript, CSS o imágenes por ejemplo.

Compilación y despliegue de la aplicación

Para la compilación y despliegue de la aplicación se hace uso de la librería Webpack. En este proyecto Webpack está configurado para ejecutar la aplicación en modo desarrollo y en modo producción. Las diferencias entre ambas radica en la manera en que se realiza la compilación del código Typescript: en el navegador o en el servidor [73].

Una aplicación Angular consiste en componentes y sus plantillas HTML. Antes de que el navegador pueda procesar la aplicación, los componentes y plantillas deben convertirse en JavaScript ejecutable por el compilador de Angular.

Se puede compilar la aplicación en el navegador, en tiempo de ejecución, a medida que la aplicación se carga, utilizando el compilador Just-in-time (JIT). Este es el enfoque que se emplea en modo desarrollo. La compilación JIT tiene una penalización de ejecución. Las plantillas tardan más en procesarse debido a la ejecución de la fase de compilación dentro del navegador. El tamaño de la aplicación es mayor pues incluye el compilador Angular y código de librerías que la aplicación realmente no necesita. Las aplicaciones más grandes tardan más en cargar. Cualquier error que se produzca durante la compilación JIT, es descubierto en tiempo de ejecución.

Por otra parte el compilador Ahead-of-time (AOT) puede detectar errores de plantilla con anticipación y mejorar el rendimiento mediante la compilación en la fase de empaquetado (generación de los bundles) de la aplicación. Con AOT, el navegador descarga una versión pre-compilada de la aplicación. El navegador carga el código ejecutable para que se pueda procesar la aplicación inmediatamente, sin esperar a compilar la aplicación primero. Para desplegar la aplicación en producción se utiliza la compilación AOT.

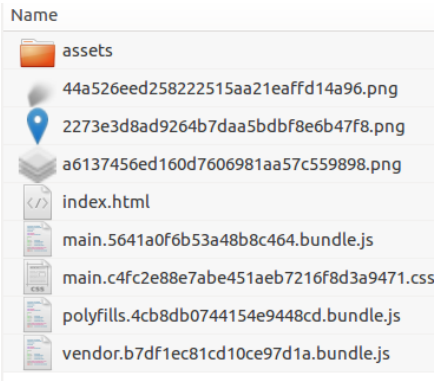

Figura 33: Ficheros de la aplicación Angular empaquetada con WebPack

La Figura 33 muestra los ficheros generados a partir de la fase de empaquetado de WebPack, para la puesta en producción de la aplicación Angular:

- main.5641a0f6b53a48b8c464.bundle.js: contiene el código de la aplicación Angular
- main.c4fc2e88e7abe451aeb7216f8d3a9471.css: contiene el código css
- polyfills.4cb8db0744154e9448cd.bundle.js: Contiene el código que define un nuevo objeto o método en navegadores que no admiten ese objeto o método [74]
- vendor.b7df1ec81cd10ce97d1a.bundle.js: contiene el código de las dependencias de la aplicación

Estos ficheros están listos para ser copiados en el directorio del servidor Nginx desde donde se sirve la aplicación Angular.

Diseño web responsivo, CSS3 y HTML5

La Figura 34 y Figura 35 muestran dos vistas diferentes de la aplicación para tamaños de pantallas de 1500 por 800 píxeles y 768 por 1024 píxeles respectivamente. Se aprecia como la aplicación cumple con el diseño responsivo pues es capaz de cambiar el tamaño, reducir, ampliar o mover el contenido para que se vea bien en las diferentes resoluciones según el dispositivo [75]. Esto se logra gracias a la librería Ng-Bootstrap y el soporte que tiene de las media queries [76]. Las media queries forman parte de la especificación de CSS3 y son un tipo de consulta que condiciona a nuestra hoja de estilos a aplicar propiedades dependiendo de las características del medio donde se muestre. Constituye una excelente manera de entregar una mejor experiencia de usuario. Las media queries permiten que la presentación del contenido se adapte a un rango específico de dispositivos de salida sin tener que cambiar el contenido en sí.

Se utilizaron clases de Bootstrap para mostrar los paneles de alerta en color verde, para los controles del formulario de búsqueda, el uso de pestañas y el efecto de animación al navegar entre ellas, el sombreado del texto y de los bordes de la página. Para estructurar la página en columnas y filas igualmente se emplea el sistema de rejillas de este framework CSS.

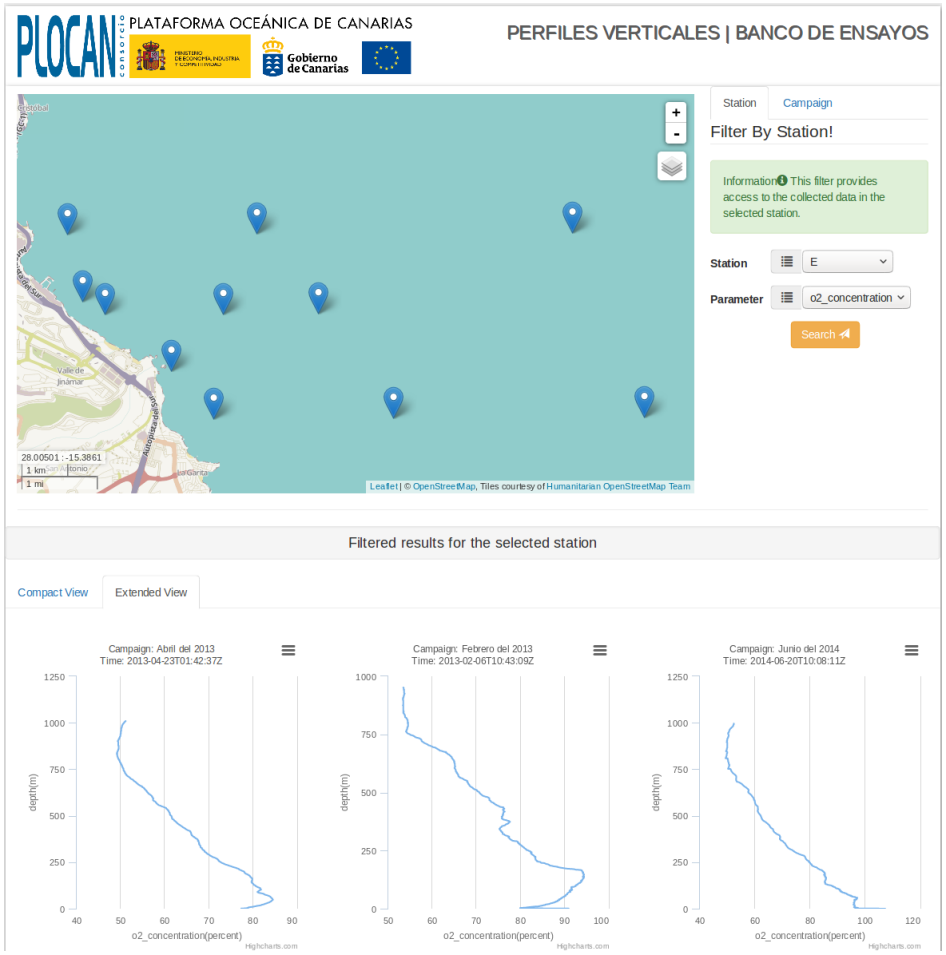

Figura 34: Vista de la aplicación Angular a una resolución de 1500 por 800 píxeles

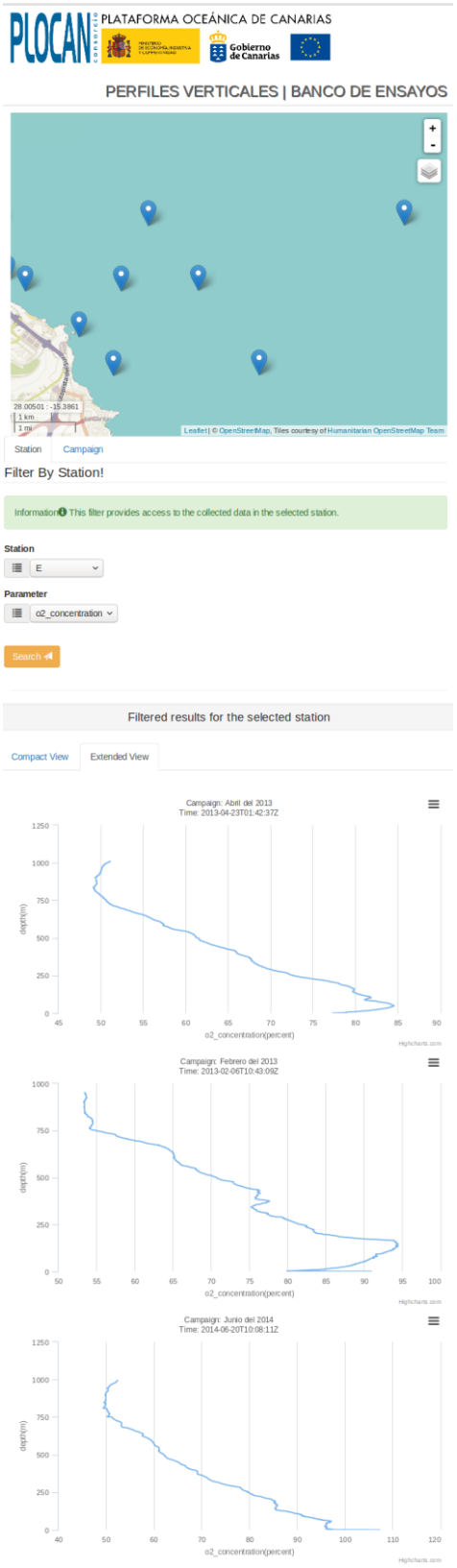

Figura 35: Vista de la aplicación Angular a una resolución de 768 por 1024 píxeles

HTML5

En la versión resumida de la estructura HTML de la página en la Figura 36, se utilizan los nuevos elementos semánticos definidos por el estándar HTML5 [77]: header, section, aside y footer.

```
<html>
<head>
  <meta charset="utf-8">
  <meta http-equiv="x-ua-compatible" content="ie=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Perfiles Verticales|Banco de Ensayos</title>
  <meta name="description" content="Perfiles Verticales|Banco de Ensayos">
  <base href="/">
</head>
<body>
  <app ng-version="4.0.3">
    <header>
    <!--aqui va contenido-->
    </header>
    <section>
    <div class="container" id="main">
        <div class="row">
            <section class="col-md-9">
                <stations-map></stations-map>
            </section>

            <aside class="sidebar col-md-3">
                <search-form></search-form>
            </aside>
        </div>

        <section class="row">
            <charts></charts>
        </section>
        <footer>
            <div class="row">
                <div class="col-lg-12">
                    <p>Copyright © 2017 PLOCAN</p>
                </div>
            </div>
        </footer>
    </div>
    </section>
  </app>
 </body>
</html>
```

Figura 36: Vista código de la estructura HTML5 de la aplicación Angular

También se incluye otro elemento característico de HTML5, el elemento de vista <meta> relacionado con el viewport :

<Meta name = "viewport" content = "anchura = ancho del dispositivo, escala inicial = 1.0">

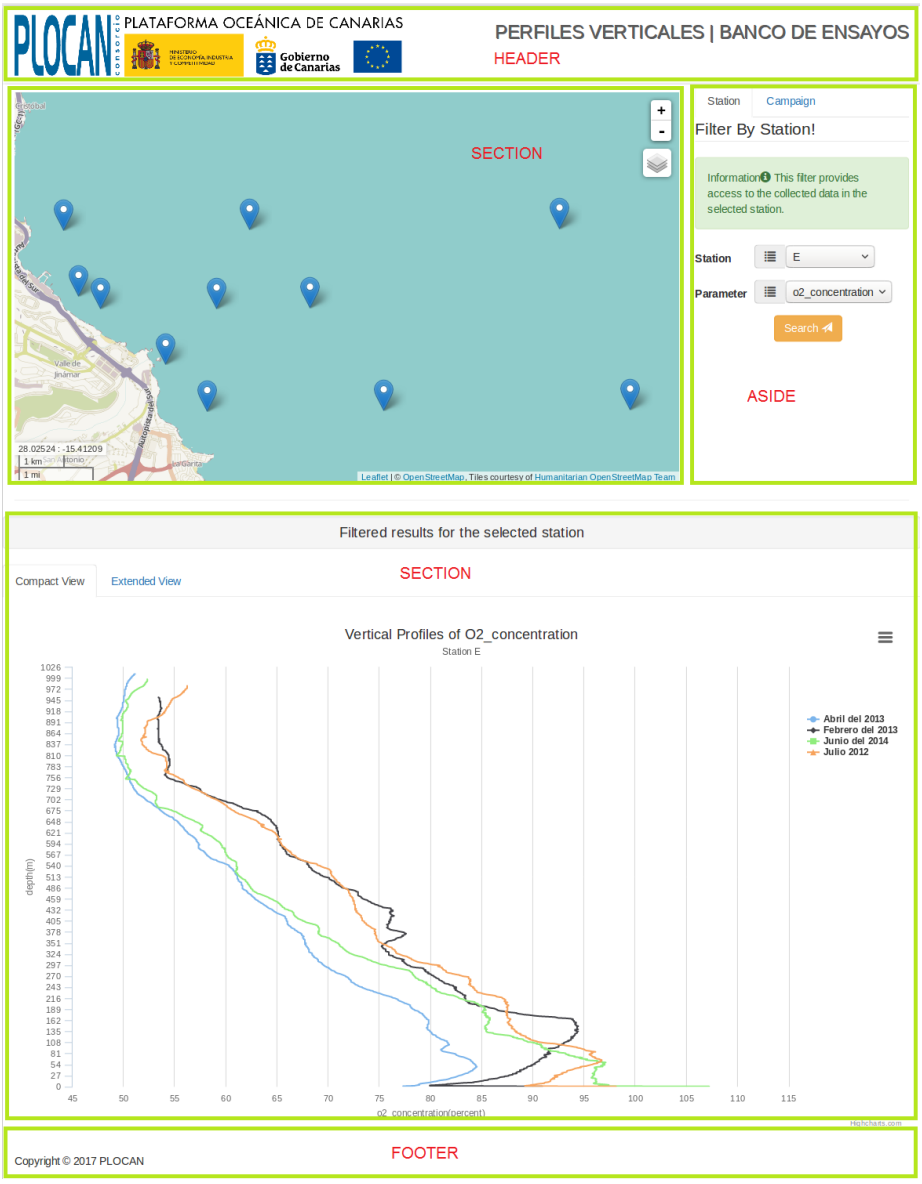

Figura 37: Vista gráfica de los elementos semánticos HTML5 de la aplicación

8.4.3.5. Postgis

Las librerías GEOS, PROJ.4 y GDAL deben ser instaladas antes de la instalación de PostGIS. El módulo psycopg2 es necesario para utilizarlo como adaptador de base de datos cuando se utiliza GeoDjango con PostGIS [78].

8.5. Pruebas y Validación

Las pruebas realizadas para validar el producto incluyen un juego de datos tipo perfil vertical compuesto por 41 ficheros, provenientes de 11 estaciones que participaron en 4 campañas realizadas entre 2012 y 2014. Ver Tabla 11.

En las cuatro campañas se midieron variables como la presión, temperatura, conductividad, salinidad, concentración de oxígeno, moles de oxígeno por unidad de masa, pH, clorofila y turbidez. En cuanto al tamaño de los datos, varía entre 1.2 y 89 kilobytes. En relación a la densidad de observaciones por perfil, fluctúa entre 15 y 1037 observaciones por perfil.

Tabla 11: Resumen del juego de datos utilizado en las pruebas

Campañas	Estaciones	Ficheros	Variables	Min. observaciones/perfil	Max. observaciones/perfil
Julio 2012	8	8	9	15	981
Febrero 2013	11	11	9	19	953
Abril 2013	11	11	9	19	1037
Junio 2014	11	11	9	26	997

Se definieron tres tipos de plantillas y por cada plantilla se creó un dataset en ERDDAP que almacena los perfiles verticales en formato netCDF. Ver Figura 38. Este es un caso extremo de la aplicación y solo es válido con el propósito de realizar las pruebas, pues normalmente al tener los 41 ficheros de perfiles verticales las mismas variables, basta con definir una sola plantilla que los represente y por tanto se almacenan en un único dataset. Con estas tres plantillas se prueba el rendimiento para casos extremos de la aplicación Angular, al ser necesario consultar más de un dataset para recuperar toda la información de una estación.

El hecho de poder ser importados en ERDDAP valida que los ficheros netCDF son creados con la estructura correcta pues de otra manera ERDDAP daría error en su importación.

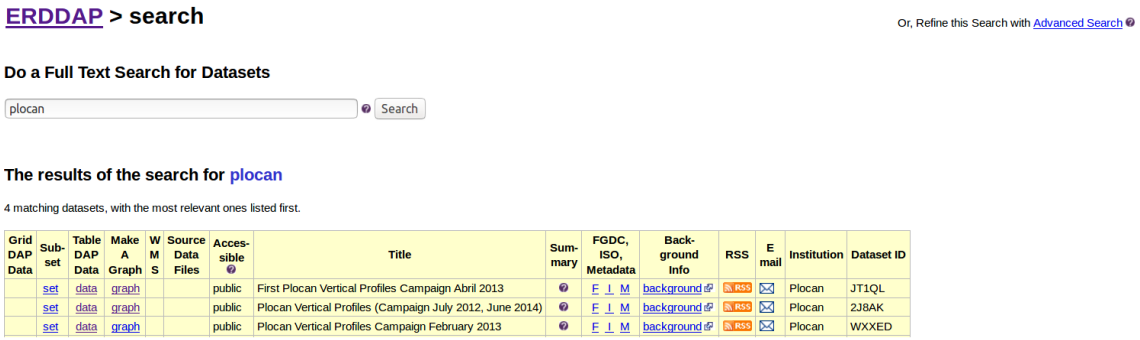

Figura 38: Datasets creados en ERDDAP

También utilizando herramientas de línea de comandos como ncdump [79] se puede validar el contenido de un fichero netCDF. Ver la Figura 39. Esta herramienta realiza un volcado de la información contenida en dichos ficheros. Se instala por defecto como parte de las librerías vistas en la sección 8.4.3.1. Se pueden apreciar metadatos como las dimensiones del netCDF, algunas de sus variables así como el comienzo del área de datos, en particular, los datos de la variable *TEMPERATURE*.

```
jose@ubuntu:~/projects/virtualenvs/geodjango/admin_django/files/netCDF/2J8AK/A/01-06-2014_30-06-2014$ ncdump A_20-06-2014T18\:43\:01.nc
netcdf A_20-06-2014T18\:43\:01 {
dimensions:
        DEPTH = 104 ;
variables:
        double TEMPERATURE(DEPTH) ;
                TEMPERATURE:_FillValue = 0. ;
                TEMPERATURE:_ChunkSizes = 104 ;
                TEMPERATURE:long_name = "Sea water temperature is the in situ temperature of the sea water." ;
                TEMPERATURE:sensor_reference = "reference" ;
                TEMPERATURE:sensor_manufacturer = "Vaisala" ;
                TEMPERATURE:units = "K" ;
                TEMPERATURE:sensor_model = "WXT520" ;
                TEMPERATURE:datacolumn = "Temp" ;
                TEMPERATURE:ancilliary_variables = "TEMP_QC" ;
                TEMPERATURE:standard_name = "sea_water_temperature" ;
        double SALINITY(DEPTH) ;
                SALINITY:_FillValue = 0. ;
                SALINITY:_ChunkSizes = 104 ;
                SALINITY:long_name = "description" ;
                SALINITY:sensor_reference = "reference" ;
                SALINITY:sensor_manufacturer = "Vaisala" ;
                SALINITY:units = "1e-3" ;
                SALINITY:sensor_model = "WXT520" ;
                SALINITY:datacolumn = "Sal" ;
                SALINITY:standard_name = "sea_water_salinity" ;
        double O2PPM(DEPTH) ;
                O2PPM:_FillValue = 0. ;
                O2PPM:_ChunkSizes = 104 ;
                O2PPM:long_name = "dissolve oxygen" ;
                O2PPM:sensor_reference = "reference" ;
                O2PPM:sensor_manufacturer = "Vaisala" ;
                O2PPM:units = "uMol" ;
                O2PPM:sensor_model = "WXT520" ;
                O2PPM:datacolumn = "O2ppm" ;
                O2PPM:standard_name = "moles_of_oxygen_per_unit_mass_in_sea_water" ;
// global attributes:
                :format_version = 1.3 ;
                :geospatial_lat_units = "degree_north" ;
                :geospatial_lon_units = "degree_east" ;
                :id = "A" ;
                :naming_authority = "OceanSITES" ;
                :title = "July 2012 Dataset" ;
                :CF\:featureType = "Profile" ;
                :featureType = "Profile" ;
                :data_type = "OceanSITES profile data" ;
                :institution = "Plocan" ;
                :Conventions = "CF-1.6" ;
                :netcdf_version = 3.5 ;
                :geospatial_vertical_positive = "down" ;
                :geospatial_vertical_units = "meter" ;
                :date_created = "2017-07-10T18:53:54.710405Z" ;
                :geospatial_lat_max = 28.05878f ;
                :geospatial_lat_min = 28.05878f ;
                :geospatial_lon_max = -15.4059f ;
                :geospatial_lon_min = -15.4059f ;
                :time_coverage_end = "2014-06-20T18:43:01.100000Z" ;
                :time_coverage_start = "2014-06-20T18:43:01.100000Z" ;
data:

 TEMPERATURE = 20.844, 21.498, 21.483, 21.505, 21.492, 21.335, 21.228,
    21.183, 21.176, 21.164, 21.153, 21.152, 21.143, 21.141, 21.133, 21.123,
    21.115, 21.113, 21.111, 21.11, 21.11, 21.109, 21.106, 21.101,
    21.1, 21.097, 21.096, 21.094, 21.093, 21.077, 21.036, 21.005, 20.992,
    20.942, 20.708, 20.682, 20.548, 20.534, 20.527, 20.446, 20.304, 20.265,
    20.228, 20.206, 20.109, 19.786, 19.63, 19.545, 19.494, 19.441, 19.383,
    19.373, 19.321, 19.28, 19.22, 19.143, 19.069, 19.015, 18.996, 18.935,
```

Figura 39: Volcado del contenido de un fichero netCDF

ERDDAP permite graficar datos contenidos en un dataset. Seguidamente se muestra una de las pruebas realizadas para comparar las gráficas de perfiles verticales realizadas por la aplicación Angular en la Figura 40 y las mismas gráficas de su contraparte ERDDAP en la Figura 41, Figura 42 y Figura 43. Se comparan los perfiles verticales del parámetro o2_concentration en las estaciones A, C y E en la campaña de Abril del 2013. Los identificadores de esas 3 estaciones en ERDDAP se corresponden con el valor de la variable profile: 1, 2 y 3 respectivamente.

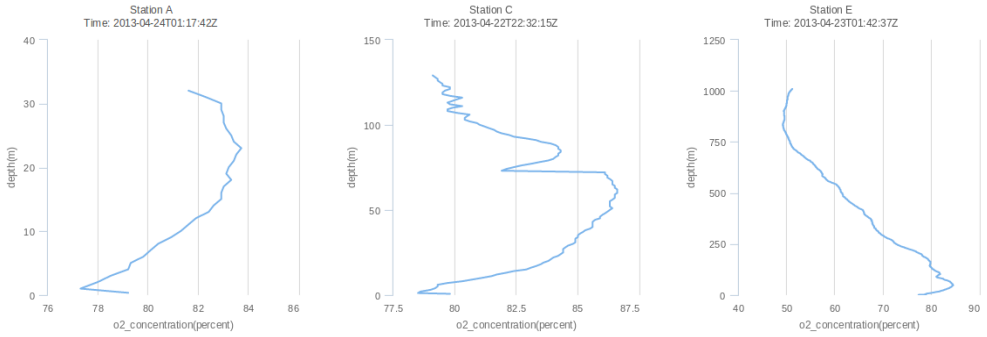

Figura 40: Gráficas de 3 perfiles verticales de o2_concentration de las estaciones A, C y E

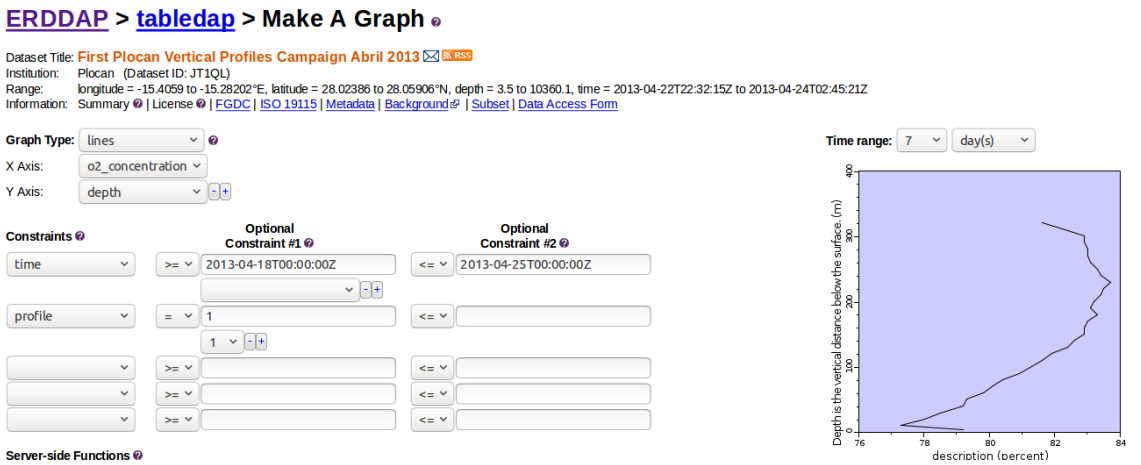

Figura 41: Gráfica de ERDDAP del perfil vertical de o2_concentration para la estación A

Figura 42: Gráfica de ERDDAP del perfil vertical de o2_concentration para la estación C

Figura 43: Gráfica de ERDDAP del perfil vertical de o2_concentration para la estación E

Se observa la coincidencia en los 3 casos comparados.

8.6. Planificación prevista inicialmente y ajuste a la misma.

El factor más importante en el éxito del presente proyecto lo constituye el hecho de que se cumplió el plan general trazado.

Antes de iniciarse el proyecto fue necesario llevar a cabo una etapa de preparación para asimilar una serie de tecnologías nuevas como Python, Django y Angular. Dentro de cada uno de estos lenguajes y frameworks a su vez se estudiaron las soluciones existentes para los requisitos del sistema. Una vez con el conocimiento técnico asimilado, el desarrollo del proyecto fue fluido.

La implantación y puesta a punto del servidor ERDDAP a pesar de presentarse como un factor de riesgo en el proyecto por ser una tecnología completamente nueva, no presentó mayores contratiempos. Sin embargo, la correcta configuración en XML de un dataset necesitó de un mayor tiempo de configuración.

El desarrollo de la aplicación Django fue rápido y eficiente pues este framework se caracteriza por ser muy productivo gracias a las herramientas con que cuenta como su potente capa ORM. Además, una gran comunidad de usuarios le da soporte a cualquier problema que se pueda presentar.

La aplicación Angular en cambio presentó algunos contratiempos en su desarrollo pues es un framework que evoluciona rápidamente y el ecosistema de herramientas que presenta es algo complejo de configurar. El desarrollo del sistema transcurrió en el tiempo planificado y no fue necesario hacer ajustes ni sacrificar funcionalidades en pos de cumplir con el objetivo trazado.

9. Conclusiones y mejoras futuras

Gracias a la reciente incorporación de PLOCAN como socio de la Red Europea de Observación y Datos Marinos, se abre un nuevo camino que le permitirá contribuir activamente a esta red científica. Uno de los objetivos claves de dicha organización es aumentar la productividad en todas las tareas relacionadas con la recogida y gestión de datos marinos y fomentar la innovación. Se impone entonces mejorar el acceso a los datos científicos marinos lo cual constituye una condición sine qua non para compartir la información y afrontar los desafíos vinculados a la gestión de un creciente número de actividades que tienen lugar en el mar.

Para contribuir a dar respuesta a esa necesidad desde PLOCAN, se ha diseñado e implementado una infraestructura web Open Source capaz de dar un servicio que permite la automatización de la estandarización, almacenamiento y visualización de los datos tipo perfil vertical medidos en campañas oceanográficas. En concreto, en PLOCAN se pretende cubrir la necesidad de automatizar la gestión de datos de los perfiles verticales tomados en el área reservada de 32 kilómetros cuadrados conocida como Banco de Ensayos. A nivel tanto canario como mundial, esta herramienta podrá ser utilizada por cualquier tipo de institución que lleve a cabo campañas oceanográficas ya que las necesidades y tipos de soluciones requeridas son muy similares para todas estas instituciones.

El presente sistema contribuye a consolidar a PLOCAN como un importante actor y productor de datos científicos de impacto en la economía y el estudio del medio ambiente marino. Una mayor disponibilidad de estos datos permite establecer políticas más eficientes de protección ambiental y reducir los riesgos asociados con la economía azul, por ejemplo en sectores como el pesquero, las energías renovables y el turismo. Recordar que PLOCAN y el presidente del Cabildo de Gran Canaria firmaron un convenio en Julio del 2016 para potenciar la capacidad de crecimiento de la economía azul y de la biotecnología marina.

Como parte del desarrollo del sistema se ha trabajado en múltiples niveles. Primero se realizó un trabajo investigativo con el estudio del Estado del Arte que permitiera identificar las mejores prácticas a seguir. Posteriormente se configuraron diferentes tipos de servidores: web, de base de datos o de datos científicos. Se han instalado igualmente librerías de datos científicos en Python para el trabajo con netCDF. Finalmente se implementaron dos aplicaciones web en Python y TypeScript, utilizando los frameworks Django y Angular respectivamente. A continuación se destacan los resultados más representativos del trabajo desarrollado:

1. Aplicación web de administración desarrollada en Django:
   1.1. Responsable de gestionar los perfiles verticales y sus metadatos asociados: sensores, variables, nombre estándar de cada variable, estaciones de medida, campañas oceanográficas y mediciones de los perfiles verticales.
   1.2. Capaz de almacenar en formato netCDF los perfiles verticales recopilados en las estaciones.
   1.3. Integra una API REST encargada de servir los metadatos almacenados en la base de datos geoespacial Postgres a la aplicación del frontend del sistema, desarrollada en Angular.

2. Aplicación web desarrollada en Angular:
   2.1. Visualiza las estaciones de observación en un mapa.
   2.2. Realiza la búsqueda de perfiles de todas las estaciones que han participado en una campaña oceanográfica y de todas las campañas en que ha participado una estación.
   2.3. Genera gráficas con los perfiles resultantes de las búsquedas, en dos modos distintos: compacto y extendido. El primer modo permite comparar los perfiles entre sí más fácilmente.
   2.4. Interactúa con la API REST de Django y ERDDAP.
   2.5. Tiene un comportamiento responsivo por lo que se adapta a la pantalla de los distintos dispositivos.
3. Servidores que se integran en el sistema:
   3.1. ERDDAP: servidor de datos científicos, creado por la NOAA, que indexa, almacena y distribuye los ficheros netCDF de los perfiles verticales.
   3.2. Nginx: gestiona las peticiones HTTP de las aplicaciones Django y Angular.
   3.3. Tomcat: gestiona las peticiones HTTP de ERDDAP.
   3.4. Postgres: almacena los metadatos del sistema.
4. Desarrollo con la librería NetCDFGenerator y el formato binario netCDF para generar ficheros compatibles con la convención CF y OceanSites.
5. Se ha aplicado la metodología de la ingeniería de software para documentar el proceso de desarrollo del proyecto pasando por todas sus fases.

Se demostró la viabilidad de la solución practicada por los centros marinos de referencia a nivel mundial: uso de ficheros netCDF para almacenar datos oceanográficos tipo perfiles verticales y almacenamiento en una base de datos geoespacial de sus metadatos. Además, implantar el uso de servidores de datos científicos como ERDDAP para el acceso a dichos datos mediante los servicios web que implementan.

En relación a las líneas futuras de mejoras para la continua evolución del sistema se propone trabajar sobre la generación automática de los datasets de ERDDAP desde la administración de Django. Al estar cada dataset asociado a una plantilla, bien se puede agregar un campo a esta entidad de la base de datos para almacenar el campo XML que define a cada dataset. Desde la interfaz web Django donde se administra la plantilla se agregaría entonces un editor de XML. Una vez salvados los cambios en la plantilla, la aplicación debe ser capaz de validar el XML y actualizar el fichero de ERDDAP donde se definen sus datasets.

Se puede extender fácilmente el diseño de la base de datos para permitir gestionar otros tipos de geometrías discretas como las series temporales. El sistema pudiera ser capaz además, de generar automáticamente las variables asociadas a un fichero de perfiles. Actualmente primero es necesario crear a priori las variables antes de importar dichos ficheros. La ventaja de contar con una funcionalidad de este tipo es que ayuda a reducir posibles errores producto del factor humano a la hora de crear las variables.

También se recomienda incorporar en el sistema de administración un módulo encargado de asegurar la calidad de las mediciones de los datos de los perfiles verticales. Muchas veces los sensores pueden sufrir desperfectos técnicos, por ejemplo debido a las adversas condiciones del tiempo. Luego, el control de la calidad de los perfiles verticales sería un paso previo a su importación en ERDDAP.

10.Referencias

[1] "Marine knowledge 2020 | Asuntos marítimos." [Online]. Available: https://ec.europa.eu/maritimeaffairs/policy/marine_knowledge_2020_es. [Accessed: 19-Jun-2017].
[2] "Noticia: La capacidad de crecimiento de la economía azul en Canarias es del 6 al 12 por ciento, un nicho extraordinario por el que Cabildo y Plocan han firmado un convenio - Sala de Prensa - Cabildo de Gran Canaria." [Online]. Available: http://cabildo.grancanaria.com/-/noticia-la-capacidad-de-crecimiento-de-la-economia-azul-en-canarias-es-del-6-al-12-por-ciento-un-nicho-extraordinario-por-el-que-cabildo-y-plocan-han-. [Accessed: 19-Jun-2017].
[3] "Plocan - PLOCAN becomes a partner of the European Marine Observation and Data Network." [Online]. Available: http://www.plocan.eu/index.php/en/newsplocan/2017/january/1636-plocan-partner-emodnet-en. [Accessed: 19-Jun-2017].
[4] "Welcome to the OGC | OGC." [Online]. Available: http://www.opengeospatial.org/. [Accessed: 20-Jun-2017].
[5] "NetCDF Climate and Forecast (CF) Metadata Conventions." [Online]. Available: http://cfconventions.org/cf-conventions/v1.6.0/cf-conventions.html. [Accessed: 20-Jun-2017].
[6] "NetCDF: FAQ." [Online]. Available: https://www.unidata.ucar.edu/software/netcdf/docs/faq.html#whatisit. [Accessed: 20-Jun-2017].
[7] Finkl, C. W. and Makowski, C., "Remote sensing and modeling: Advances in coastal and marine resources," *Springer*, vol. 9, 2014.
[8] Liu, Y., Qiu, M., and Liu, C., "Big Data in Ocean Observation: Opportunities and Challenges," *Int. Conf. Big Data Comput. Commun.*, 2016.
[9] "Plocan - Descripción." [Online]. Available: http://plocan.eu/index.php/es/sobre-nosotros/quienes-somos/descripcion. [Accessed: 09-Jun-2017].
[10] "Discrete Sampling Geometries (DSG) NetCDF Climate and Forecast (CF) Metadata Conventions." [Online]. Available: http://cfconventions.org/Data/cf-conventions/cf-conventions-1.6/build/cf-conventions.html#discrete-sampling-geometries. [Accessed: 14-Jun-2017].
[11] Hidas, M.G., Proctor, R., Atkins, N. et al., "Information infrastructure for Australia's Integrated Marine Observing System," *Earth Sci. Inform.*, Nov. 2016.
[12] "Moored instrument data processing at BODC." [Online]. Available: https://www.bodc.ac.uk/submit_data/what_do_we_do_with_your_data/data_processing_steps/moored_instrument_data_processing/. [Accessed: 20-Jun-2017].
[13] "Climate and Forecast (CF) netCDF format - the format used by BODC to distribute model data." [Online]. Available: https://www.bodc.ac.uk/resources/delivery_formats/cfnetcdf_format/. [Accessed: 20-Jun-2017].
[14] "About Us - The U.S. Integrated Ocean Observing System (IOOS)." [Online]. Available: https://ioos.noaa.gov/about/about-us/. [Accessed: 20-Jun-2017].
[15] Signell, R., & Snowden, D., "Advances in a Distributed Approach for Ocean Model Data Interoperability.," *J. Mar. Sci. Eng.*, 2014.

[16] "ERDDAP - Home Page." [Online]. Available: https://coastwatch.pfeg.noaa.gov/erddap/index.html. [Accessed: 20-Jun-2017].
[17] "TDS." [Online]. Available: http://www.unidata.ucar.edu/software/thredds/current/tds/TDS.html. [Accessed: 20-Jun-2017].
[18] "Client-Server overview - Learn web development | MDN." [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Client-Server_overview. [Accessed: 20-Jun-2017].
[19] García, L., "EXPLOTACIÓN DE ANALÍTICAS DE APRENDIZAJE EN ENTORNOS EDUCATIVOS ONLINE," Universidad Autónoma de Madrid, 2015.
[20] "Apache vs Nginx: Practical Considerations | DigitalOcean." [Online]. Available: https://www.digitalocean.com/community/tutorials/apache-vs-nginx-practical-considerations. [Accessed: 05-Jul-2017].
[21] "Server-side web frameworks - Learn web development | MDN." [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Web_frameworks. [Accessed: 05-Jul-2017].
[22] M.R. Heda and S. V. Chikurde, "A Review: Geo-Information Technology for Web-Mapping Application," presented at the International Journal of Advanced Research in Computer and Communication Engineering, 2016.
[23] "ECMAScript 2015 Language Specification – ECMA-262 6th Edition." [Online]. Available: http://www.ecma-international.org/ecma-262/6.0/. [Accessed: 15-Jun-2017].
[24] "Objetivos y Competencias del MII | Web de la Escuela de Ingeniería Informática de la ULPGC." [Online]. Available: http://www.eii.ulpgc.es/tb_university_ex/?q=objetivos-y-competencias-del-mii. [Accessed: 19-Jun-2017].
[25] "The Potential Economic Benefits of Coastal Ocean Observing Systems: The Southeast Atlantic Region: Coastal Management: Vol 36, No 2." [Online]. Available: http://www.tandfonline.com/doi/full/10.1080/08920750701861007?scroll=top&needAccess=true. [Accessed: 19-Jun-2017].
[26] "What is Agile Software Development? | Agile Alliance." [Online]. Available: https://www.agilealliance.org/agile101/. [Accessed: 19-Jun-2017].
[27] "Rational Unified Process - Best Practices for Software Development Teams." [Online]. Available: https://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf. [Accessed: 19-Jun-2017].
[28] "The simple artifacts of Analysis and Design." [Online]. Available: https://www.ibm.com/developerworks/rational/library/4871.html. [Accessed: 19-Jun-2017].
[29] "IBM Knowledge Center." [Online]. Available: https://www.ibm.com/support/knowledgecenter/es/SSWSR9_11.0.0/com.ibm.pim.dev.doc/pim_tsk_arc_definingusecases.html. [Accessed: 11-Jun-2017].
[30] "Microsoft Word - IS1 0506 Guiones.doc - unidad6-DOC.pdf." [Online]. Available: http://www.ie.inf.uc3m.es/grupo/docencia/reglada/psi/unidad6-DOC.pdf. [Accessed: 16-Jun-2017].
[31] "Conceptual, Logical and Physical Data Model." [Online]. Available: https://www.visual-paradigm.com/support/documents/vpuserguide/3563/3564/85378_conceptual,l.html. [Accessed: 09-Jun-2017].

[32] "Web Map Service | OGC." [Online]. Available: http://www.opengeospatial.org/standards/wms. [Accessed: 11-Jun-2017].
[33] "Control de acceso HTTP (CORS) - HTTP | MDN." [Online]. Available: https://developer.mozilla.org/es/docs/Web/HTTP/Access_control_CORS. [Accessed: 11-Jun-2017].
[34] "PyCharm :: Features." [Online]. Available: https://www.jetbrains.com/pycharm/features/web_development.html. [Accessed: 12-Jun-2017].
[35] "Visual Studio Code - Code Editing. Redefined." [Online]. Available: https://code.visualstudio.com/. [Accessed: 12-Jun-2017].
[36] "Angular v4 TypeScript Snippets - Visual Studio Marketplace." [Online]. Available: https://marketplace.visualstudio.com/items?itemName=johnpapa.Angular2. [Accessed: 12-Jun-2017].
[37] "Chrome DevTools | Web | Google Developers." [Online]. Available: https://developers.google.com/web/tools/chrome-devtools/?hl=es. [Accessed: 12-Jun-2017].
[38] "GitHub - plocan/netCDF-Generator." [Online]. Available: https://github.com/plocan/netCDF-Generator. [Accessed: 12-Jun-2017].
[39] "netCDF4 1.2.8 : Python Package Index." [Online]. Available: https://pypi.Python.org/pypi/netCDF4. [Accessed: 12-Jun-2017].
[40] "HDF5 for Python." [Online]. Available: http://www.h5py.org/. [Accessed: 12-Jun-2017].
[41] "4.1. Numpy — Computación científica con Python para módulos de evaluación continua en asignaturas de ciencias aplicadas." [Online]. Available: http://pendientedemigracion.ucm.es/info/aocg/Python/modulos_cientificos/numpy/index.html. [Accessed: 12-Jun-2017].
[42] "Python Data Analysis Library — pandas: Python Data Analysis Library." [Online]. Available: http://pandas.pydata.org/. [Accessed: 12-Jun-2017].
[43] "Django Overview." [Online]. Available: https://www.tutorialspoint.com/django/django_overview.htm. [Accessed: 20-Jun-2017].
[44] "Models | Django documentation | Django." [Online]. Available: https://docs.djangoproject.com/en/1.11/topics/db/models/. [Accessed: 12-Jun-2017].
[45] "The Django admin site | Django documentation | Django." [Online]. Available: https://docs.djangoproject.com/en/1.11/ref/contrib/admin/. [Accessed: 12-Jun-2017].
[46] "Creating forms from models | Django documentation | Django." [Online]. Available: https://docs.djangoproject.com/en/1.11/topics/forms/modelforms/. [Accessed: 12-Jun-2017].
[47] "Model Admin Objects | Django documentation | Django." [Online]. Available: https://docs.djangoproject.com/en/1.11/ref/contrib/admin/#modeladmin-objects. [Accessed: 12-Jun-2017].
[48] "GeoDjango | Django documentation | Django." [Online]. Available: https://docs.djangoproject.com/en/1.11/ref/contrib/gis/. [Accessed: 12-Jun-2017].
[49] "GitHub - makinacorpus/django-leaflet: Use Leaflet in your Django projects." [Online]. Available: https://github.com/makinacorpus/django-leaflet. [Accessed: 12-Jun-2017].
[50] "Leaflet Draw Documentation." [Online]. Available: https://leaflet.github.io/Leaflet.draw/docs/leaflet-draw-latest.html. [Accessed: 13-Jun-2017].

[51] "Home - Django REST framework." [Online]. Available: http://www.django-rest-framework.org/. [Accessed: 13-Jun-2017].
[52] "GitHub - ottoyiu/django-cors-headers: Django app for handling the server headers required for Cross-Origin Resource Sharing (CORS)." [Online]. Available: https://github.com/ottoyiu/django-cors-headers. [Accessed: 13-Jun-2017].
[53] "Using CORS - HTML5 Rocks." [Online]. Available: https://www.html5rocks.com/en/tutorials/cors/. [Accessed: 14-Jun-2017].
[54] "GitHub - josdejong/jsoneditor: A web-based tool to view, edit, format, and validate JSON." [Online]. Available: https://github.com/josdejong/jsoneditor. [Accessed: 13-Jun-2017].
[55] "ERDDAP - Set Up Your Own ERDDAP." [Online]. Available: https://coastwatch.pfeg.noaa.gov/erddap/download/setup.html. [Accessed: 14-Jun-2017].
[56] "ERDDAP - Working with the datasets.xml File." [Online]. Available: https://coastwatch.pfeg.noaa.gov/erddap/download/setupDatasetsXml.html. [Accessed: 14-Jun-2017].
[57] "Node.js." [Online]. Available: https://nodejs.org/es/. [Accessed: 15-Jun-2017].
[58] "npm." [Online]. Available: https://www.npmjs.com/. [Accessed: 15-Jun-2017].
[59] "Angular 2 and TypeScript - A High Level Overview." [Online]. Available: https://www.infoq.com/articles/Angular2-TypeScript-High-Level-Overview. [Accessed: 15-Jun-2017].
[60] "Funciones Flecha - JavaScript | MDN." [Online]. Available: https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Funciones/Arrow_functions. [Accessed: 15-Jun-2017].
[61] "typings." [Online]. Available: https://www.npmjs.com/package/typings. [Accessed: 15-Jun-2017].
[62] "GitHub - ng-bootstrap/ng-bootstrap: Angular powered Bootstrap." [Online]. Available: https://github.com/ng-bootstrap/ng-bootstrap. [Accessed: 15-Jun-2017].
[63] "@types/leaflet." [Online]. Available: https://www.npmjs.com/package/@types/leaflet. [Accessed: 15-Jun-2017].
[64] "GitHub - Bigous/ng2-highcharts: Angular2 library to use Highcharts out of the box." [Online]. Available: https://github.com/Bigous/ng2-highcharts. [Accessed: 15-Jun-2017].
[65] "Modular JavaScript: A Beginners Guide to SystemJS & jspm — SitePoint." [Online]. Available: https://www.sitepoint.com/modular-javascript-systemjs-jspm/. [Accessed: 15-Jun-2017].
[66] "GitHub - systemjs/systemjs: Dynamic ES module loader." [Online]. Available: https://github.com/systemjs/systemjs. [Accessed: 15-Jun-2017].
[67] "GitHub - webpack/webpack: A bundler for javascript and friends." [Online]. Available: https://github.com/webpack/webpack. [Accessed: 15-Jun-2017].
[68] "Angular CLI." [Online]. Available: https://cli.angular.io/. [Accessed: 15-Jun-2017].
[69] "AngularClass · GitHub." [Online]. Available: https://github.com/AngularClass. [Accessed: 15-Jun-2017].
[70] "systemjs, webpack - Explorar - Google Trends." [Online]. Available: https://trends.google.es/trends/explore?cat=5&q=systemjs,webpack. [Accessed: 15-Jun-2017].
[71] "Tags - Stack Overflow." [Online]. Available: https://stackoverflow.com/tags. [Accessed: 15-Jun-2017].

[72] "Angular - Webpack: An Introduction." [Online]. Available: https://angular.io/guide/webpack. [Accessed: 15-Jun-2017].
[73] "Angular - Ahead-of-Time Compilation." [Online]. Available: https://angular.io/guide/aot-compiler. [Accessed: 15-Jun-2017].
[74] "Angular - Browser support." [Online]. Available: https://angular.io/guide/browser-support. [Accessed: 15-Jun-2017].
[75] "HTML Responsive Web Design." [Online]. Available: https://www.w3schools.com/html/html_responsive.asp. [Accessed: 15-Jun-2017].
[76] "CSS media queries - CSS | MDN." [Online]. Available: https://developer.mozilla.org/es/docs/CSS/Media_queries. [Accessed: 15-Jun-2017].
[77] "HTML5 Semantic Elements." [Online]. Available: https://www.w3schools.com/html/html5_semantic_elements.asp. [Accessed: 15-Jun-2017].
[78] "Installing PostGIS | Django documentation | Django." [Online]. Available: https://docs.djangoproject.com/en/1.11/ref/contrib/gis/install/postgis/. [Accessed: 14-Jun-2017].
[79] "ncdump - The NetCDF Users' Guide." [Online]. Available: https://www.unidata.ucar.edu/software/netcdf/netcdf-4/newdocs/netcdf/ncdump.html. [Accessed: 15-Jun-2017].

11. Anexos

A. Descripción detallada de los casos de uso y Realización de la interfaz

Se tiene en cuenta que existen casos de uso principales y secundarios que plantean una secuencia de casos de uso necesaria. Es decir, es posible que se deban realizar uno o varios casos de uso correctamente para que se inicie otro caso de uso. Para ello se utiliza el campo Condiciones previas, de cada caso de uso.

Casos de uso de la Administración desarrollada en Django

Esta sección presenta los casos de uso disponibles a los usuarios con el rol de administradores del sistema. En la Figura 44 se muestra la página de Inicio de la Administración. Existen dos secciones fundamentales. La sección **Metadata** y la sección **Station**. En las siguientes páginas se describen los casos de uso asociados.

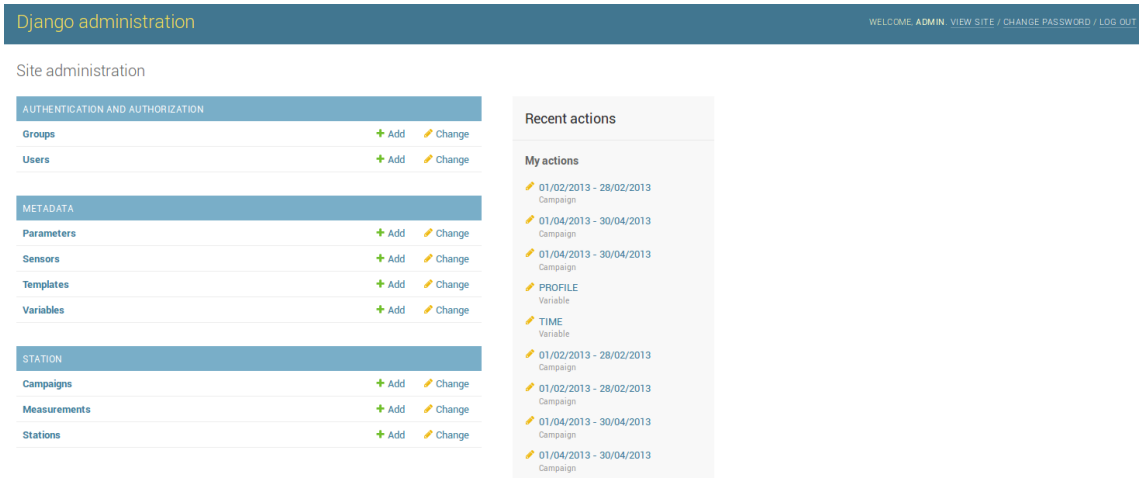

Figura 44: Página de Inicio de la administración

Tabla 12: Caso de uso Gestión de Parámetros

Nombre del caso de uso	Gestión de los Parámetros
Área a la que pertenece	Administración de la aplicación. Sección Metadata
Actores	Administrador del sistema
Condiciones previas	Sin condiciones
Descripción del caso de uso	El caso de uso se inicia cuando se entra a la sección **Parameters**. Una vez hecho esto, el sistema muestra un listado de los parámetros existentes y da la posibilidad de agregar o editarlos. Para agregar o editarlos es necesario proporcionar o actualizar la siguiente información: • Name : nombre estándar del parámetro. • Alias: la unidad de medida. • Description: breve descripción.
Resultado de la terminación	Como resultado, un parámetro es insertado/actualizado en la base de datos.

Notas del caso de uso	Cada parámetro constituye el tipo base de las variables que se le asocian.

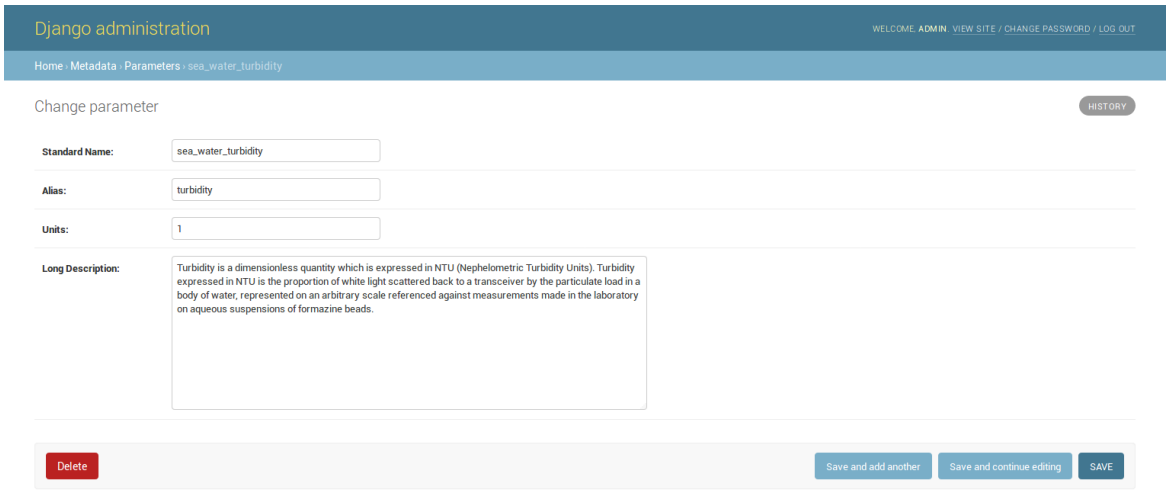

Figura 45: Página de edición de los datos de un parámetro

Para todos los casos de uso de gestión dentro de la administración, el sistema da cuatro posibilidades que son comunes. Ver opciones al final de la Figura 45:

- Delete: elimina el registro
- Save and add another: salva el registro y abre la página de Agregar Nuevo registro
- Save and continue editing: salva el registro y continua en la misma página de edición
- Save: salva el registro y regresa a la página del listado

Tabla 13: Caso de uso Gestión de Sensores

Nombre del caso de uso	Gestión de los Sensores
Área a la que pertenece	Administración de la aplicación. Sección Metadata
Actores	Administrador del sistema
Condiciones previas	Sin condiciones
Descripción del caso de uso	El caso de uso se inicia cuando se entra a la sección **Sensors**. Una vez hecho esto, el sistema muestra un listado de los sensores existentes y da la posibilidad de agregar o editarlos. Ver Figura 46. Para agregar o editarlos es necesario proporcionar o actualizar: • Name: nombre estándar del modelo. • Manufacturer: nombre del fabricante. • Reference: referencia del sensor.
Resultado de la terminación	Como resultado, un sensor es insertado/actualizado en la base de datos.

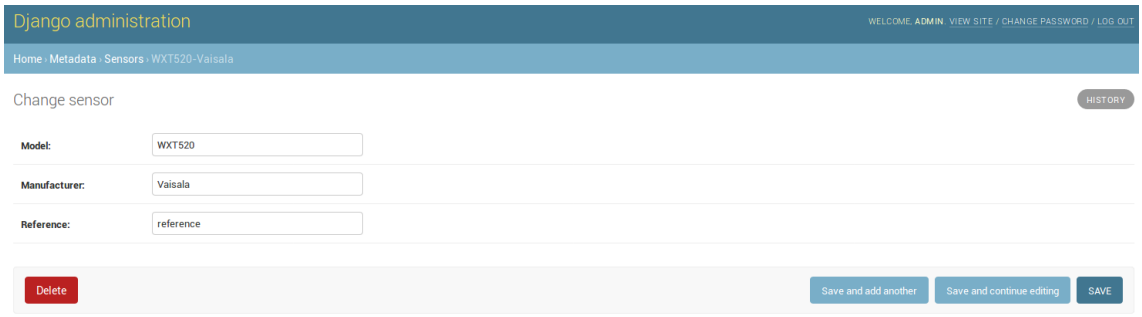

Figura 46: Página de edición de los datos de un sensor

Tabla 14: Caso de uso Gestión de Variables

Nombre del caso de uso	Gestión de las Variables
Área a la que pertenece	Administración de la aplicación. Sección Metadata
Actores	Administrador del sistema
Condiciones previas	Deben haber sensores y parámetros ya insertados
Descripción del caso de uso	El caso de uso se inicia cuando se entra a la sección **Variables**. Una vez hecho esto, el sistema muestra un listado de las variables existentes y da la posibilidad de agregar o editarlas. Ver Figura 47. Para agregar o editarlas es necesario proporcionar o actualizar: • Name: nombre. • Parameter: parámetro asociado • Sensor: sensor asociado. • Processed At Laboratory: fecha de procesamiento en el laboratorio. • Methodology: metodología empleada en caso de haberse procesado en el laboratorio. • Description: descripción de la variable. • Attributes: proporciona un editor JSON donde se pueden rellenar los valores de una serie de propiedades que describen a la variable. Las más importantes son la propiedad datacolumns que mapea a cuál variable del fichero de medición hará referencia y el typeof que declara el tipo de datos que contendrá.
Resultado de la terminación	Como resultado, una variable es insertada/actualizada en la base de datos.

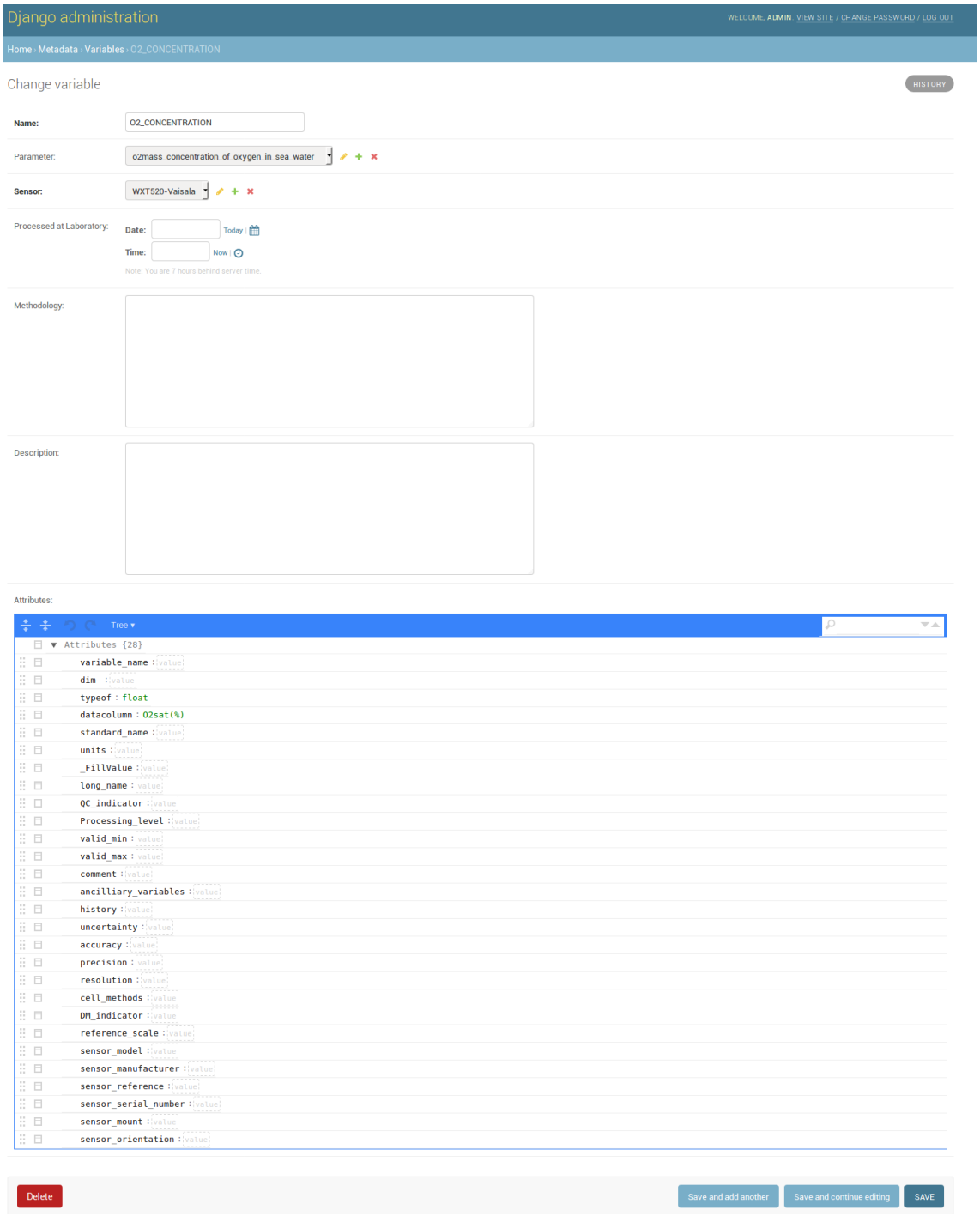

Figura 47: Página de edición de los datos de una variable

Tabla 15: Caso de uso Gestión de Plantillas

Nombre del caso de uso	Gestión de las Plantillas
Área a la que pertenece	Administración de la aplicación. Sección Metadata
Actores	Administrador del sistema

Condiciones previas	Deben haber variables ya insertadas
Descripción del caso de uso	El caso de uso se inicia cuando se entra a la sección **Templates**. Una vez hecho esto, el sistema muestra un listado de las plantillas existentes, ver Figura 48, y da la posibilidad de agregar o editarlas. Para agregar o editarlas es necesario proporcionar o actualizar: • Name: nombre. • DatasetID: identificador de la colección de datos de ERDDAP donde se almacenarán en formato netCDF las mediciones que estén asociadas a esta plantilla. • Variables: variables contenidas en las mediciones asociadas a esta plantilla. • Global attributes: proporciona un editor JSON donde se pueden entrar los valores de una serie de atributos que son globales a cada fichero netCDF asociado a la plantilla. Este campo es indispensable para almacenar las mediciones en formato netCDF. Ver Notas del caso de uso.
Resultado de la terminación	Como resultado, una plantilla es insertada/actualizada en la base de datos.
Notas del caso de uso	Se hablará de las mediciones en el caso de uso Gestión de las mediciones, de la sección **Station**.

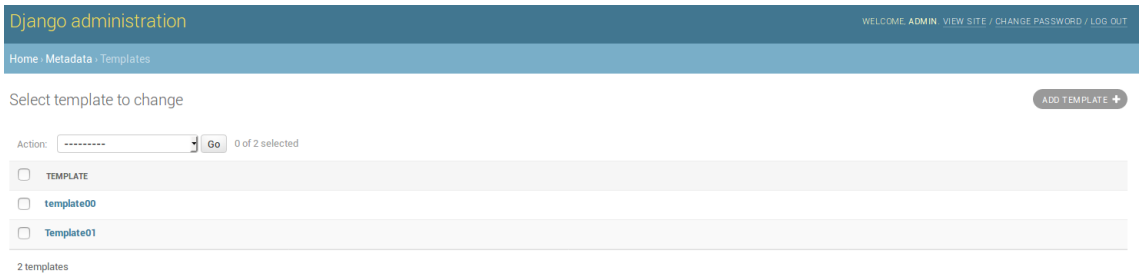

Figura 48: Listado de Plantillas

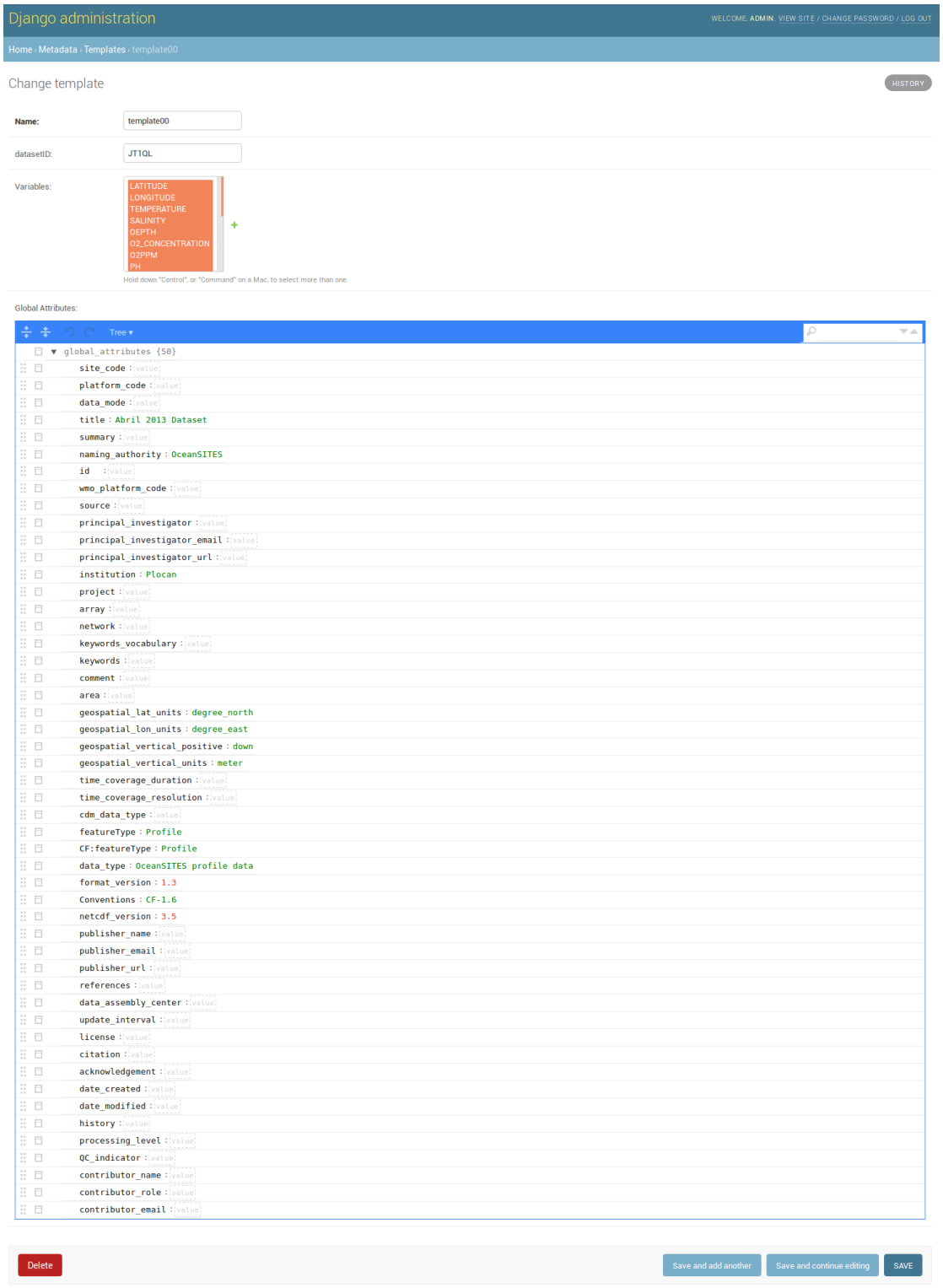

Figura 49: Página de edición de los datos de una plantilla

Tabla 16: Caso de uso Gestión de Estaciones

Nombre del caso de uso	Gestión de las Estaciones
Área a la que pertenece	Administración de la aplicación. Sección Station
Actores	Administrador del sistema
Condiciones previas	Sin condiciones
Descripción del caso de uso	El caso de uso se inicia cuando se entra a la sección **Stations**. Una vez hecho esto, el sistema muestra en la Figura 50 un listado de las estaciones existentes y da la posibilidad de agregar o editarlas. Ver Figura 51. Para agregar o editarlas es necesario proporcionar o actualizar: • Name: nombre • Location : localización en el mapa de la estación. Puede escribirse directamente la latitud y longitud en los respectivos campos de texto y al hacer click en el botón **Update marker position** se actualiza la posición del marcador en el mapa. También puede insertarse o moverse en el mapa el marcador y automáticamente se actualiza la latitud y longitud de los respectivos campos de texto
Resultado de la terminación	Como resultado, una estación es insertada/actualizada en la base de datos.
Notas del caso de uso	En la vista que muestra el listado de las estaciones se presenta un mapa con la posición de todas, dando la posibilidad de ver su posición al pasar el ratón sobre cada una.

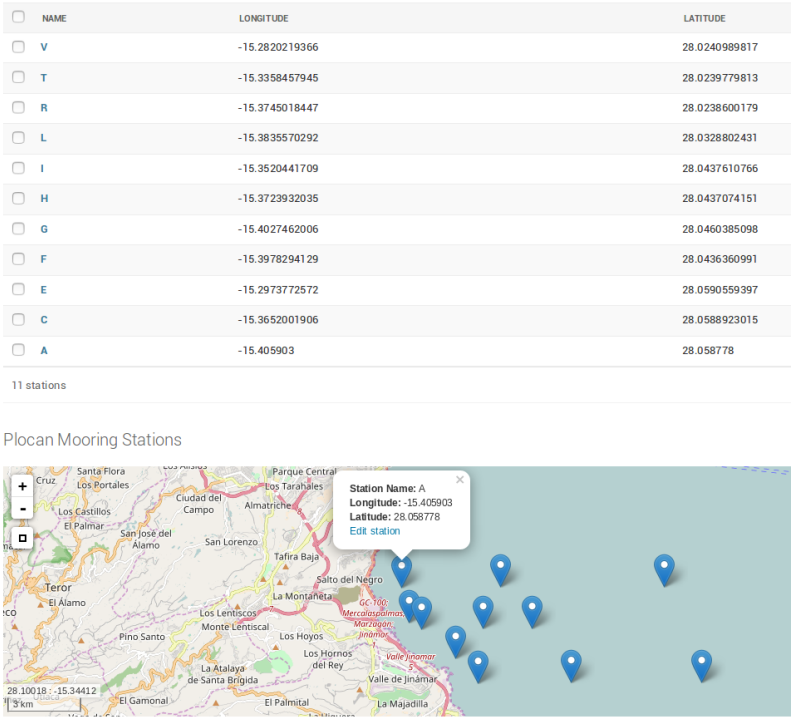

Figura 50: Listado de estaciones

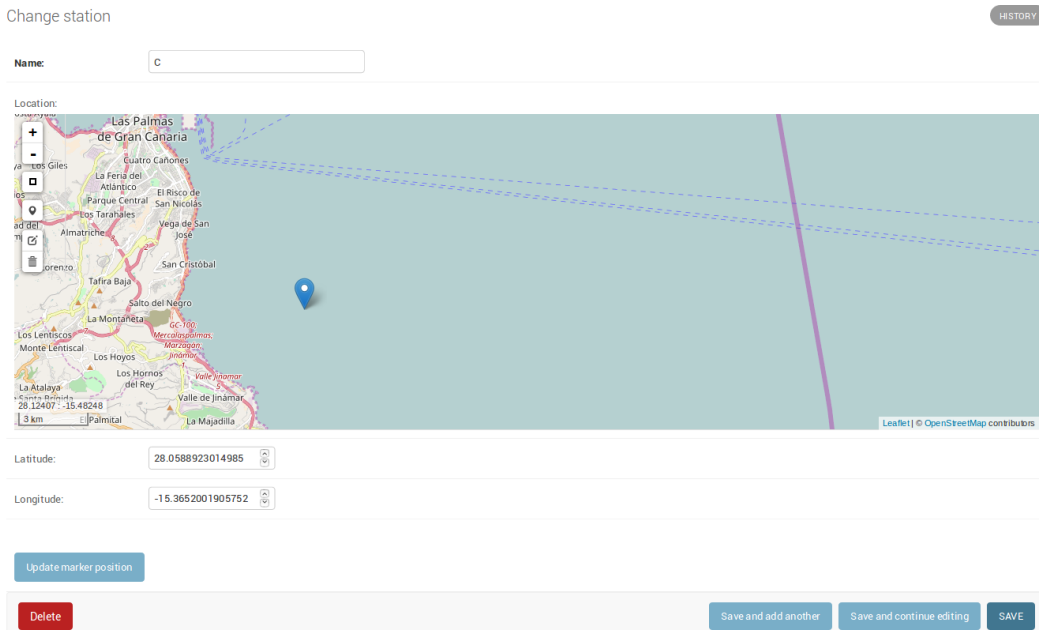

Figura 51: Página de edición de los datos de una estación

Tabla 17: Caso de uso Gestión de Mediciones

Nombre del caso de uso	Gestión de las Mediciones
Área a la que pertenece	Administración de la aplicación. Sección Station
Actores	Administrador del sistema
Condiciones previas	Deben haber estaciones y campañas ya insertadas
Descripción del caso de uso	El caso de uso se inicia cuando se entra a la sección Measurement. Una vez hecho esto, el sistema muestra un listado de las mediciones existentes y da la posibilidad de agregar o editarlas. Ver Figura 52. Para agregar o editarlas es necesario proporcionar o actualizar: • Station: nombre de la estación donde se obtuvo la medición. • Campaign : campaña en la cual se realiza la medición. • Template: plantilla que se asocia a los datos medidos con el propósito de su posterior almacenamiento en formato netCDF. Recordar que dicha plantilla especifica las variables contenidas en los datos medidos. • Data FileName: fichero de texto que contiene los datos de la medición. • Time: tiempo en que se realizó la medición de los datos. Este campo es de solo lectura pues el sistema es capaz de deducirlo a partir del fichero que contiene los datos. • JSON FileName: campo de solo lectura. Especifica la ruta del fichero JSON de metadatos necesario para poder convertir a netCDF el fichero de datos de la medición. • NetCDF FileName: campo de solo lectura. Especifica la ruta del

	fichero netCDF que almacena los datos medidos en este formato binario. • Dimension: campo de solo lectura que expresa el número de profundidades distintas a las que se tomaron los datos de la medición. Al salvar la información contenida, el sistema genera primeramente el fichero JSON de metadatos a partir de la información de la plantilla y estación asociada. Dentro de la estructura del fichero JSON, el sistema rellena los campos relacionados a la latitud y longitud de la estación que realiza la medición y el identificador en la base de datos de dicha estación. Todos estos datos se obtienen de la estación asociada. En cambio, la información del JSON relacionada con las variables se obtiene a través de la plantilla asociada. El JSON también necesita el tiempo de la medición el cual se obtiene del propio fichero subido en el campo Data FileName. Ver Tabla 18. Posteriormente con este fichero y el fichero de datos Data FileName, el sistema procede a generar el fichero netCDF gracias a la librería NetCDFGenerator.
Resultado de la terminación	Como resultado, una medición es insertada/actualizada en la base de datos. Por cada medición se salvan en el sistema local de archivos tres ficheros diferentes. Todos con diferentes extensiones pero que siguen un mismo patrón en sus nombres, es decir, el tiempo en que se realizó la medición: **Fichero de datos**: A_06-022013T17:51:33**.csv** **Fichero JSON**: A_06-02-2013T17:51:33**.json** **Fichero netCDF**: A_06-02-2013T17:51:33**.nc** Son almacenados de acuerdo a la estructura siguiente: **Fichero de datos**:files**/uploads/<**nombre-de-la-estacion>/<nombre-de-la-campaña>/<nombre-del-fichero> **Fichero JSON**: files**/json_templates/<**nombre-de-la-estacion>/<nombre-de-la-campaña>/<nombre-del-fichero> **Fichero netCDF**: files**/netCDF/<datasetID-de-la-plantilla>/**<nombre-de-la-estacion>/<nombre-de-la-campaña>/<nombre-del-fichero>

Tabla 18: Ejemplo del contenido simplificado de un fichero JSON de metadatos

```
{
   "global_attributes": {
      /* Aquí va contenido...*/
```

```
    "naming_authority": "OceanSITES",
    "id": "A",
    "featureType": "Profile",
    "CF:featureType": "Profile",
    "data_type": "OceanSITES profile data",
    "format_version": "1.3",
    "Conventions": "CF-1.6 ",
    "netcdf_version": "3.5"
  },
  "dimensions": [
    {
      "dimension_name": "DEPTH",
      "length": 41
    }
  ],
  "variables": [
    {
      "variable_name": "PROFILE",
      "typeof": "i",
       "cf_role": "profile_id",
      "value": 1
    },   {
      "variable_name": "TIME",
      "typeof": "double",
      "standard_name": "time",
      "units": "seconds since 1970-01-01",
      "long_name": "seconds since 1970-01-01",
      "value": 1360173093.4
    },{
      "variable_name": "LATITUDE",
      "typeof": "f4",
      "standard_name": "latitude",
      "units": "degree_north",
      "value": 28.058778
    },
    {
      "variable_name": "LONGITUDE",
      "typeof": "f4",
     "standard_name": "longitude",
      "units": "degree_east",
      "value": -15.405903
    },
    {
      "variable_name": "TEMPERATURE",
      "dim": "DEPTH",
      "typeof": "float",
      "datacolumn": "Temp",
      "standard_name": "sea_water_temperature",
      "units": "K",
      "sensor_model": "WXT520",
      "sensor_manufacturer": "Vaisala"
    },
    {
      "variable_name": "O2_CONCENTRATION",
      "dim": "DEPTH",
      "typeof": "float",
      "datacolumn": "O2sat(%)",
```

```
        "standard_name": "o2mass_concentration_of_oxygen_in_sea_water",
        "units": "%",
        "sensor_model": "WXT520",
        "sensor_manufacturer": "Vaisala",
      },
     /* Aquí van otras variables...*/
   ]
}
```

La Tabla 18 muestra el contenido de un fichero de metadatos JSON generado a partir de la estación A con identificador 1 como se puede ver en la definición de la variable PROFILE. En las variables LATITUDE y LONGITUDE se almacena la ubicación de la estación A. En la dimensión DEPTH se guarda el número de observaciones a diferentes profundidades contenidas en el fichero de perfiles verticales asociado al JSON. También se salva el tiempo de la medición en la variable TIME. Finalmente se muestran variables como TEMPERATURE y O2_CONCENTRATION, con su respectivos nombres estándar y el campo datacolumn, que constituye el nombre de la columna del fichero de perfiles verticales donde se encuentran los datos de cada variable.

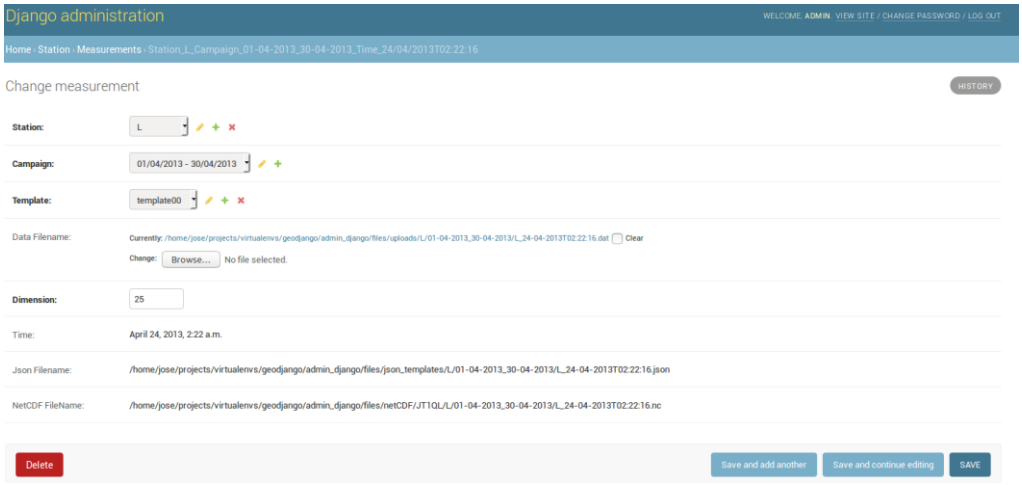

Figura 52: Página de edición de los datos de una medición

Tabla 19: Caso de uso Gestión de campañas

Nombre del caso de uso	Gestión de las Campañas
Área a la que pertenece	Administración de la aplicación. Sección Station
Actores	Administrador del sistema
Condiciones previas	Este caso de uso permite asociar múltiples mediciones a la campaña. En caso de asociar alguna medición a la campaña, es necesario que exista alguna plantilla ya definida. En caso de no asociársele ninguna medición pues no tiene condiciones previas
Descripción del caso de uso	El caso de uso se inicia cuando se entra a la sección **Campaigns**. Una vez hecho esto, el sistema muestra un listado de las campañas existentes y da la posibilidad de agregar o editarlas. Ver Figura 53. Para agregar o editarlas es necesario proporcionar o actualizar: • Name: nombre de la campaña.

	• Project: nombre del proyecto. • Institution: nombre de la institución que patrocina la campaña. • Ship: nombre del barco en caso de participar alguno. • Chief Campaign: nombre del jefe de la campaña. • Arrival and Departure Location: puerto de partida y puerto de llegada del barco. • Starting Date: fecha de inicio de la campaña. • Ending Date: fecha de finalización de la campaña. • Personal: nombre del personal involucrado. • Extent Area: área de extensión de la campaña. El mapa da la posibilidad de arrastrar y soltar el rectángulo que representa el área o de actualizar su posición mediante los controles de texto de su esquina superior e inferior. A continuación de los campos anteriores se encuentra el área que permite insertar múltiples mediciones asociadas a esta campaña. Por cada fila se ejecuta el caso de uso Gestión de las Mediciones.
Resultado de la terminación	Como resultado, una campaña es insertada/actualizada en la base de datos. En caso de tener mediciones asociadas, estas pueden ser insertadas por primera vez o actualizadas, según sea el caso.

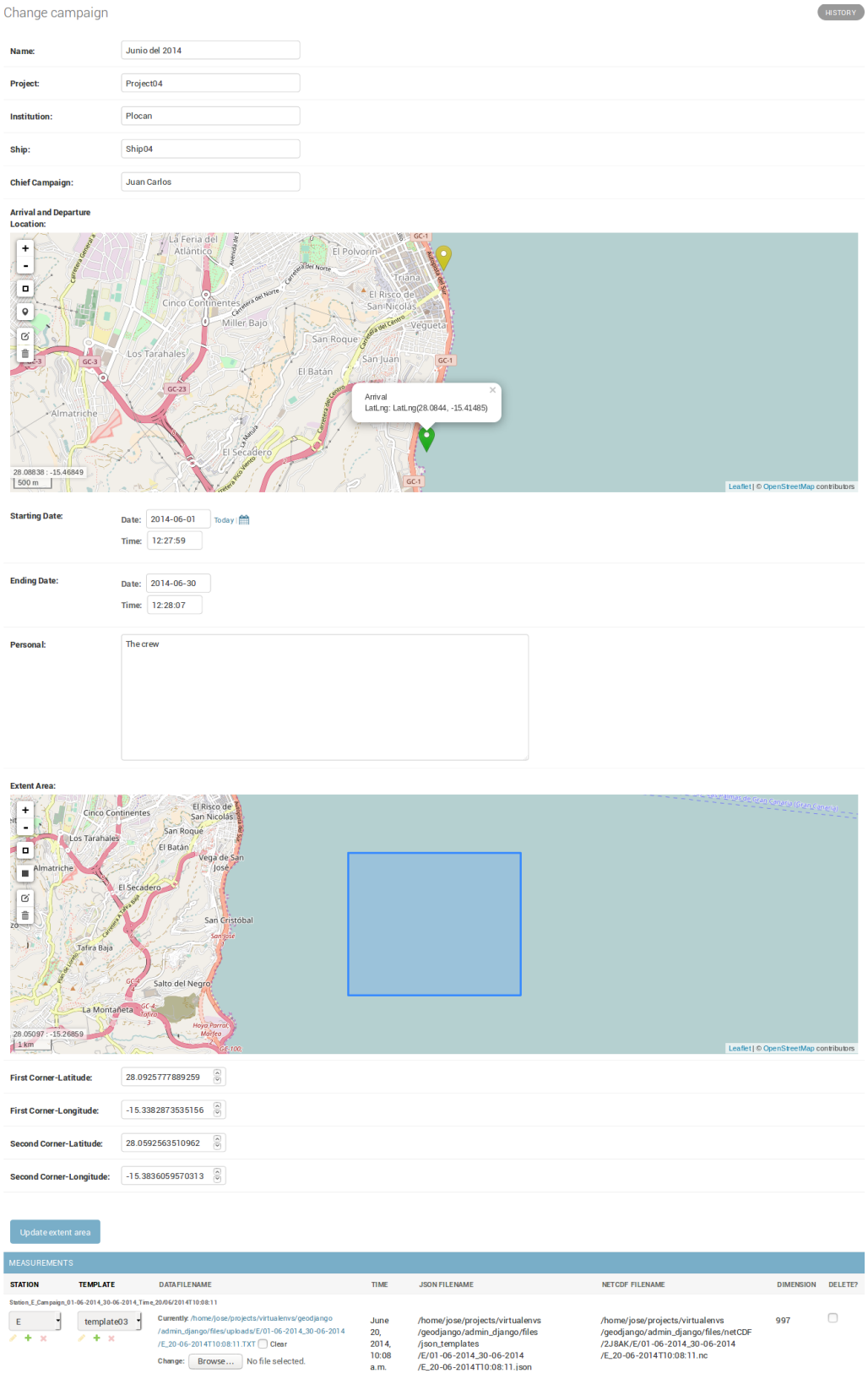

Figura 53: Página de edición de los datos de una campaña

8.6.1.1. Casos de uso de la aplicación Angular

Esta sección presenta los casos de uso disponibles a los usuarios con el rol de usuarios que navegan por internet y acceden a la parte pública de la web.

Tabla 20: Caso de uso Visualización de estaciones

Nombre del caso de uso	Visualización de las estaciones
Área a la que pertenece	Parte pública de la aplicación
Actores	Cualquier usuario
Condiciones previas	Que la información a mostrar se encuentre disponible en la base de datos Postgres
Descripción del caso de uso	El caso de uso se inicia cuando se carga la página de inicio de la aplicación Angular. El sistema lista una serie de marcadores en el mapa que representan las estaciones de observación. Ver Figura 54. Se puede seleccionar cada estación haciendo click sobre ella y se despliega una pequeña ventana con información sobre su nombre y localización. Además, al realizar esta acción automáticamente se selecciona en la lista desplegable del buscador de estaciones, la estación seleccionada en el mapa.
Resultado de la terminación	Como resultado, un listado de estaciones se visualiza en el mapa.

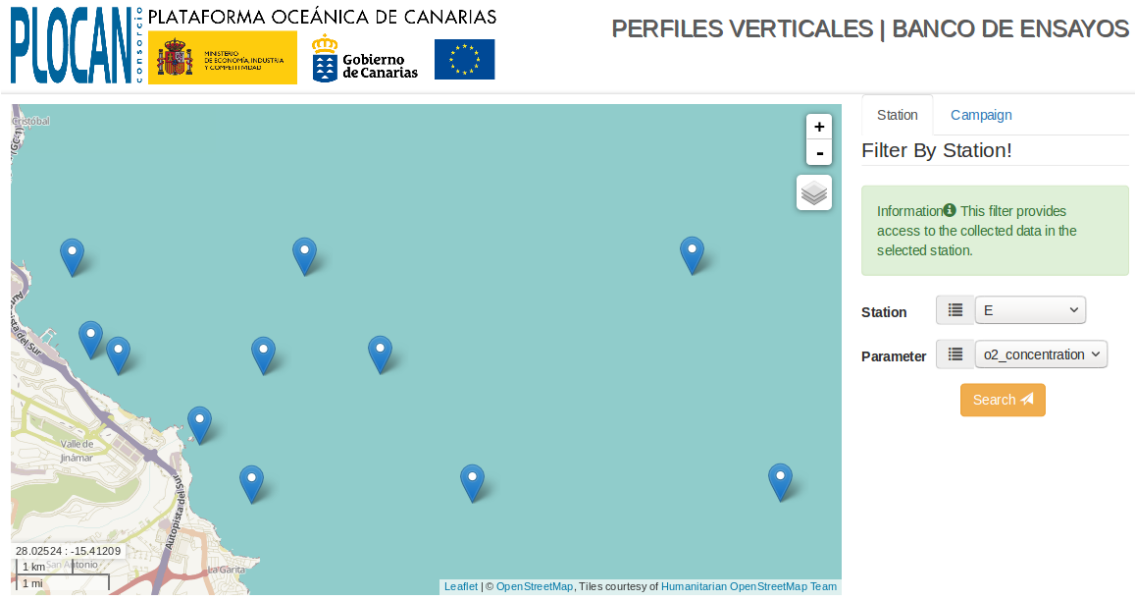

Figura 54: Visualización de estaciones en el mapa

Tabla 21: Caso de uso Búsqueda por estación

Nombre del caso de uso	Búsqueda por estación
Área a la que pertenece	Parte pública de la aplicación

Actores	Cualquier usuario
Condiciones previas	Que la información a mostrar se encuentre disponible en la base de datos Postgres y en ERDDAP
Descripción del caso de uso	El caso de uso se inicia cuando después de entrar en la pestaña Station del menú vertical de la derecha de la página, se selecciona una estación, un parámetro y se pulsa el botón Search. El sistema realiza una consulta a la API REST de Django en busca de aquellas campañas en las que haya participado la estación seleccionada y que contengan entre sus variables el parámetro seleccionado. La respuesta contiene todas las mediciones que cumplen esas condiciones, incluye información adicional del dataset de ERDDAP donde se encuentran almacenadas en formato netCDF. Con la información del dataset, el identificador de la estación de interés y el parámetro seleccionado la aplicación realiza una nueva búsqueda utilizando la API REST de ERDDAP.
Resultado de la terminación	Como resultado de la búsqueda se presenta un listado de gráficas que muestran los perfiles verticales del parámetro seleccionado, pertenecientes a la estación seleccionada y organizados por campañas. En la vista Compact View, los resultados se muestran en una única gráfica con los perfiles verticales (del parámetro seleccionado) de todas las campañas en la que participó la estación. Esto facilita la comparación de los perfiles. Ver Figura 55. La vista Extended View muestra por separado cada perfil en su propia gráfica.

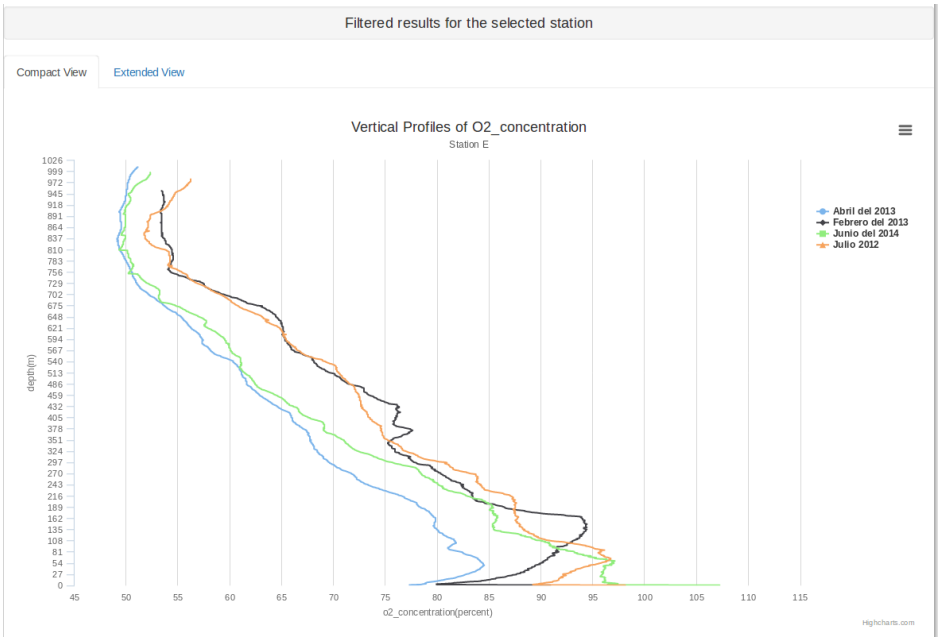

Figura 55: Ejemplo de Vista compacta del resultado de la búsqueda por estación.

Tabla 22: Caso de uso Búsqueda por campaña

Nombre del caso de uso	Búsqueda por campaña
Área a la que pertenece	Parte pública de la aplicación
Actores	Cualquier usuario
Condiciones previas	Que la información a mostrar se encuentre disponible en la base de datos Postgres y en ERDDAP
Descripción del caso de uso	El caso de uso se inicia cuando después de entrar en la pestaña Campaign del menú vertical de la derecha de la página, se selecciona una campaña, un parámetro y se pulsa el botón Search. El sistema realiza una consulta a la API REST de Django en busca de aquellas estaciones que hayan participado en la campaña seleccionada y en las cuáles se haya medido el parámetro seleccionado. La respuesta contiene todas las mediciones que cumplen esas condiciones, incluye información adicional del dataset de ERDDAP donde se encuentran almacenadas en formato netCDF. Con la información del dataset, el rango de fechas de la campaña de interés y el parámetro seleccionado la aplicación realiza una nueva búsqueda utilizando la API REST de ERDDAP.
Resultado de la terminación	Como resultado de la búsqueda se presenta un listado de gráficas que muestran los perfiles verticales del parámetro seleccionado, pertenecientes al rango de fechas de la campaña seleccionada y organizadas por estación.

	En la vista Compact View, los resultados se muestran en una única gráfica con los perfiles verticales (del parámetro seleccionado) de todas las estaciones participantes en la campaña. Esto facilita la comparación de los perfiles. En la vista Extended View se muestra por separado cada perfil en su propia gráfica. Ver Figura 56. En la Tabla 23 se aprecia cómo HighCharts permite hacer zoom sobre cada gráfica o exportarla en diferentes formatos de imagen.

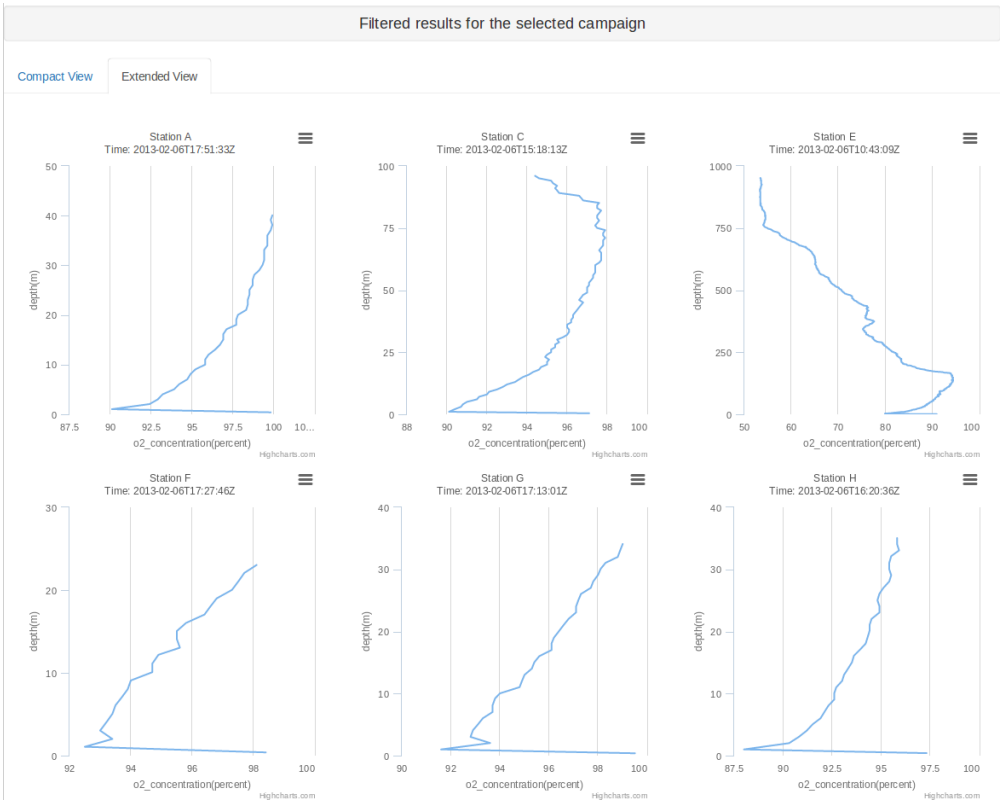

Figura 56: Ejemplo de Vista extendida de los resultados de la búsqueda por campaña

Tabla 23: Características de Zoom y Exportar desde HighCharts