Centro Universitario de la Defensa en la Escuela Naval Militar

TRABAJO FIN DE GRADO

Prueba de concepto de un modelo de sistema de información para su implantación en un gemelo digital oceanográfico

Grado en Ingeniería Mecánica

ALUMNO: Enrique Manuel Barahona Peláez

DIRECTORES: Milagros Fernández Gavilanes

Carlos Pérez Collazo

CURSO ACADÉMICO: 2021-2022

Universidade Vigo

Centro Universitario de la Defensa
en la Escuela Naval Militar

TRABAJO FIN DE GRADO

Prueba de concepto de un modelo de sistema de información para su implantación en un gemelo digital oceanográfico

Grado en Ingeniería Mecánica

Intensificación en Tecnología Naval

Cuerpo General

Universidade Vigo

RESUMEN

La implantación de las tecnologías 4.0 constituye una de las prioridades básicas en investigación tanto de la Armada como de la sociedad civil. Así, la definición de gemelos digitales avanzados que permitan una óptima gestión son clave para adaptarse a un entorno cambiante y dinámico. En este sentido la hibridación de la metodología Building Information Modelling (BIM) con la gestión de datos tanto en tiempo real como de series históricas, da lugar a los denominados BIM Digital Twin. En este trabajo se busca generar el entorno para un gemelo digital oceanográfico acerca de los datos relevantes para la navegación dentro de las rías de Pontevedra y Vigo. Para ello, este trabajo podría dividirse en dos secciones: en la primera se hace una recogida de datos de diferentes fuentes oficiales para a continuación, darles una estructura utilizando el lenguaje de código Python para su posterior inserción en una base de datos no estructurada; mientras que la segunda trata de todo el apartado referente a la semántica, en esta se establecen todas las relaciones entre los datos para que, posteriormente, se puedan realizar consultas SPARQL accediendo a dicha base de datos.

PALABRAS CLAVE

Gemelo digital, base de datos semántica, Python, RDF, SPARQL

AGRADECIMIENTOS

Agradezco a mi familia, amigos y compañeros de promoción por el apoyo recibido en la realización de este proyecto y, en especial, por haber estado ahí y por todo lo que me han aportado estos últimos cinco años de Escuela tanto en los momentos buenos como en los no tan buenos. Por último, quería agradecer a mis tutores D.ª Milagros Fernández Gavilanes y D. Carlos Pérez Collazo por haberme guiado en todo momento en la realización de este trabajo fin de grado

CONTENIDO

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN Y OBJETIVOS

1.1 Motivación del trabajo

El presente trabajo viene motivado por la necesidad de adaptar las diferentes fuentes de información a una estructura de base de datos que permita conceptualizar un gemelo digital oceanográfico. Esta mencionada necesidad viene del programa de modernización de la Armada llamado Transformación Digital Armada 4.0, con el que la Armada pretende adaptarse a un entorno tecnológico de cambio constante y de esta manera aprovechar, a su vez, todas las ventajas y oportunidades que les puede brindar las nuevas tecnologías.

Uno de los factores más importantes a la hora de planear una navegación es la meteorología, este elemento es de vital importancia pues dependiendo de él y del tipo de buque en cuestión se podrá determinar si sería o no recomendable establecer una zona de ejercicios, una derrota o cualquier tipo de actividad marítima en una zona determinada.

Este trabajo pretende servir como referencia a la hora de tomar esas decisiones, ya que no solo se trata de una base de datos en la que se almacena información, se han establecido una serie de relaciones entre todos los datos para que, a la hora de realizar consultas, nos faciliten en gran medida la obtención del resultado que queremos. De esta forma se busca que el usuario pueda acceder fácilmente a una información concreta ya sea consultándola a través de una fecha, una localización, un valor o todas ellas.

1.2 Objetivos

Como se verá más adelante, este trabajo extrae la información de unos archivos en formato JSON descargables de páginas web como MeteoGalicia, Puertos del Estado, EMODnet, etc. La razón de ser de este trabajo es la definición de una prueba de concepto de gemelo digital oceanográfico, para lo que se plantean los siguientes objetivos:

- Manejar y procesar la información de las distintas fuentes existentes.
- Familiarizarse con el concepto de base de datos semántica.
- Entender las posibles interrelaciones de los datos extraídos de las distintas fuentes.
- Familiarizarse con los métodos de consulta semántica SPARQL.

1.3 Estructura del trabajo

En este apartado se va a definir de qué forma se ha estructurado el presente trabajo. Este trabajo se compone de seis capítulos. El primero consiste en una breve introducción en la que se define cuál ha sido la motivación de este trabajo y qué objetivos se buscan con la realización del mismo.

El segundo capítulo está dedicado al estado del arte el cual a su vez se compone de cuatro apartados que tratan acerca de los gemelos digitales, la metodología BIM, las fuentes de datos fiables de las que se disponen y por último un subapartado de semántica en que se hace referencia a términos con RDF, SPARQL y se explican las diferencias que tienen las bases no estructuradas de las estructuradas.

El tercer capítulo podría considerarse el núcleo principal del trabajo, este también esta a su vez dividido en distintos apartados, comienza describiendo la estructura del sistema en conjunto determinado el papel que tienen las distintas herramientas, en el siguiente apartado explica todas las herramientas que se van a emplear en el trabajo, el tercer apartado trata de la metodología y narra cómo se descargan los datos y cuáles son los datos que se van a descargar aportando información acerca de qué parámetros meteorológicos y que estaciones se van a emplear en el trabajo, también hace referencia a cómo se estructuran los datos en AllegroGraph y cuáles son las relaciones establecidas entre los datos. A continuación, dispone de otro apartado que trata la implementación de los datos, en este apartado se define a grandes rasgos de qué manera se vuelcan los datos en MongoDB y de qué forma se establecen las tripletas de AllegroGraph a través de código Python.

En el capítulo cuatro se explica qué propiedades tenían los archivos descargados y, a la hora de trabajar con diferentes fuentes, cuantos objetos se generan en la base de datos MongoDB. En el apartado cinco se recopilan las conclusiones obtenidas una vez finalizado el trabajo y las líneas futuras que se podrían seguir con respecto al mismo. El capítulo seis recoge toda la bibliografía y para finalizar, este trabajo consta de tres anexos: uno acerca de cómo instalar las herramientas necesarias para realizar este trabajo y los otros dos acerca del código tanto de volcado de datos en MongoDB como de establecimiento de relaciones en AllegroGraph.

2 ESTADO DEL ARTE

2.1 Gemelo digital

Un gemelo digital no es más que la simulación de un proceso físico que se efectúa de forma simultánea al mismo y que, por lo general, suele coincidir exactamente con el funcionamiento del proceso físico que tiene lugar en tiempo real. Esta tecnología apareció por primera vez en los primeros años de la década de los 2000 por parte de Michael Grieves, cuya experiencia en el diseño de productos hizo que inicialmente se concibiese para la ingeniería de producción. Sin embargo, desde su creación, el concepto se ha ampliado hacia el sentido de que ahora se le aplica, o más bien se utiliza, para caracterizar una variedad de modelos de simulación digital que se ejecutan junto a los procesos en tiempo real que pertenecen a los sistemas sociales y económicos, así como a los sistemas físicos, como es nuestro caso [1].

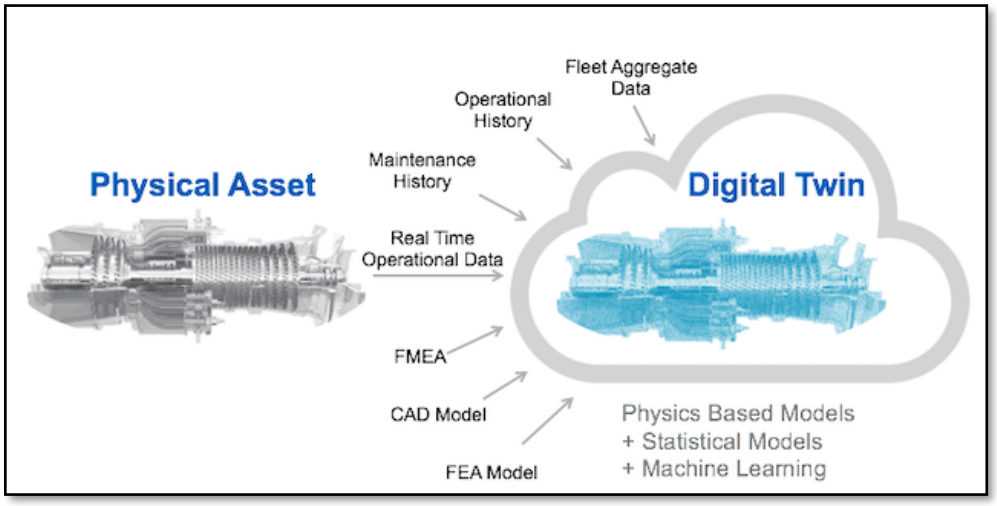


Figura 2-1 Representación gráfica de un gemelo digital [2]

En general, cualquier sistema que refleje el funcionamiento de otro sistema diferente suele definirse como un modelo, que a su vez es una representación de la estructura y los procesos que definen el sistema con el que se está comparando. Los modelos son, por definición, simplificaciones de lo real y, en ese sentido, no pretenden replicar el sistema original con el mismo detalle que éste. En resumen, se desechan las características clave del sistema original y, por lo general, el modelo abstrae lo que se considera clave de las características del sistema real que se está examinando. Básicamente, los modelos no intentan

reflejar todo lo que hay en un sistema real y esto es lo que hace que sean intrínsecamente diferentes del sistema original. De hecho, otra diferencia es que la representación física de cualquier sistema suele diferir de los modelos [1].

Por lo tanto, parece que, en general, un modelo informático de un sistema físico nunca puede ser la base de un gemelo digital, ya que muchos elementos del sistema real se ignoran en cualquier abstracción de este tipo. Sin embargo, no cabe duda de que algunos modelos se acercan más a la realidad que otros, ya que toda la variedad de modelos va desde los "experimentos teóricos", que son totalmente conceptuales, hasta las representaciones digitales muy adaptadas que intentan reflejar el mayor número posible de características del sistema real. Incluso es posible concebir una transformación de un modelo digital desde una concepción totalmente abstracta hasta una imagen completa del sistema en cuestión. Sin embargo, si el modelo es una imagen fiel del sistema, que es la definición que se da de un gemelo digital, se podría argumentar que el gemelo digital ya no está separado del sistema, sino que es el propio sistema. En este sentido, todos los sistemas físicos pueden tener un equivalente digital que converge y se fusiona con el sistema en cuestión. Por tanto, un verdadero gemelo digital que funcione en tiempo real no es diferente del propio sistema, lo que plantea la cuestión de cómo se puede utilizar el gemelo digital para aprender sobre el sistema y para explorar, simular y probar nuevos diseños. Para que el gemelo digital pueda utilizarse de este modo, es de suponer que tiene que estar desconectado del sistema real. Entonces, hay una dificultad lógica para hacerlo, ya que ambos sistemas funcionarán a la par y si son imágenes especulares el uno del otro, la pregunta se convierte en "cómo puede utilizarse el gemelo digital para explorar e informar al gemelo original" [1].

A continuación, se analizarán los gemelos digitales orientados a la oceanografía. Las Observaciones Oceánicas son una parte esencial de los esfuerzos mundiales para comprender y proteger los sistemas socio-ecológicos marinos al tiempo que se benefician de sus recursos marinos. Pueden ser muestras recogidas en barcos, mediciones de instrumentos en plataformas fijas, sistemas autónomos y a la deriva, plataformas sumergibles, barcos en el mar o sistemas de observación a distancia, como satélites y aviones [3].

Hace 10-20 años, los datos marítimos de estas observaciones eran difíciles de encontrar, sólo accesibles a través de largas y a veces costosas negociaciones y difíciles de reunir para crear una imagen completa debido a las diferentes normas, nomenclatura y líneas de base. La previsión oceánica era una actividad de investigación. En las últimas dos décadas, la Unión Europea ha invertido en políticas e infraestructuras para que el conocimiento del océano fuera fundamental en las políticas medioambientales y climáticas, así como asentarse la base de la economía azul. Sus Estados Miembros, junto con sus vecinos, han creado una infraestructura de datos y previsiones marítimas sin precedentes. El trabajo conjunto y los principios de acceso libre y gratuito, interoperabilidad y "medir una vez, usar muchas veces", son promovidos en gran medida a través de, Copernicus, los Programas Marco de Investigación Europeos FP7 y Horizonte 2020, y las actividades de EMODnet han demostrado un claro valor [3].

Figura 2-2 Logo de la Comisión Europea [4]

El Gemelo Digital del Océano es el siguiente paso, que satisface la necesidad de integrar una amplia gama de datos fuentes, transformar los datos en conocimiento y conectar, comprometer y dar a los

ciudadanos, gobiernos e industrias la capacidad de informarse previamente antes de tomar decisiones. Ello permitirá potenciar una responsabilidad común para vigilar, preservar y mejorar los hábitats marinos, y apoyar una economía azul sostenible (pesca, acuicultura, transporte, energía en alta mar, etc.). Esto mismo debe permitir evaluar el estado de los ecosistemas, los hábitats y el impacto de la actividad humana; la previsión de sus cambios a corto y largo plazo; el desarrollo de estrategias de fomento de la biodiversidad; la gestión de actividades económicas sostenibles; el desarrollo de planes de mitigación, adaptación y sustitución para hacer frente a los riesgos climáticos y la optimización de las respuestas de emergencia a acontecimientos graves como la marejada ciclónica [3].

La Unión Europea contribuirá al desarrollo de modelos digitales interactivos de alta resolución de los océanos. Aprovechando la integración de las actuales capacidades de vanguardia de la UE en materia de observación de los océanos (como Eurofleets+, EuroArgo, Jerico, EMBRC, etc.), infraestructuras de datos y servicios de previsión (Copernicus, EMODnet, Blue Cloud, ERICs) a través de tecnología informática innovadora [3].

Como podemos observar la Comisión Europea busca dar el paso a un gemelo digital oceanográfico para tratar asuntos en relación a ecosistemas, biodiversidad y planes sostenibles. Sin embargo, en este presente trabajo el uso que se le busca dar al gemelo digital oceanográfico es puramente con fines de navegación y centrado únicamente en las Rías Bajas.

2.2 Metodología BIM

La metodología de trabajo BIM es aplicada en el sector de la construcción, consiste en organizar un gran conjunto de datos relacionados con el edificio a construir con el fin de facilitar la gestión de proyectos de ingeniería, arquitectura y construcción permitiendo a todo el personal implicado en el proyecto acceder en tiempo real a la misma información y al estado del proyecto [5].

En programas de modelaje es muy sencillo discernir y nombrar objetos, podemos observar que una puerta es una puerta y que una silla es una silla. Por el contrario, en una base de datos no existe información espacial por lo que los elementos no se pueden agrupar según su apariencia o según donde se encuentren. En lugar de ello se estructuran basándose en la función que desempeñan. En el momento de crear una base de datos no se separan las habitaciones basándose en la planta en la que se encuentran sino en concordancia con los departamentos en los que está separado el edificio. Esto se debe a que en el momento en el que se comienza a diseñar no se suele disponer de un esquema del proyecto y el agrupar la información de esta forma nos permite controlar todas las zonas que cumplan una misma función específica [6].

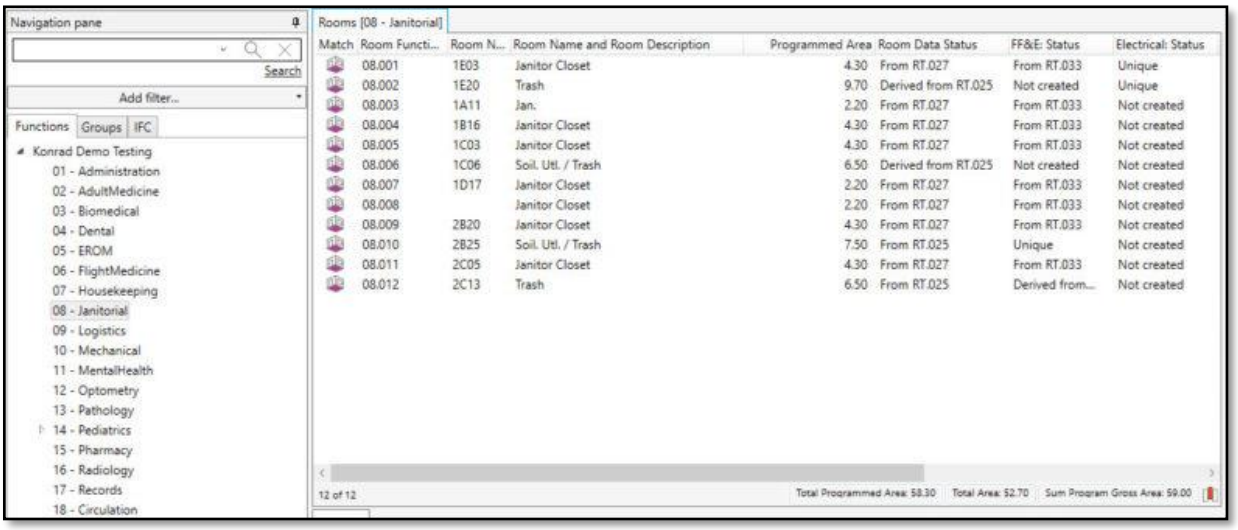


Figura 2-3 Ejemplo de BIM aplicado a un hospital [6]

El siguiente paso sería el de establecer, dentro de cada departamento, las distintas salas pues, aunque se puedan encontrar en zonas distintas todas ellas pertenecen a un grupo que desarrolla una misma función común. Una vez dentro de las mencionadas salas encontramos las características que deben de cumplimentar desde la superficie en metros cuadrados hasta las ventanas que tiene que tener. En el siguiente nivel podríamos encontrar los elementos de los que debería disponer una sala como muebles, lámparas, radiadores, etc. Cada elemento a su vez se iría agrupando según su campo en diferentes categorías lo que facilita bastante la búsqueda del mismo. Por poner un ejemplo, no es lo mismo que llegue un electricista y tenga que buscar en el listado de todos los elementos a que llegue y busque en la categoría de energía y alumbrado. Lo siguiente que nos encontramos serían las especificaciones de los objetos como pueden ser las de una puerta, determinando el tipo de pomo si es redondo o de manilla, el color y el material del mismo; el color de la puerta; si debe o no disponer de pestillo, como ha de ser el pestillo; las dimensiones de la puerta, etc. [6].

2.3 Fuentes de datos

Ya hemos hablado de su aplicación y de en qué se basará la estructura de los mismos, en este apartado se recopilarán unas de las principales fuentes que se emplearán para la descarga de datos. A la hora de buscar información, la web está llena de lugares donde podemos encontrar todo tipo de respuestas a nuestras consultas, sin embargo, ha de ser trabajo propio el saber discernir cuáles son de confianza y muestran datos fidedignos y cuáles, por el contrario, no. Es por ello que a la hora de descargar los datos necesarios para este proyecto se ha decidido emplear únicamente páginas web pertenecientes a entidades oficiales.

2.3.1 EMODnet

Los datos del medio marino son un valioso recurso. El acceso rápido a información fiable y precisa es vital para hacer frente a las amenazas que pesan sobre el medio marino, para la elaboración de políticas y legislación que protejan las zonas vulnerables de nuestras costas y océanos, para comprender las tendencias y para prever los cambios futuros. Asimismo, unos datos marítimos de mejor calidad y más fácilmente accesibles son un requisito previo para un mayor desarrollo económico sostenible, el llamado "crecimiento azul". Lamentablemente la recopilación, el almacenamiento y el acceso a los datos marinos en Europa se ha llevado a cabo de forma fragmentada durante muchos años. La mayor parte de la recogida de datos se ha centrado en satisfacer las necesidades de un único propósito por parte de una amplia gama de organizaciones privadas y públicas, a menudo aisladas entre sí [7].

La Red Europea de Observación y Datos Marinos (EMODnet) es una red de organizaciones apoyadas por la política marítima integrada de la UE. Estas organizaciones colaboran en la observación del mar, procesan los datos según normas internacionales y ponen esa información a disposición del público en forma de capas de datos interoperables. Esta filosofía de "recoger una vez y usar muchas veces" beneficia a todos los usuarios de datos marinos, incluidos los responsables políticos, los científicos, la industria privada y el público. Se calcula que esta política integrada de datos marinos permitirá ahorrar al menos mil millones de euros al año, además de abrir nuevas oportunidades de innovación y crecimiento [7].

Figura 2-4 Logo de EMODnet [8]

EMODnet proporciona acceso a una variedad de datos marítimos distribuidos en 7 categorías:

- Batimetría: proporciona información de batimetría, líneas de costa y pecios.
- Biología: proporciona datos sobre la distribución temporal y espacial de las especies.
- Química: proporciona datos acerca de la concentración de nutrientes, materia orgánica, pesticidas y metales pesados en el agua y los sedimentos.
- Actividad humana: proporciona información acerca de la intensidad y extensión de la actividad humana en la mar.
- Física: datos de salinidad, temperatura, oleaje, viento, altura del nivel del mar, atenuación de la luz y ruido submarino.
- Hábitats del lecho marino: datos, cartas y modelos sobre la distribución espacial y la extensión de los hábitats y las comunidades de los fondos marinos.

2.3.2 AEMET

La Agencia Estatal de meteorología se fundó en 2008 remplazando a la histórica Dirección General del Instituto Nacional de Meteorología. La función principal de AEMET es la de desarrollar, implantar y presentar los servicios meteorológicos de competencia del Estado con el objetivo de contribuir a la protección de vidas y bienes mediante una adecuada predicción y vigilancia de los fenómenos meteorológicos adversos. Es el responsable de todas aquellas actividades relacionadas con la meteorología a nivel estatal, así como representante directo en ámbitos internacionales.

AEMET ofrece multitud de servicios entre los que destacan la emisión de avisos y predicciones meteorológicas que puedan suponer un riesgo directo sobre la seguridad de las personas o bienes, servicios de apoyo a la navegación aérea y marítima con fines de seguridad y eficiencia en el tránsito y la actualización de los registros históricos de meteorología. El portal de AEMET proporciona una serie de datos accesible para cualquier tipo de usuario que los requiera, en este proyecto nos centraremos en dos apartados en concreto: el tiempo y los datos abiertos.

El apartado del tiempo se divide en dos subapartados, uno acerca de la observación de la meteorología donde nos muestra información de los últimos días referente a temperatura, viento, rachas, precipitaciones, presión y humedad. Mientras que el otro es acerca de la predicción del mismo donde,

más allá de poder escoger la zona de predicción, podemos escoger la predicción meteorológica orientada a tránsitos marítimos o aeronáuticos, visualizar mapas de frentes o incluso modelos numéricos.

AEMET OpenData es una API REST [1]a través de la cual se puede realizar la descarga de datos de forma gratuita. Permite dos tipos de acceso: el acceso general y el AEMET OpenData API. El acceso general está destinado al usuario medio, es de tipo gráfico con una interfaz muy intuitiva y los datos que ofrece son de carácter puntual. En cuanto al acceso de AEMET OpenData API, ofrece otro tipo de interacción con los datos, no está pensado para ser consultado por un usuario medio sino para que terceras entidades pueden utilizar los datos de AEMET en sus propios sistemas de información. La interacción en este tipo de acceso se caracteriza por la posibilidad de hacerla periódica y programada.

2.3.3 Puertos del Estado

Dependiente del Ministerio Transportes, Movilidad y Agenda Urbana de España, Puertos del Estado es un ente público empresarial cuyas responsabilidades recaen sobre el conjunto del sistema portuario estatal. Se encarga de la ejecución de la política portuaria establecida por el Gobierno y de controlar y coordinar la eficiencia del sistema portuario, formado por 46 puertos de interés general administrados por 28 Autoridades Portuarias. Para contextualizar la importancia que cobran los puertos en los sistemas de cadenas logísticas y de transporte, por ellos pasan cerca del 60% de las exportaciones y el 85% de las importaciones, esto representa el 53% del comercio exterior de España con la Unión Europea y el 96% con terceros países [9].

Dentro de sus funciones se encuentran:

- La ya mencionada ejecución de la política portuaria del Gobierno y control de la eficiencia del sistema portuario estatal.
- Asegurar que se establezcan controles en espacios portuarios en coordinación con los diferentes órganos de la Administración General del Estado.
- Formar y promocionar la investigación y desarrollo tecnológico en relación a las actividades portuarias.
- A través de la Comisión de Faros, realizar la planificación, coordinación y control del sistema de señalización marítima español.

La ventaja que nos ofrece esta página a diferencia del resto es que, aparte de contar con boyas como es el caso de MeteoGalicia, cuenta con los denominados puntos SIMAR los cuales consisten en un conjunto de datos modelados y se encuentran mejor repartidos y en mayor número. No son tan fiables como las boyas, pero aun así arrojan buenas aproximaciones.

Figura 2-5 Logo Puertos del Estado [10]

2.3.4 MeteoGalicia

Fruto de un convenio entre la Universidad de Santiago de Compostela y la Junta de Galicia, MeteoGalicia, también conocida como Unidad de Observación y Predicción Meteorológica de Galicia

[1]

fue creada en el año 2000 y a diferencia de Puertos del Estado, esta depende de la Consejería de Medio Ambiente, Territorio e Infraestructuras. Su cometido principal es la de efectuar una predicción meteorológica de Galicia, así como la explotación y mantenimiento de la red de observación meteorológica y climatológica de la Junta de Galicia. Con objeto de mejorar y optimizar el funcionamiento de MeteoGalicia, dicha entidad se encuentra estructurada en cuatro áreas principales: visualización, predicción operativa, predicción numérica y climatología. La alta calidad de los modelos meteorológicos, oceanográficos y de oleaje de alta resolución que ofrece MeteoGalicia son el fundamento de mejora de sus predicciones meteorológicas [11].

En lo que a este trabajo se refiere, esta es la fuente de la que más información se puede extraer ya que aparte de contar con estaciones costeras que proporcionan información acerca de temperatura, viento, precipitaciones, etc. también proporciona datos tomados desde boyas. Dispone de una interfaz muy intuitiva desde la cual se pueden realizar consultas configurándolas con los parámetros y fechas deseadas, una vez obtenido los resultados permite la descarga de los mismos en distintos formatos de archivo.

Figura 2-6 Logo de MeteoGalicia [12]

2.4 Semántica y bases de datos

En lo referente a datos, la semántica se define como un modelo conceptual que tiene la capacidad de mostrar información para interpretar el significado de las instancias sin necesidad de conocer su verdadera representación. La principal característica que plantea este modelo es el poder definir el significado de los datos en contexto de sus relaciones con otros. A mayor número de datos más le cuesta al ser humano leerlos, de ahí nace la necesidad de establecer un método que facilite dicha tarea por lo que surge Notation3[2] como formato de lectura más amigable. El número 3 refiere a las tripletas propias del formato conocido como RDF, el cual se basa en realizar sentencias acerca de recursos en la forma de expresiones sujeto-predicado-objeto. RDF necesita de un lenguaje propio a la hora de realizar consultas, de aquí surge SPARQL. SPARQL es un lenguaje de consultas RDF que además de poder manipular datos en bases semánticas, también tiene la capacidad de construir consultas a múltiples recursos.

2.4.1 Web Semántica

Tim Berners-Lee, James Hendler y Ora Lassila son conocidos como los fundadores de la web semántica tras presentar, en el año 2001, las bases de la misma en la revista Scientific American[3]. Surgió un cambio en el que se pasó de la clásica Web de Documentos a una Web Programable, esto derivó en una gran oferta de aplicaciones que ejecutan tareas de manipulación de datos publicados en la web. El objetivo buscado a la hora de agregarles una semántica a los datos es la de facilitar su comprensión e interoperabilidad en un sistema lleno de información heterogénea publicada en distintos formatos y con diferentes protocolos de acceso como es la web.

[2] https://www.w3.org/TeamSubmission/n3/
[3] https://www.scientificamerican.com/espanol/

Podría decirse que la Web Semántica viene definida por tres bloques básicos: un modelo de datos estándar, un conjunto de vocabularios de referencia y un protocolo estándar de consulta. La Web Semántica busca facilitar la comunicación entre todos los miembros participantes del ecosistema creando un modelo mental común, minimizando al máximo la existencia de ambigüedades y, de esta forma, facilitar el trabajo a las distintas aplicaciones en el uso de las diversas fuentes de datos.

2.4.2 RDF

La Web ofrece una infraestructura simple y universal en la que se realizan cantidades ingentes de intercambios de datos entre los usuarios que la componen. Pese a ello, las aplicaciones encuentran grandes dificultades a la hora de procesar la información contenida en los documentos si previamente no se les ha agregado algún tipo de información específica. Se decidió emplear metadatos para describir los datos publicados en la Web, que sean de utilidad a la hora de localizar o procesar la información, ya sea especificando la estructura que poseen, el contenido o cualquier tipo de información que le pueda resultar útil a la aplicación a la hora de entender el documento. Para realizar este cometido es necesario disponer de una base de datos que sea capaz de procesar dichos metadatos con el objetivo de permitir la interoperabilidad de esas descripciones entre diferentes aplicaciones, respaldadas en estándares sobre la sintaxis y la semántica de los metadatos, además de un conjunto de vocabularios comunes y modos de acceso estandarizados [13].

El RDF es una estructura empleada para la representación de datos en la Web. Permite realizar afirmaciones sobre recursos. Entiéndase por recurso cualquier cosa, puede ser un objeto físico, una idea abstracta o incluso un equipo de fútbol. Cualquier cosa de la que se pueda decir algo al respecto es considerada un recurso. Estas afirmaciones se denominan “statements” y constan de tres elementos estructurados de la siguiente manera: <sujeto> <predicado> <objeto>. A esta estructura se la conoce como tripleta. En ella, tanto el sujeto como el objeto representan a los ya mencionados recursos mientras que el predicado se encarga de denominar la relación existente entre ellos. En el ejemplo mostrado en la Figura 2-7 se observa como un sujeto puede estar relacionado con más de un objeto a través de distintos predicados, en este caso vemos como el libro es el sujeto, el título y el autor son los diferentes predicados y “El Coronel no tiene quien le escriba” y Gabriel García Márquez son en este caso los objetos. Añadir que, a su vez, un objeto puede ser sujeto de otra tripleta, en el caso del ejemplo Gabriel García Márquez a su vez es sujeto, tiene como predicados una nacionalidad y un año de nacimiento y como objetos “Colombiana” y “1927”.

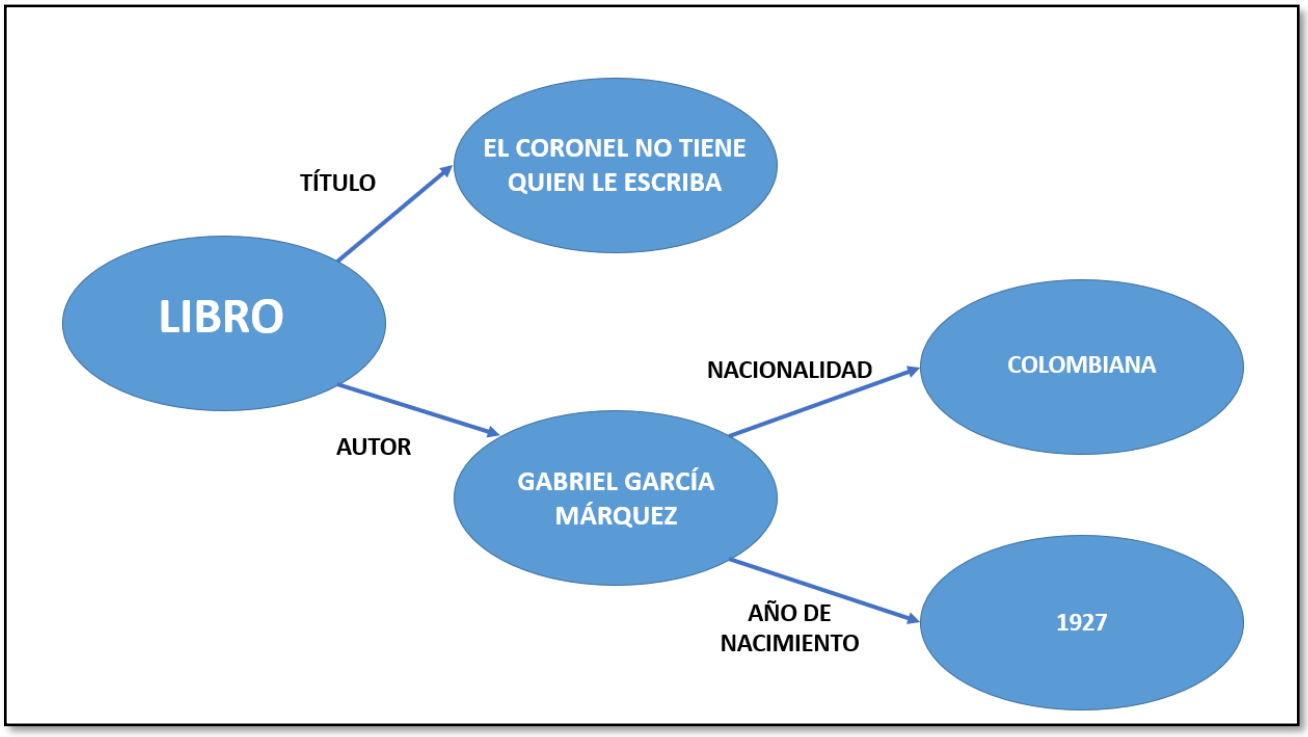


Figura 2-7 Ejemplo de tripleta RDF

2.4.3 SPARQL

La Web Semántica emplea como lenguaje de consulta el SPARQL[4]. Este lenguaje permite al usuario consultar información de bases de datos o de cualquier fuente de datos que pueda ser mapeada a RDF. SPARQL fue diseñado y es avalado por el W3C para ayudar a los usuarios y desarrolladores a focalizarse en el qué es lo que quieren saber en lugar de conocer el cómo está organizada la base de datos.

SPARQL es comparado constantemente con su homónimo el SQL [5]y al igual que este, SPARQL permite a los usuarios obtener y modificar datos en una base relacional. Además, una consulta SPARQL puede ejecutarse en cualquier tipo de base de datos que pueda ser visualizada como RDF mediante un software intermediario. A diferencia de SQL, las consultas SPARQL no se limitan a trabajar dentro de una única base de datos: las consultas pueden acceder a múltiples almacenes de datos (endpoints). Esto es técnicamente posible porque SPARQL es más que un simple lenguaje de consulta. También es un protocolo de transporte basado en HTTP, en el que se puede acceder a cualquier punto final de SPARQL a través de una capa de transporte estandarizada. Los resultados de RDF pueden devolverse en varios formatos y las entidades RDF se identifican mediante URIs (Universal Resource Identifiers). La identificación de los datos con URIs permite que los datos sean referenciados de forma inequívoca en todas las aplicaciones y supera las limitaciones que plantea la búsqueda local. En consecuencia, se pueden desarrollar APIs específicas para aplicaciones adicionales que pueden hacer referencia a esos datos. SPARQL está diseñado para hacer posibles los enlaces entre datos en la Web Semántica. Su objetivo es enriquecer los datos vinculándolos a otros recursos semánticos, para así compartir, fusionar y reutilizar los datos de forma más significativa. Como resultado, el potencial de SPARQL junto con la flexibilidad de RDF puede reducir los costes de desarrollo al facilitar la fusión de resultados de múltiples fuentes de datos [14].

[4] https://skos.um.es/TR/rdf-sparql-query/

[5] https://support.microsoft.com/es-es/office/access-sql-conceptos-b%C3%A1sicos-vocabulario-y-sintaxis-444d0303-cde1-424e-9a74-e8dc3e460671

2.4.4 Bases de datos no estructuradas

La ubicuidad de las tecnologías de la información ha dado lugar a una mayor complejidad y diversidad en los datos que se producen y se procesan. Las tecnologías asociadas a la Web Semántica surgen como alternativa para atender ese aumento de esa complejidad y de la necesidad de trabajar con fuentes heterogéneas. Lenguajes como RDF[6] y OWL[7] son lenguajes base de lo que se conoce como tecnologías semánticas, y cuyo objetivo es representar, integrar y razonar sobre datos de diversos tipos.

Así, un documento RDF no es más que un grafo, representado mediante tripletas, esto es, estructuras que relacionan entre sí mediante un arco un nodo sujeto con otro nodo de tipo objeto. En este sentido, el arco representa el predicado que une dichos nodos. Los grafos RDF pueden almacenarse ya sea en archivos de texto plano, o en bases de datos de tripletas, las cuales proveen funcionalidades similares a los motores de bases de datos tradicionales Estos sistemas también se llaman *triplestores*. Estas bases de datos, están especialmente diseñadas para el almacenamiento y recuperación de triples a través de consultas semánticas. Estas consultas se realizan generalmente mediante el lenguaje SPARQL, lenguaje de consultas nativo para este formato. Sin embargo, en la práctica, la gran mayoría de los datos no se almacenan usando representaciones directamente accesibles por tecnologías semánticas.

Para poder hacer uso de datos que no poseen una estructura directamente accesible por tecnologías semánticas se necesita traducir esos datos a una estructura compatible con la Web semántica. A este proceso se le conoce como Acceso a Datos basados en Ontologías (ADBO) (en inglés *Ontology Based Data Access)*. El modo de realizar esa traducción es simplemente reescribiendo las consultas SPARQL al lenguaje de consulta nativo de la fuente de datos original. Es lo que se conoce como *reescritura de consultas*.

Sin embargo, una alternativa para generar un grafo RDF completo correspondiente a todos los datos almacenados en la fuente original es utilizar reglas de mapeo. En la práctica, las bases de datos relacionales han sido siempre de las alternativas más comunes usadas como medio de almacenamiento. La problemática de realizar este proceso de traducción, ADBO, con información almacenada en este tipo de bases de datos se ha abordado con relativa asiduidad en la literatura e incluso por parte del *World Wide Web Consortium* (W3C) con una recomendación. Así, existen varias herramientas que implementan y extienden las funcionalidades especificadas por el W3C para realizar el proceso de ADBO. Estas son:

- *R2RML*[8]: Este lenguaje define reglas y todos los métodos de mapeo entre tablas de bases de datos relacionales y grafos RDF.
- *Morph-RDB* implementa materialización y reescritura de consultas basándose en la especificación R2RML.
- Ontop implementa OBDA sobre bases de datos relacionales mediante la reescritura de consultas permitiendo realizar inferencias RDFS y OWL-QL.

En los últimos años, debido a la creciente complejidad de los datos, se han hecho esfuerzos por desarrollar nuevas formas de almacenamiento que permitan satisfacer las necesidades de flexibilidad y versatilidad en el almacenamiento de datos.

Las bases de datos NoSQL están adquiriendo una gran popularidad debido a la capacidad que poseen de recoger datos de diferente naturaleza. Dentro de las distintas opciones, MongoDB se ha posicionado como líder en este segmento siendo parte de la infraestructura de importantes empresas de base tecnológica. En MongoDB, los registros se almacenan en documentos JSON, los cuales a su vez pueden tratarse de documentos anidados. Esto último junto con la capacidad que tiene de manejar grandes volúmenes de datos y la flexibilidad que dispone al no depender de un esquema, son las características

[6] RDF: Resource Description Framework
[7] OWL: Ontology Web Language
[8] https://www.w3.org/TR/r2rml/

centrales de MongoDB, las cuales han dado lugar a su alto grado de adopción dentro de las bases de datos no relacionales.

Estas características están en concordancia con alguno de los propósitos de la Web Semántica, dentro del cual se plantea que no se puede anticipar un esquema debido a la diversidad y cambios de los datos que se producen constantemente. Debido a este parecido y al aumento de su uso, emplear una herramienta que realice ADBO sobre bases de datos MongoDB es una opción realmente interesante. En este sentido existen actualmente las siguientes herramientas disponibles:

- MorphxR2RML, una extensión del ya mencionado Morph-RDB.
- MongoGraph, un componente dentro de la triplestore AllegroGraph, que permite la interoperabilidad con MongoDB.
- Tripod: es un ORM que permite la utilización de MongoDB como capa de almacenamiento para grafos RDF.

Los datos estructurados son aquellos que se encuentran interrelacionados unos con otros, en su mayoría se clasifican como datos cuantitativos y son los que podemos encontrar organizados en filas y columnas con títulos. Debido a estas relaciones establecidas entre unos y otros son muy fácilmente comprendidos por el lenguaje de máquina y los datos se pueden manipular, buscar e ingresar con mayor rapidez. El lenguaje de programación que emplean es el SQL desarrollado por IBM en la década de los 70. Esta manera de almacenar los datos resultó un gran cambio para instituciones, empresas y organizaciones quienes, de forma tradicional, lo hacían en papel [15] [16].

Los datos no estructurados se encuentran en bruto sin organizarse de ninguna manera, están dispuestos de tal forma que el lenguaje de máquina no los entiende. Hay cierta información que no necesitamos que el sistema entienda, simplemente que sea capaz de almacenarlo y de mostrarlo. Por ejemplo, el caso de nombres de personas y números de DNI, esto podría estructurarse de forma que a cada persona le corresponda su número de DNI, podríamos acceder a ello de forma rápida y sencilla. Sin embargo, hay ciertos datos que es difícil clasificarlos de alguna manera como es el ejemplo de los correos electrónicos, se pueden clasificar según el remitente o el asunto, pero el contenido del correo es muy complejo de clasificar ya que puede ser desde contenido multimedia hasta un informe. Aquí es donde entran a jugar las bases de datos no estructuradas donde lo que nos interesa es simplemente almacenar la información y no tenemos la necesidad de que la máquina lo entienda o ni que nos lo clasifique de ningún modo. Más del 80% de los datos que encontramos hoy en día son no estructurados [15] [16].

El factor clave por el que se escoge la base de datos no estructurada frente a la estructurada es la ausencia de restricciones. La base de datos estructurada se ve muy condicionada debido a que tendríamos que clasificar cada uno de los datos y darle una relación directa con respecto al resto. Los datos no estructurados sin embargo son mucho más flexibles ya que disponen de esa libertad a depender de un esquema predefinido.

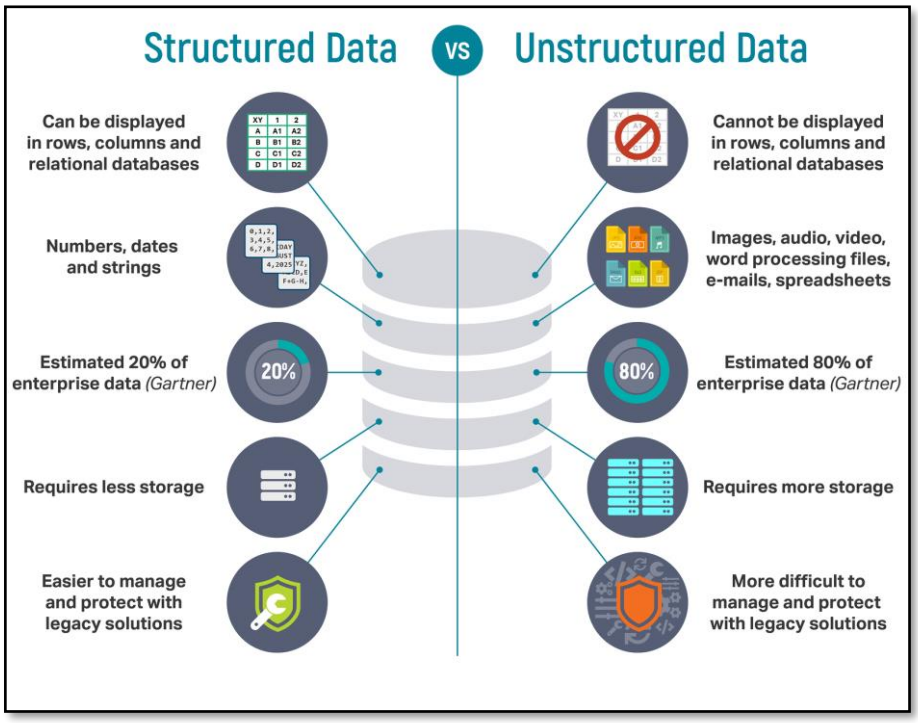


Figura 2-8 Esquema de los dos tipos de bases de datos [17]

2.4.4.1 MongoDB con RDF

Dentro de los diferentes modelos de bases de datos que ofrece MongoDB, el que nos interesa para este proyecto es el MongoDB con RDF, es decir, una base de datos orientada a grafos.

Las bases de datos de grafos satisfacen una necesidad que las bases de datos tradicionales han dejado sin cubrir: dan prioridad a las relaciones entre los elementos. Esto hace que las bases de datos de grafos sean increíblemente eficientes para buscar patrones, hacer predicciones y encontrar soluciones. Aunque es una base de datos de documentos de propósito general, MongoDB proporciona capacidades de recorrido de grafos y árboles a través de su herramienta $graphLookup. Para las aplicaciones que requieren mayor capacidad de grafos o que utilizan grafos con frecuencia, MongoDB también puede acoplarse con una base de datos gráfica dedicada [18].

Para entenderlo mejor vamos a empezar por definir lo que es un base de datos orientada a grafos. Se refiere a la base de datos que almacena los datos en nodos y aristas: los nodos para la información sobre una entidad y las aristas para la información sobre la relación o las acciones entre los nodos. Este modelo de datos permite que las bases de datos de grafos den prioridad a las relaciones entre los datos, lo que facilita la búsqueda de conexiones y patrones en los datos en comparación con las bases de datos tradicionales [18].

Las bases de datos orientadas a grafos utilizan la siguiente terminología:

- Nodos: almacenan la información en sí de las cosas.
- Aristas: definen las acciones que se dan entre los distintos nodos.
- Propiedades: clave-valor que almacena información acerca de un nodo o arista concreta.
- Etiquetas: de forma opcional se pueden utilizar para nombrar un conjunto de nodos relacionados.

Veamos cómo se almacena la información en una base de datos orientada a grafos. Supongamos que queremos almacenar información acerca de trayectos de tren. En este caso cada nodo representaría una estación de tren y la información de cada estación se almacenaría en las propiedades del nodo. Por

ejemplo, ponemos el caso de la Estación de Chamartín y la Estación de Pontevedra, cada una estaría representada por un nodo cuyas propiedades podrían ser la provincia donde se encuentran, el número de andenes del que disponen, la abreviatura de la estación, etc. Luego dichos nodos estarían unidos mediante una arista que en este caso sería el trayecto que une ambas estaciones y como propiedades tendría el número de tren y si se trata de un ALVIA o un AVE [18].

Figura 2-9 Ejemplo MongoGraph con estaciones de tren

Mongograph funciona a través de Cloud por lo que pese a ser una gran herramienta que se ajusta a la mayoría de las necesidades de este proyecto se ha tenido que descartar pues trabajaremos con los datos de forma local.

3 DESARROLLO DEL TFG

3.1 Arquitectura

La idea es la de poder cargar la base de datos que se va a generar en un gemelo digital de forma que pudiese generar de forma simulada el estado de las rías en las que navegamos con las lanchas de instrucción y así poder ver de qué forma nos afectaría a la hora de realizar nuestros ejercicios. En la Escuela Naval Militar antes de realizar navegaciones de tipo nocturna o pernocta se realiza un *briefing* en el cual siempre se incluye una matriz de impacto en la cual de forma muy visual se determina si con las condiciones meteorológicas existentes será o no factible realizar los distintos ejercicios programados. La aplicación de la base de datos sobre un gemelo digital oceanográfico facilitaría enormemente la confección de dicha matriz de impacto e incluso podría determinar en qué otra zona de la ría se podrían hacer los ejercicios u otras cuestiones como cuando sería la hora óptima para cruzar de una ría a otra.

En este apartado se explicará, a grandes rasgos, la arquitectura que compone este proyecto definiendo los elementos de los que dispone, así como la relación existente entre ellos y las funciones que desempeñan.

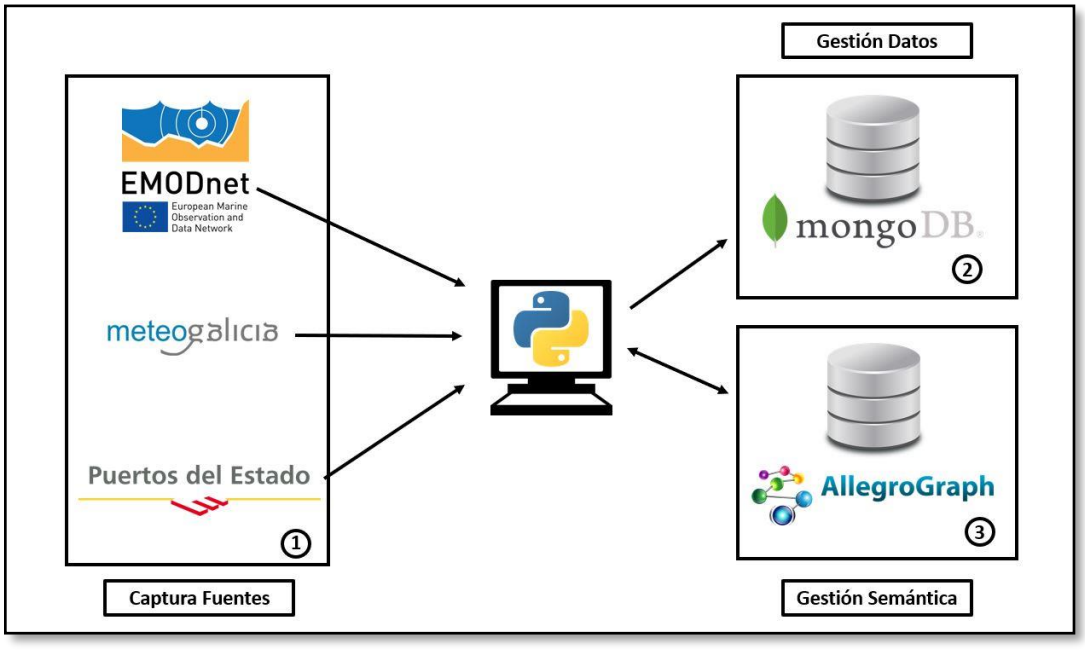


Figura 3-1 Esquema de la arquitectura del proyecto

Como se puede observar en la Figura 3-1, por una parte, se encuentran las fuentes de datos como pueden ser EMODnet, MeteoGalicia y Puertos del Estado, por otro lado, la base de datos MongoDB que almacenará la información extraída de esas fuentes y finalmente el servidor AllegroGraph.

El funcionamiento es el siguiente: la información es extraída de las diferentes fuentes ya sea en archivos con formato JSON, CSV o XML; a continuación, se procesa automáticamente su contenido y la información considerada de relevancia para el sistema se almacenan en la base de datos MongoDB. Al tiempo que se almacena en esa base de datos, se almacena la estructura semántica en el servidor AllegroGraph. En él se guardarán las referencias a los elementos de la colección de MongoDB a través de su identificador y servirá para establecer las relaciones entre los diversos conceptos existentes. Esta interacción se realiza a través de la inclusión de tripletas especiales o tripletas de enlace, dentro del repositorio. Por último, una vez la información almacenada en la base de datos NoSQL y guardada la referencia a sus elementos en el servidor AllegroGraph, junto con las relaciones existentes entre elementos, ya se puede pensar en realizar consultas semánticas sobre la información almacenada procedente de las fuentes externas. Así, por ejemplo, para un estudio acerca del clima oceánico que hay en Galicia se pordría realizar la consulta de en qué zonas se llegó a una determinada temperatura y el servido nos mostraría todas las estaciones que tuviesen registradas esas temperaturas y en qué fecha y hora fue registrada.

3.2 Herramientas utilizadas

3.2.1 El lenguaje de programación Python

A la hora de crear un programa se emplean los denominados lenguajes de programación, estos se encargan de establecer un entendimiento entre el desarrollador y la máquina. En este aspecto existe una amplia variedad de opciones. En el caso de este trabajo, debido a su gran popularidad y sencillez a la hora de resolver cuestiones, se ha optado por emplear el lenguaje de programación Python[9].

Se trata de un lenguaje de programación de propósito general. Desarrollado en el Centrum Wiskunde & Informatica se caracteriza, entre otras cosas, por la influencia que tienen en él otros lenguajes como C. A este lenguaje se le puede clasificar como multiparadigma imperativo y funcional, esto viene a decir que excepto en las estructuras condicionales y bucles, sigue unas instrucciones de forma secuencial y que, a la hora de diseñar, el programador especifica qué es lo que quiere hacer estableciendo primero las funciones y empleando de variables a las que se les pueden asignar diferentes valores. A la hora de expresar algoritmos tiene en consideración las capacidades cognitivas de los seres humano por encima de la capacidad de las máquinas. Esto lo convierte en un lenguaje de alto nivel capaz de realizar acciones con pocas líneas de código.

Todas las ventajas previamente mencionadas y la similitud que presenta a la hora de trabajar con él a otros lenguajes, han llevado a Python a ser, hoy en día, uno de los lenguajes de programación más utilizados a nivel global. Posicionado como uno de los 5 mejores en el ranking TIOBE[10], se encuentra mejor valorado que otros muy conocidos como Java o C++.

Para ello, antes de comenzar a generar nuestro código en el intérprete Pycharm, hay que instalar la librería “pymongo” de forma que este nos entienda y pueda crear nuestra base de datos en MongoDB. Para instalarlo nos dirigiremos al apartado “file” dentro de la aplicación Pycharm, de ahí a “Settings…”, se abrirá una ventana emergente, desplegamos la pestaña de nuestro proyecto y seleccionamos la opción del intérprete de Python, en la parte de la derecha le damos al “+” (Figura 3-2) y se abrirá un buscador

[9] https://python.org/

[10] El índice TIOBE es un indicador que utiliza distintas variables como el número de personas que lo utilizan o los cursos que existen para determinar la popularidad de los lenguajes de programación.

de paquetes, aquí buscamos e instalamos el paquete “pymongo” y ya tenemos la aplicación lista para crear la base de datos, así como verter la información en esta.

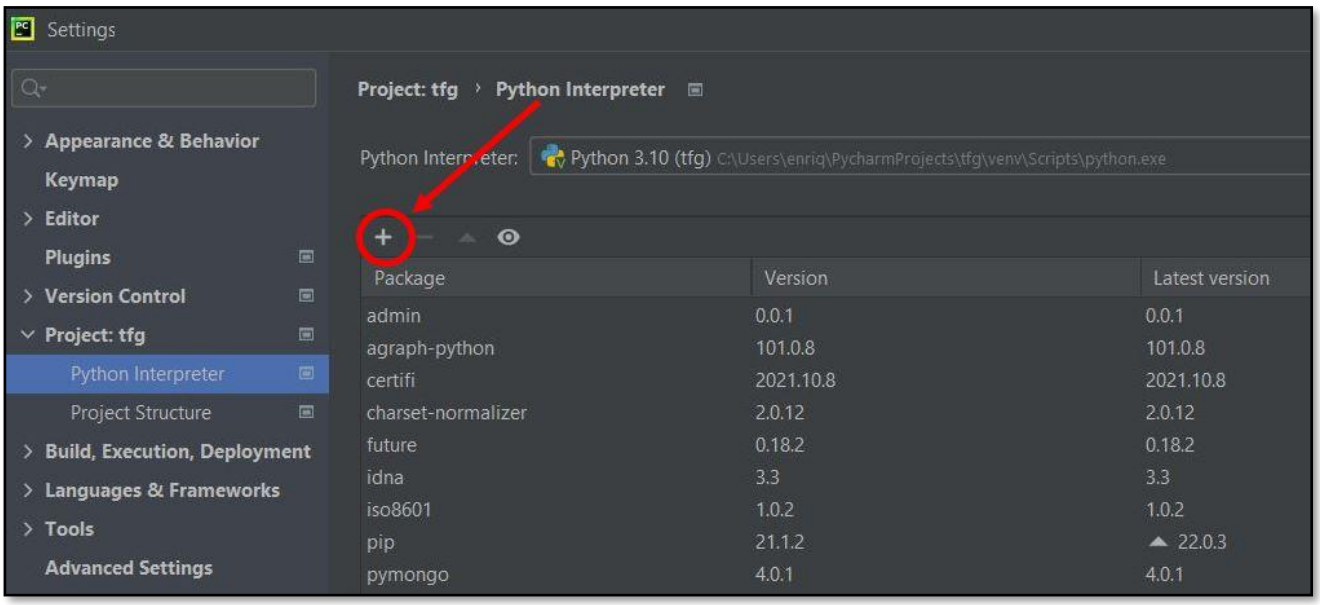


Figura 3-2 Instalación del paquete "pymongo"

3.2.2 El formato JSON

En la sintaxis de objetos de JavaScript, los datos estructurados son representados por el formato basado en texto estándar *JavaScript Object Notation* (JSON). Este formato resulta muy versátil ya que, pese a ser muy parecido a la sintaxis de objeto literal de JavaScript, puede ser empleado fuera de JavaScript, por ello es que muchos entornos de programación poseen la capacidad de leer y generar JSON. Este tipo de archivo resulta de gran utilidad a la hora de transmitir datos a través de una red [19].

Debido a la facilidad que supone a la hora de trabajar con él en lenguaje Python, en este proyecto se empleará el formato de archivo JSON. La Figura 3-3 es un ejemplo de archivo en formato JSON acerca de la meteorología captada por la estación del puerto de Marín el día 3 de marzo de 2022. Como se puede observar están todos los datos juntos unos seguidos de otros lo cual dificulta su entendimiento por lo que para explicarlo se hará uso de la Figura 3-4. Lo primero que aparece es la palabra “resultados” seguida de dos puntos y un corchete a partir del cual empieza la información, esto es así para denominar que todo lo que hay dentro de dichos corchetes son los resultados. Ahora si se analizan los datos se observa que todos comparten una misma estructura. Cada secuencia de datos viene definida entre dos llaves y separando los mismos entre comas, a estas secuencias se las denomina de diferentes formas, en este trabajo se hará referencia a ellas como objetos. En este caso se puede apreciar como cada secuencia consta de un número de validación, una fecha, un código del parámetro medido, el parámetro, el valor medido y las unidades. Los datos vienen representados en pares de nombre/valor, esto se puede apreciar en la Figura 3-4 donde se ve claramente el nombre del dato y a continuación, separado por “:”, el valor del mismo.

Figura 3-3 Archivo en formato JSON

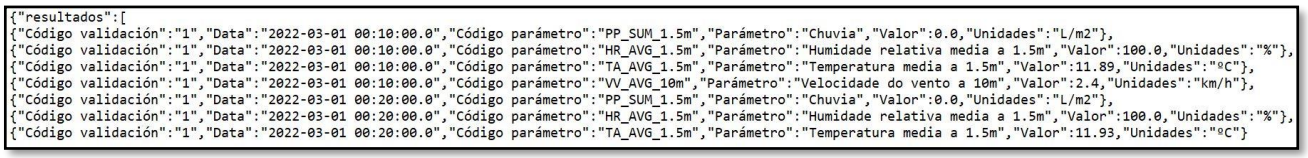


Figura 3-4 Archivo en formato JSON reorganizado

3.2.3 El entorno de desarrollo Pycharm

Debido al éxito que ha tenido Python, se han desarrollado varios entornos de desarrollo. El utilizado en el marco de este TFG ha sido PyCharm[11].

Pycharm es un entorno de desarrollo integrado desarrollado por la empresa checa JetBrains, que se usa para programar en lenguaje Python. Este software, entre otras cosas, ofrece análisis de código, un depurador gráfico, integración con sistemas de control de versiones y, además, soporta desarrollo web con Django, así como la ciencia de datos con Anaconda. Pycharm está diseñado para facilitar al desarrollador la escritura del código, para ello cuenta con una serie de ayudas como un indicador de errores sobre la marcha, inspecciones del código, refactorización de código automática, finalización del código inteligente funcionalidades de navegación.

Hoy en día, PyCharm se encuentra disponible para los tres grandes sistemas operativos, con versiones para Windows, macOS y Linux. Cuenta con una Edición Comunitaria que se publica bajo la Licencia Apache y es totalmente gratuita, también hay una Edición Profesional con características adicionales publicada bajo una licencia propietaria financiada por suscripción y existe una tercera versión educativa.

[11] https://www.jetbrains.com/es-es/pycharm/

3.2.4 MongoDB

MongoDB[12] es un sistema de base de datos NoSQL de código abierto orientado a documentos que se caracteriza por ofrecer un modelo de consultas e indexación avanzado, así como una gran escalabilidad y flexibilidad. Es de código abierto escrito en C++ donde a diferencia de las bases de datos relacionales, que guardan los datos en tablas, guarda estructuras de datos BSON con un esquema dinámico, lo que facilita y acelera la integración de datos en ciertas aplicaciones [20].

BSON se refiere a una configuración de intercambio de datos, una representación binaria de mapas y estructuras de datos utilizada principalmente para su almacenamiento y transferencia en MongoDB. BSON viene de "Binary JSON", y eso es exactamente para lo que fue inventado. La estructura binaria de BSON codifica la longitud y el tipo de información, lo que permite analizarlo mucho más rápidamente. Desde su concepción inicial, BSON se ha ampliado para añadir algunos tipos de datos opcionales no nativos de JSON, como fechas y datos binarios, sin los cuales MongoDB habría echado en falta un valioso soporte. Los lenguajes que soportan cualquier tipo de matemática compleja suelen tener enteros de diferentes tamaños (ints vs longs) o varios niveles de precisión decimal (float, double, decimal128, etc.). No sólo es útil poder representar esas distinciones en los datos almacenados en MongoDB, sino que también permite que las comparaciones y los cálculos se realicen directamente sobre los datos de forma que se simplifique el consumo de código de la aplicación [21].

Figura 3-5 Logo de MongoDB [22]

Pese a que MongoDB esté escrito en C++, las consultas se hacen pasando objetos JSON como parámetro ya que los propios documentos se almacenan en BSON. MongoDB viene de serie con una consola en la que se pueden definir variables, funciones o utilizar bucles. Esta está construida en JavaScript lo que permite el empleo de este lenguaje a la hora de ejecutar comandos y permite hacer uso de muchas de las funciones de JavaScript [23].

En caso de que se desee utilizar otro tipo de lenguajes de programación, MongoDB los admite a través de una serie de drivers. Existen drivers oficiales de C#, Node.js, Java, PHP, Ruby, Python, C, C++, Perl y Scala. Bien es cierto que a la hora de escoger el lenguaje que se va a utilizar hay que tener en cuenta que no todos los drivers se encuentran en el mismo estado de madurez, hay unos que se encuentran en un estado de desarrollo superior a otros, por ejemplo, el driver de C se encuentra en una versión Alpha. Alpha se refiere a la fase del lanzamiento en la que se encuentra un software, es la primera versión completa y por lo general todavía sigue siendo bastante inestable [23].

La característica principal que define a MongoDB es su velocidad la cual, gracias a su sistema de consulta de contenidos, consigue un balance perfecto entre rendimiento y funcionalidad. A parte de su velocidad cuenta con otras características que la hacen la favorita de muchos desarrolladores. Entre ellas se encuentran [24]:

[12] https://www.mongodb.com/es

- Indexación: la estructura de su índice es similar a la de las bases de datos estructuradas, pero a diferencia de estas, cualquier campo específico puede ser indexado y añadir múltiples índices secundarios.
- Replicación: MongoDB cuenta con el tipo de replicación primario-secundario. Esto significa que mientras se realizan consultas con el primario, el secundario actúa de forma paralela como réplica de datos a modo de copia de seguridad y permite elegir un nuevo primario en caso de que el actual dejase de funcionar.
- Consultas ad hoc: permite realizar búsquedas por campos, consultas de rangos y expresiones regulares.
- Almacenamiento de archivos: dispone de una funcionalidad llamada GridFS que permite manipular archivos y contenido por lo que MongoDB también puede ser utilizado como un sistema de archivos.
- Balanceo de carga: MongoDB tiene la capacidad de escalar la carga de trabajo, se puede ejecutar de forma simultánea en varios servidores por lo que en caso de fallo de hardware se puede mantener el sistema funcionando.
- Ejecución de JavaScript del lado del servidor: se pueden hacer las consultas empleando JavaScript y de esta manera que se envíen directamente a la base de datos para ser ejecutadas.

3.2.5 AllegroGraph

Finalmente se ha optado por utilizar la herramienta AllegroGraph[13] la cual nos permite utilizar los archivos de forma local y se puede combinar con MongoDB. Para ello vamos a empezar por definir qué es AllegroGraph. AllegroGraph es una tecnología de grafos de conocimiento de entidades y eventos distribuidos horizontalmente y multimodelos (documentos y grafos) que permite a las empresas extraer información para la toma de decisiones y para el análisis predictivo de sus datos complejamente distribuidos que no pueden responderse con las bases de datos convencionales.

FedShard acelera las consultas complejas a través de una función patentada de federación de memoria, los resultados de cada máquina se combinan para que el proceso de consulta parezca que se está accediendo a una sola base de datos, aunque en realidad se está accediendo a muchas, a almacenes de datos y a bases de conocimiento diferentes y devolviendo resultados. Esta capacidad única de federación de datos acelera los resultados de consultas muy complejas en conjuntos de datos y bases de conocimiento muy distribuido.

A diferencia de las bases de datos relacionales tradicionales o de las bases de datos de grafos de propiedades simples, el producto AllegroGraph de Franz emplea una combinación de tecnologías de archivos (JSON y JSON-LD) y de grafos que procesan los datos con inteligencia contextual y conceptual. Los grafos de conocimiento construidos en la plataforma AllegroGraph son capaces de ejecutar consultas de una complejidad sin precedentes para apoyar el análisis predictivo que ayuda a las empresas a tomar mejores decisiones en tiempo real.

3.3 Metodología

3.3.1 Captura de Datos

3.3.1.1 Meteogalicia

Parte de los datos con los que vamos a trabajar en este proyecto los obtendremos tanto de estaciones meteorológicas costeras como de boyas a través del portal de MeteoGalicia. En primer lugar, accedemos

[13] https://allegrograph.com/

a su página web[14] y tras pulsar en el menú “Observación”, en la columna de la derecha se selecciona “Acceso a datos”. Ahí, nos encontramos una lista con todas las estaciones meteorológicas ordenadas (Figura 3-6) según su localización. Además, se puede configurar nuestra consulta, como, por ejemplo, indicando la fecha de la que queremos obtener datos, si queremos descargar datos con resultados mensuales, diarios o de cada 10 minutos y qué información meteorológica deseamos consultar. Cabe mencionar que, a la hora de realizar la consulta de datos para su posterior descarga, MeteoGalicia establece tres restricciones referentes a las fechas de las consultas: la primera es que, si solo se escoge un parámetro, permite consultar los datos de los últimos 12 meses. La segunda es que, si escogemos hasta tres parámetros, se puede consultar la información de los último 4. En cualquier otro caso se puede consultar como mucho la información referente al último mes. Una vez seleccionada nuestra configuración realizamos la consulta, se nos abrirá una página con todos los datos. Tendremos la opción de descargarnos dichos datos en archivos de distinto formato, más concretamente en formato JSON, PDF y CSV.

A la hora de introducir la información en nuestra base de datos podríamos subir directamente el archivo sin ningún tipo de modificaciones, sin embargo, es necesario establecer relaciones con esos objetos y los enlaces a los conceptos en el servidor AllegroGraph. De no ser así, esto dificultaría enormemente la posibilidad de realizar consultas SPARQL a través de él. Es por ello que la información requiere de un proceso previo para su inserción.

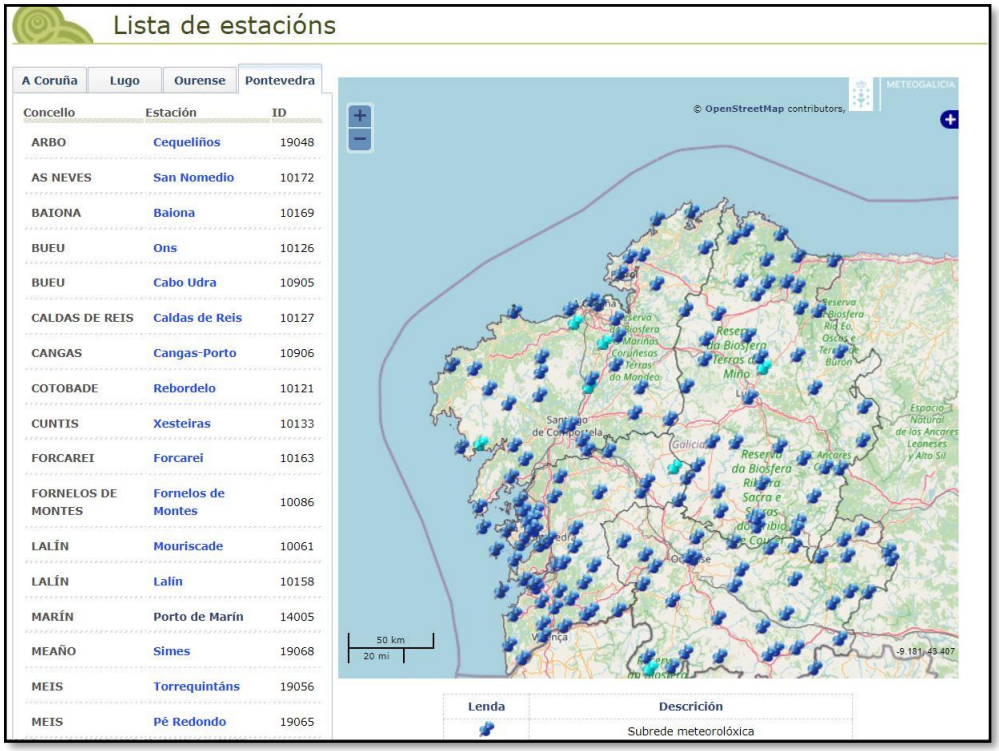


Figura 3-6 Estaciones MeteoGalicica [25]

La primera cuestión que hay que plantear es la de qué archivos se van a emplear en este proyecto. A la hora de escoger los datos se han tenido en cuenta aquellos que se consideran más relevantes para la navegación es por ello que para este trabajo se ha decidido descargar datos de: temperatura, humedad relativa, dirección y componente del viento, presión y precipitaciones (Figura 3-7). En cuanto a la localización, se trabajará con información de la ría de Pontevedra y de la de Vigo por lo que se han tomado datos de 6 estaciones distintas, tres de ellas referentes a la ría de Pontevedra y otra tres a la de Vigo. Concretando, se trata de las siguientes estaciones: la estación del Puerto de Marín, la estación de

[14] https://www.meteogalicia.gal/web/inicio.action

Sanxenxo, la estación de Cabo Udra, la estación del Puerto de Vigo, la estación de las Islas Cíes y la estación de Bayona. Cabe decir que tanto las estaciones de Cabo Udra, Bayona y de las Islas Cíes, no proporcionan información acerca de la presión. Pueden verse todas las estaciones empleadas rodeadas en rojo en la Figura 3-8.

El abanico de boyas del que dispone MeteoGalicia no es muy amplio y no está tan orientado a las zonas que va a tratar el presente trabajo. Es por ello que de esta fuente extraeremos información de una sola boya, concretamente la boya situada en las Islas Cíes la cual se complementará con la estación que se encuentra allí situada para ofrecer la información meteorológica referente a la ría de Vigo, además aportará datos acerca de la temperatura del agua en esa zona. Los pasos a seguir para la descarga de datos son muy similares a los de las estaciones, nos dirigiremos al apartado donde se encuentran las boyas en la página web de MeteoGalicia. Una vez allí, y al igual que con las estaciones, nos permitirá escoger un intervalo de tiempo y unos parámetros para realizar una consulta. En este caso se marcan las casillas de los siguientes parámetros: temperatura del agua, humedad relativa, temperatura y componente e intensidad del viento.

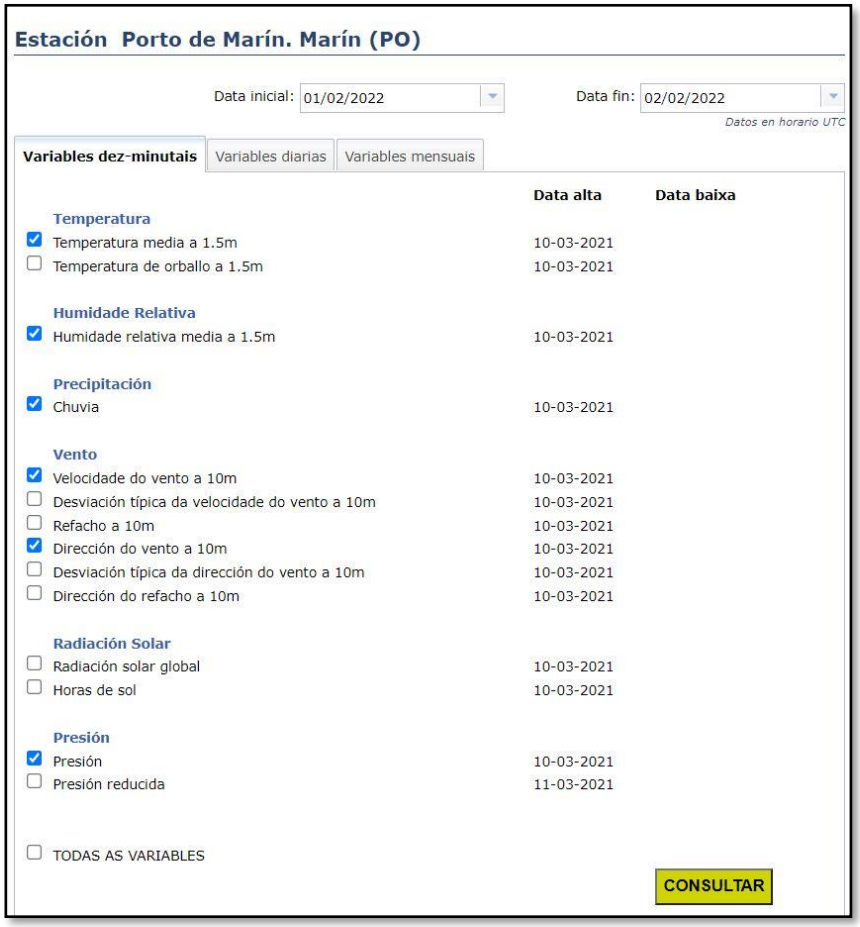


Figura 3-7 Configuración para la descarga de datos de MeteoGalicia

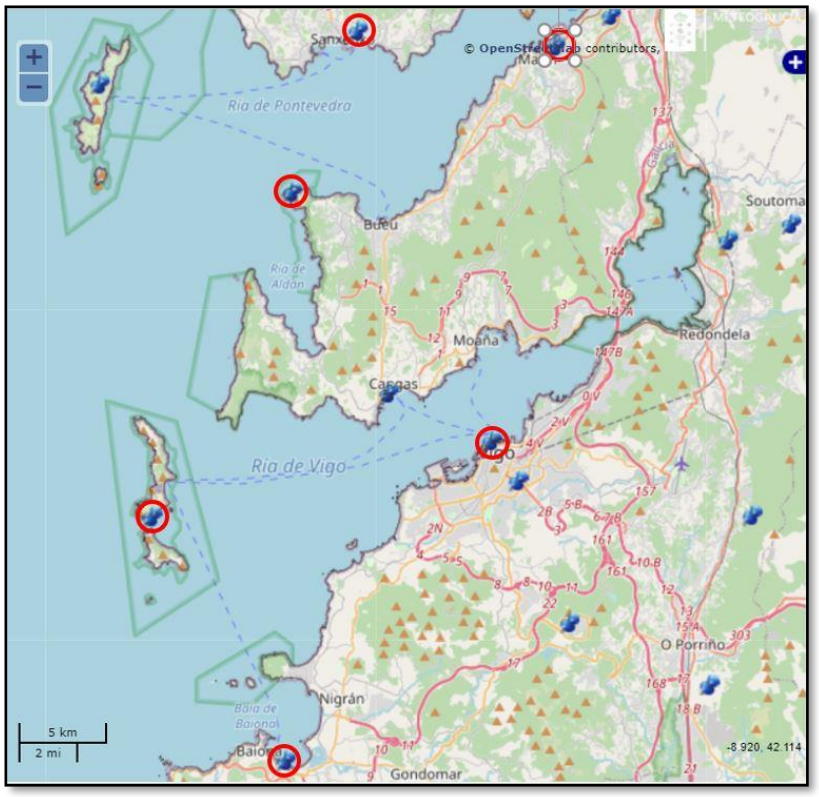


Figura 3-8 Estaciones empleadas de MeteoGalicia

3.3.1.2 Puertos del Estado

La segunda web de la cual extraeremos los datos para la realización de este trabajo es la de Puertos del Estado[15], la cual nos aportará los datos históricos acerca del oleaje mediante puntos SIMAR. Como ya se mencionó en el estado del arte, los puntos SIMAR son conjuntos de datos modelados que se encuentran mejor repartidos y en mayor número. Para acceder a dichos puntos hay que dirigirse a la web oficial de Puertos del Estado, una vez dentro, hay que seleccionar en el menú izquierdo la opción "Oceanografía". Esto nos dirigirá a otra página en la que se encuentra un mapa satélite de la península ibérica y a su derecha unos parámetros para configurar la búsqueda (Figura 3-9). Basta con configurar la búsqueda, seleccionar el punto que nos interesa y a continuación se nos abrirá una ventana en la que se nos permite incluir la información escogida en nuestras descargas. La desventaja más clara que presenta frente a MeteoGalicia es que mientras en esta última se escogen los datos y se descargan de forma directa, en Puertos del Estado se realiza una petición para la descarga de datos la cual contestan en 24h - 48h mediante un email.

[15] https://www.puertos.es

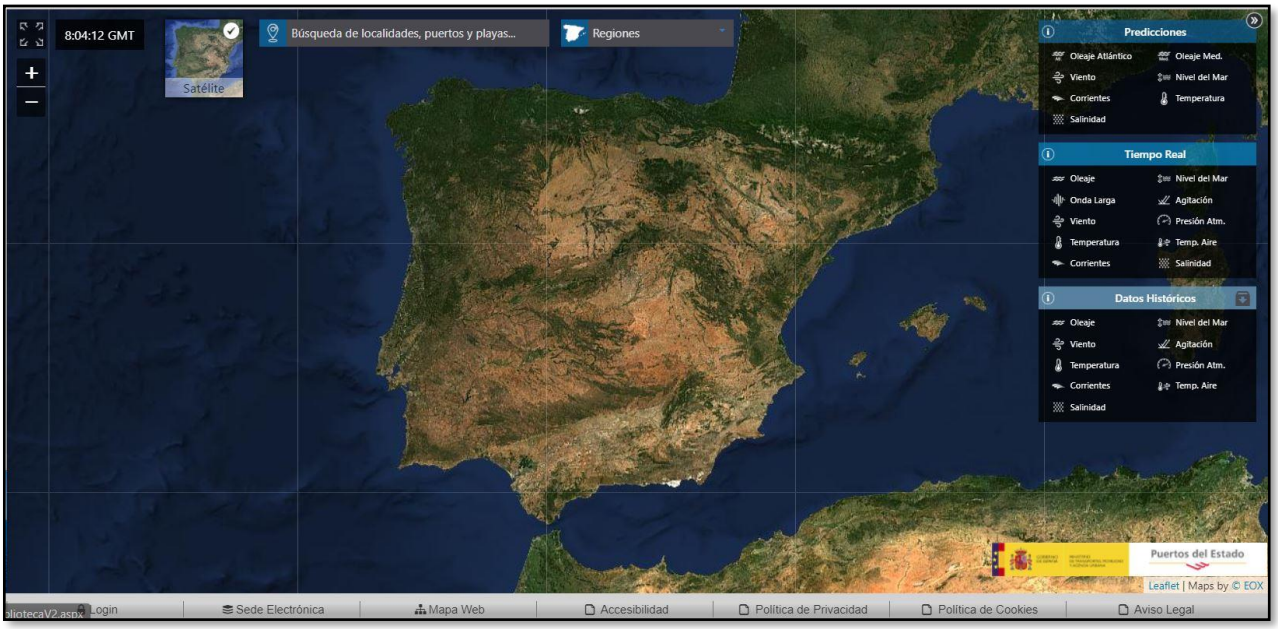


Figura 3-9 Página web de Puertos del Estado

Como ya se ha mencionado previamente, este trabajo centra sus esfuerzos en las rías de Pontevedra y Vigo por lo que se solicitará una descarga de todos los puntos SIMAR de la zona. Para realizar la solicitud de descarga de datos hay que dirigirse al menú de "Datos Históricos" situado a la derecha de la página, a continuación, se ha de marcar la casilla de la variable que interese que en nuestro caso es la del oleaje. Una vez realizados estos dos sencillos pasos se nos mostrarán todas las boyas y puntos SIMAR de la zona, presentándose estos últimos en verde y las boyas en rojo (Figura 3-10).

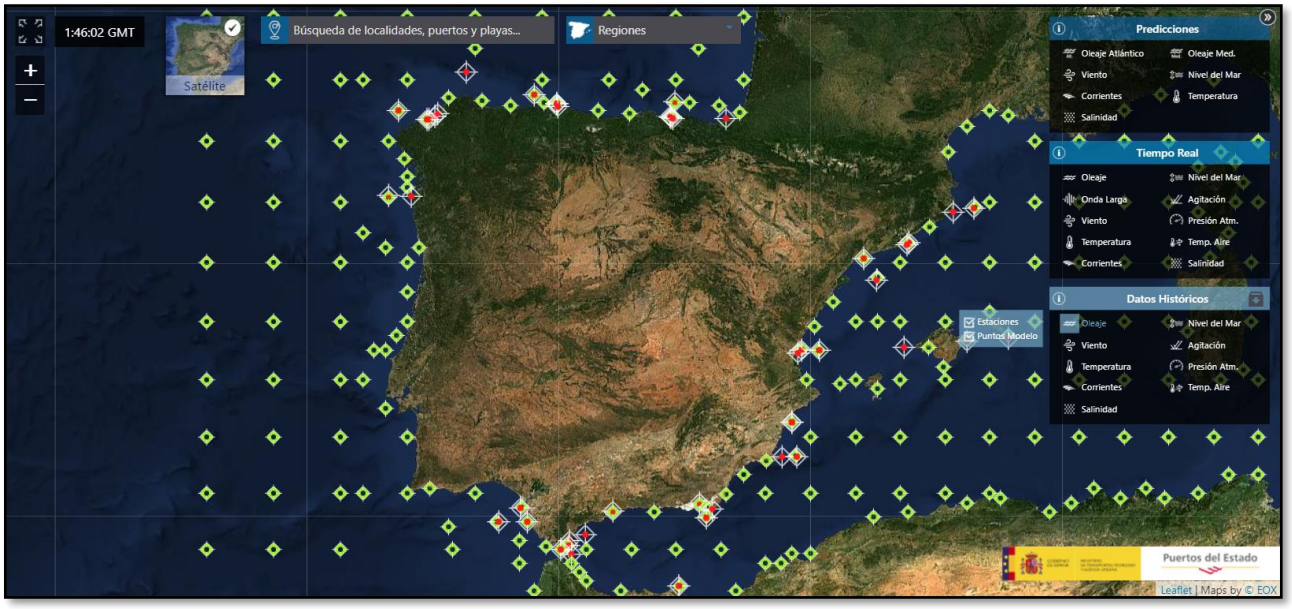


Figura 3-10 Configuración de descarga de datos de Puertos del Estado [23]

En este proyecto se escogerán únicamente puntos SIMAR debido a que la única boya de interés disponible se encuentra en las inmediaciones de Bayona y solo proporciona información hasta 2008 por lo que para este trabajo no nos sirve. Una vez seleccionado el punto deseado se nos abrirá una ventana emergente que consta de dos pestañas, una de oleaje y otra de todas las variables, en este caso se ha

seleccionado la de todas las variables con el fin de, posteriormente, realizar un cribado manual de los datos. Esto ha de hacerse con todos los puntos por separado, en el caso de este trabajo se han tomado los datos de los puntos rodeados en rojo de la Figura 3-11.

Una vez recibido el correo con la información de las boyas seleccionadas, hay que realizar una conversión de formato pues estos datos vienen en formato CSV. El formato CSV es completamente válido, sin embargo, con el objetivo de realizar el presente trabajo lo más uniforme posible se trabajará exclusivamente con archivos en formato JSON. Existen muchas formas de realizar esta conversión, en el presente trabajo se realizará desde una página web[16].

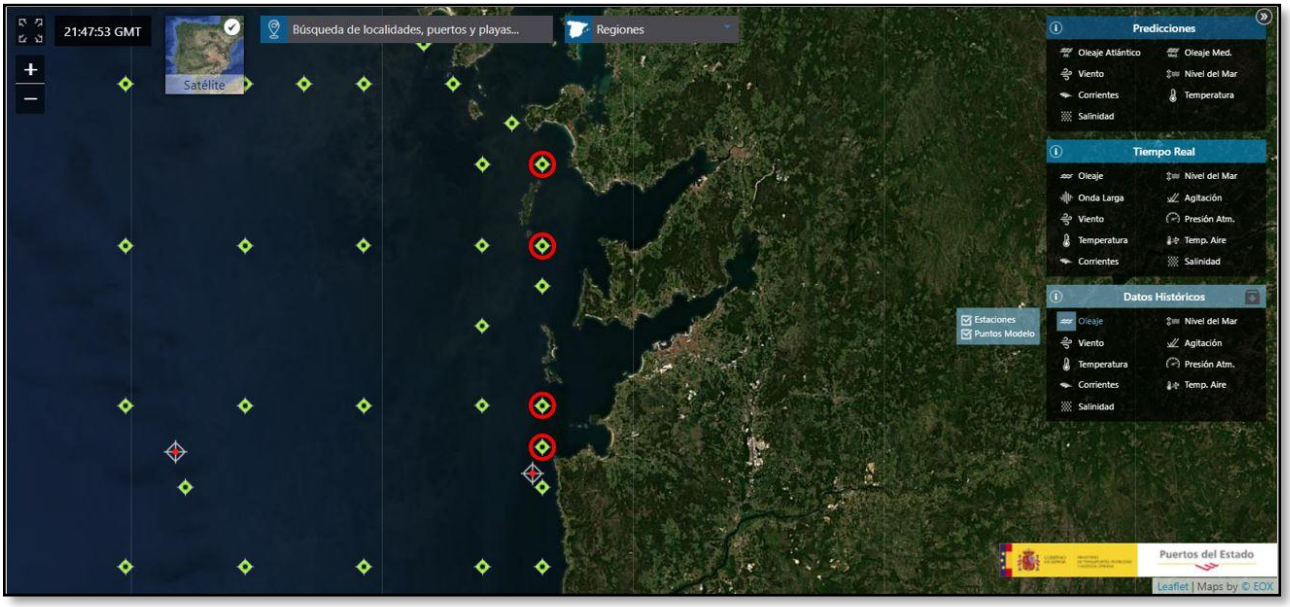


Figura 3-11 Puntos SIMAR seleccionados

3.3.1.3 EMODnet

Los datos acerca de la batimetría de las rías pueden extraerse de la página web de EMODnet[17]. Para ello se accede a la sección "DATA PRODUCTS" situada en el menú superior de la página. A continuación, se selecciona la opción "Bathymetry Viewing and Downloading Service" situada en el menú izquierdo. Esto nos llevará a una página en la que figura la Península Ibérica y desde la cual se puede realizar la descarga de la batimetría (Figura 3-12). A la hora de escoger el área cuenta con tres opciones: "DTM Tiles", "High resolution areas" y "Area of interest". Esta última permite seleccionar mediante un recuadro la zona que nos interesa, no obstante, hay que tener en cuenta que no todas las opciones permiten descargar los archivos en el mismo formato. Una vez seleccionada la zona que se desea, hay que proporcionar a EMODnet los datos de: nombre, correo electrónico, país, organización, tipo de organización y para que se van a emplear los datos descargados. Una vez realizado este formulario, EMODnet envía un correo a la dirección proporcionada con los documentos seleccionados.

[16] https://csvjson.com/
[17] https://www.emodnet-bathymetry.eu/

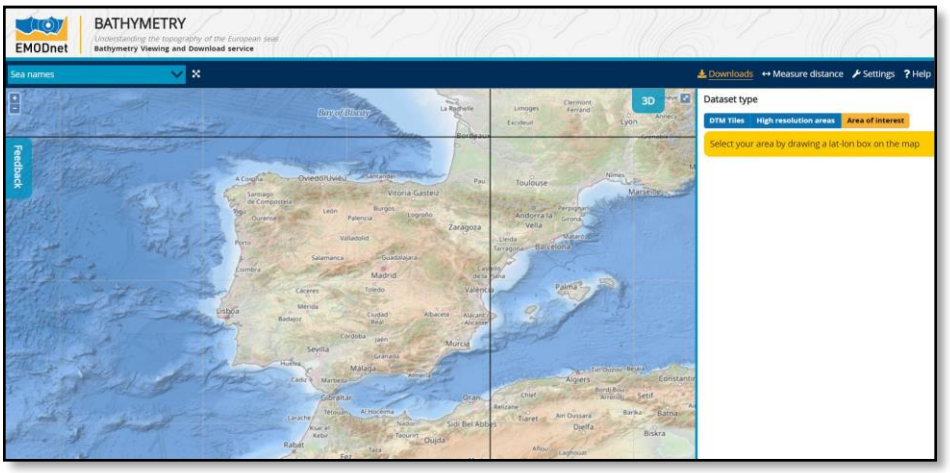


Figura 3-12 Descarga de datos de EMODnet

A la hora de escoger una de las opciones ya mencionadas se ha de descartar la de "*High resolution areas*" pues en ella no se encuentra ninguna zona con las que estamos trabajando y entre "*Area of interest*" y "DTM Tiles" solo la última nos ofrece un formato con el que resulta más cómodo trabajar, el XYZ.

3.3.2 Gestión semántica y de los datos

A la hora de generar la base de datos hay que clarificar cuales son los datos que interesan y de qué manera se van a estructurar. El objetivo fundamental de establecer relaciones entre los datos es de generar un falso entendimiento en la máquina. Para ello se han establecido tres predicados[18] básicos: "value_off", "subtype_off" y "location_off". Como se aprecia en la Figura 3-13, el paso de azul a rojo establece una relación de valor entre las medidas y el parámetro que se está midiendo; el de rojo a verde establece una relación de tipología entre dos objetos, en el caso de la Figura 3-13 se establece que los parámetros son tipos de meteorología; por último, el paso de azul a rojo de la Figura 3-14 establece una relación de ubicación entre dos objetos, en este caso el valor se encuentra ubicado en la estación X.

Una vez visto como se estructurarían los datos de forma genérica, en la Figura 3-15 , Figura 3-16 y Figura 3-17 queda reflejado como serían las estructuras con los objetos reales del trabajo.

[18] Entiéndase por predicado la relación que se establece entre el sujeto y el objeto.

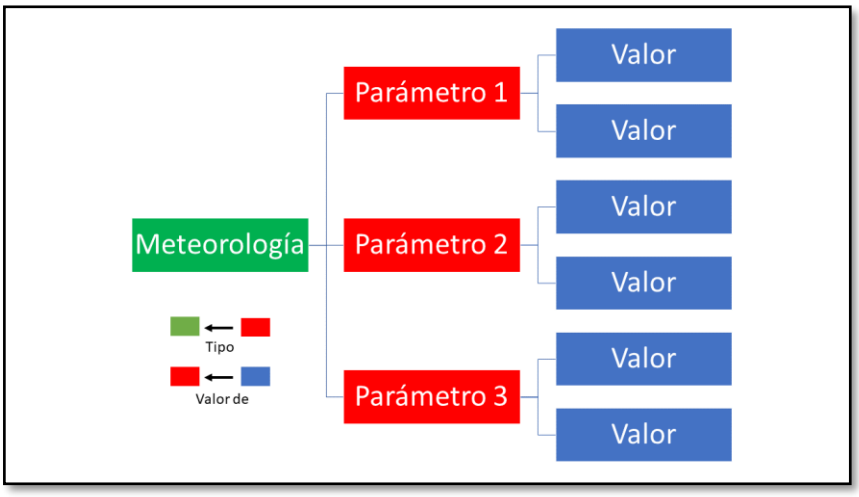


Figura 3-13 Esquema genérico 1

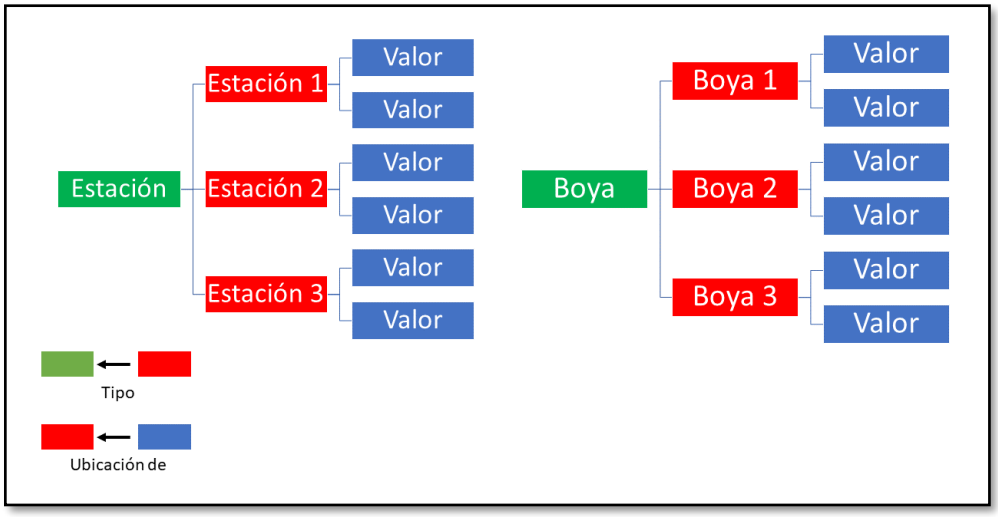


Figura 3-14 Esquema genérico 2

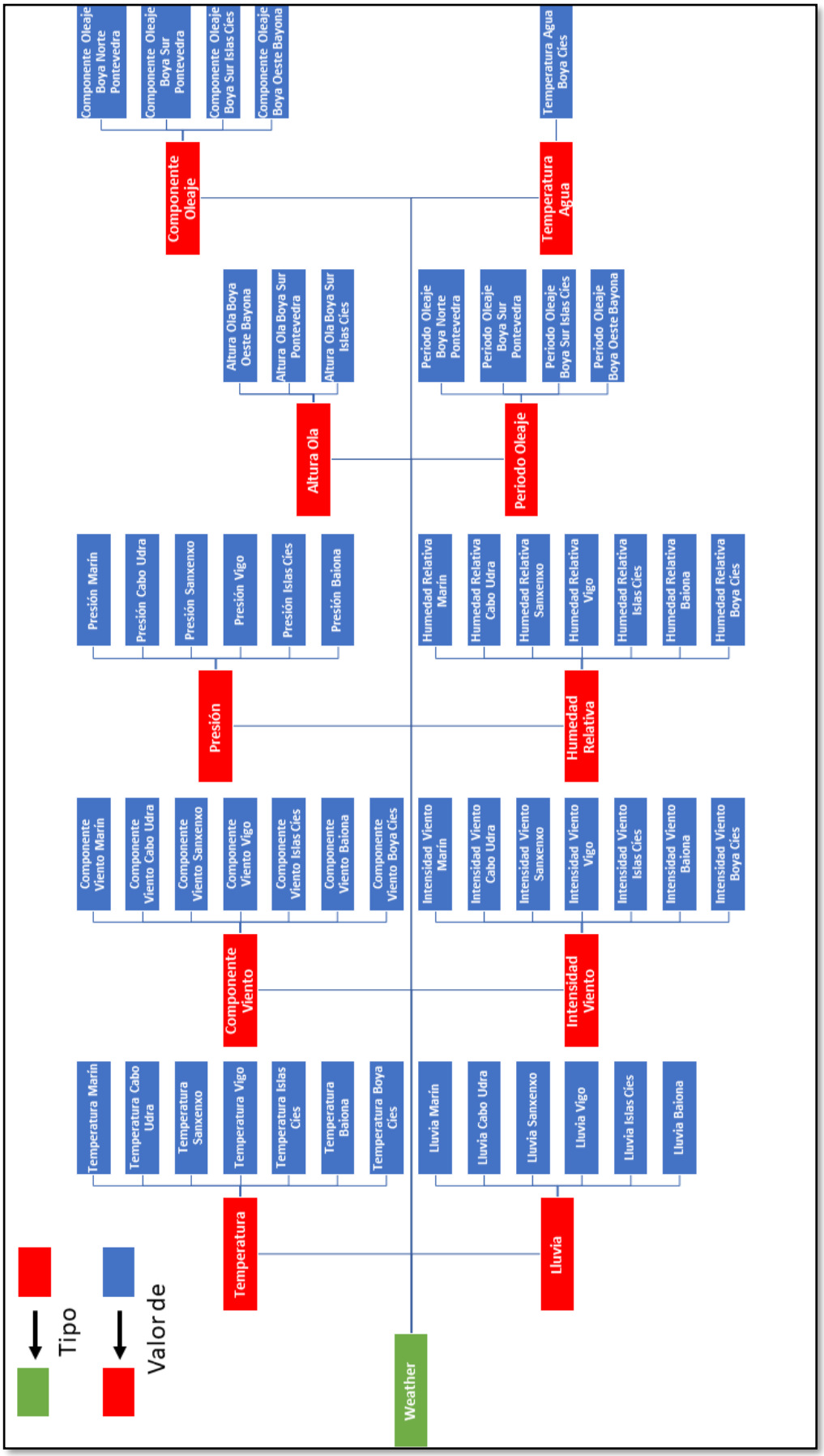


Figura 3-15 Estructura AllegroGraph 1

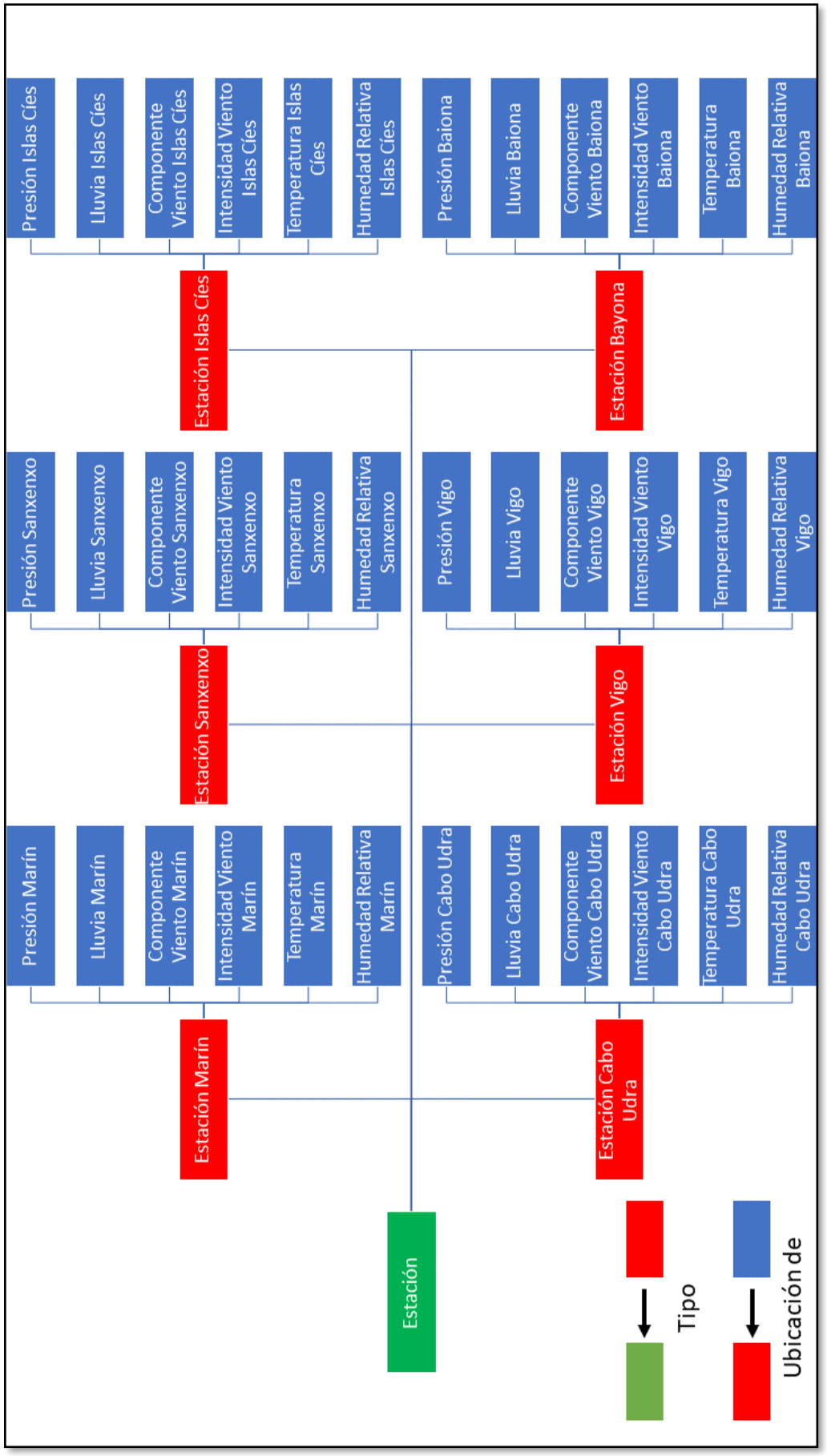


Figura 3-16 Estructura AllegroGraph 2

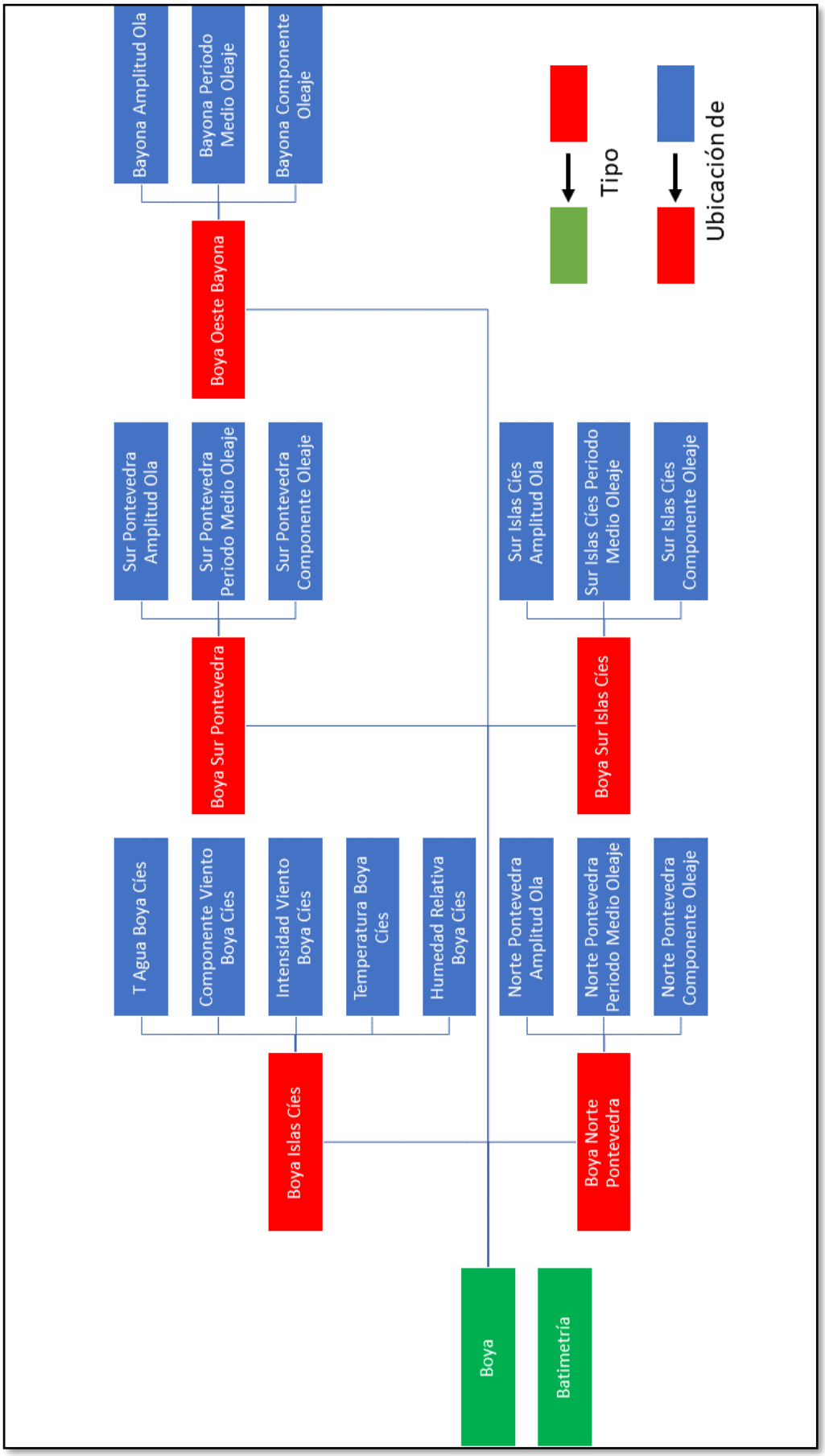


Figura 3-17 Estructura AllegroGraph 3

3.4 Implementación

3.4.1 Gestión de datos

En este apartado se abarcará de qué manera se han empleado los datos y de qué forma se han estructurado. Una vez diseñadas las relaciones entre unos objetos y otros y previo a su establecimiento en el servidor AllegroGraph se puede proceder a volcar la información a nuestra base de datos en MongoDB.

Antes de nada, es necesario establecer comunicación con nuestra base de datos y definir el nombre de la base de datos a crear, así como la colección (Figura 3-18). En cuanto a la forma de organizar los datos, se ha realizado un objeto por parámetro, fecha y estación. Para ello, se abre el archivo en el intérprete a través del comando referenciado en la Figura 3-19. Una vez abierto el archivo se emplea un bucle "`for`" (Figura 3-19) para leer dicho archivo, es decir, con este bucle se consigue hacer referencia a cada objeto y poder interactuar de forma individual con todos y cada uno de ellos. Como se verá más adelante, el bucle "`for`" se empleará reiteradas veces en el código.

```
4    myclient = pymongo.MongoClient("localhost")
5    mydb = myclient["Gemelo"]
6    mycol = mydb["TFG"]
```

Figura 3-18 Conexión MongoDB desde Pycharm

```
59   with open('Dia.json') as file:
60       data = json.load(file)
61
62       for resultado in data['resultados']:
```

Figura 3-19 Apertura y lectura del fichero JSON

MongoDB le asigna un identificador a cada objeto que se almacene en él, de forma general este se asigna automáticamente. Sin embargo, cabe la posibilidad de asignarle uno de forma manual. En este trabajo se asignará manualmente para poder hacer referencia a ello en el servidor AllegroGraph. Antes de asignarle un identificador hay que tener en cuenta que cada entrada en nuestra base de datos tiene que tener uno y este tiene que ser único. Por lo tanto, si ponemos el ejemplo de la temperatura, tendrá que haber tantos identificadores distintos como temperaturas tomadas. En este caso se ha optado por definir los identificadores haciendo referencia a la estación desde la que se ha tomado la medida, al parámetro y al número de medida de ese parámetro.

Para llevar el conteo de todos y cada uno de los objetos se define previamente una variable denominada con la letra "`i`" seguida de un guion bajo y las iniciales del parámetro al que hace referencia. El identificador se puede ver ejemplificado en la Figura 3-20 en el cual la letra "`M`" hace referencia a que se trata de la estación situada en Marín y las letras "`iv`" hacen referencia a que ese objeto es el que nos aporta la información acerca de la intensidad del viento.

Una vez definido ya se pueden definir el resto de datos, en el caso de la Figura 3-20 se establece el tipo de parámetro que se está midiendo y, del fichero JSON se recogen los datos acerca de la fecha y hora en la que se tomó dicha medida, el valor de la misma y en que unidades se mide. Una vez ya hemos definido la estructura del objeto solo queda establecer una suma de una unidad a nuestra variable de forma que, cuando se repita el bucle "`for`" para dar paso al siguiente objeto este pueda contar con un

identificador distinto. Y con esto último ya tenemos el objeto completamente definido por lo que ya solo queda insertarlo en nuestra base de datos.

Esto genera un gran número de objetos en nuestra base de datos, la otra opción disponible sería la de que en un mismo objeto se almacenasen todas las medidas, es decir, generar un objeto para un mismo parámetro y una misma estación y en él almacenar todos los valores con su correspondiente fecha. Con este método se reduciría considerablemente el número de objetos en nuestra colección, sin embargo, esto dificultaría enormemente el establecimiento de tripletas y, por consecuencia, la consulta de datos.

```
133         if resultado['Parámetro'] == 'Velocidade do vento a 10m':
134             data7['_id'] = "M_iv_"+str(i_iv)
135             data7['Tipo'] = 'Intensidad del Viento'
136             i_iv = i_iv + 1
137             data7['Fecha'] = resultado['Data']
138             data7['Valor'] = resultado['Valor']
139             data7['Unidad'] = resultado['Unidades']
140             x = mycol.insert_one(data7)
```

Figura 3-20 Ejemplo de objeto referido a la intensidad del viento

Con el fin de poder establecer una serie de relaciones en el apartado semántico con AllegroGraph y para facilitar la obtención del resultado buscado a la hora de realizar una consulta SPARQL, aparte de establecer objetos acerca de los valores obtenidos de las medidas, también se han insertado objetos con el fin de definir conceptos. Se han definido a modo de concepto todos y cada uno de los parámetros del archivoJSON. Así, cada objeto presenta un identificador que consiste en la letra "P" seguida de un guion bajo y el número de parámetro que le corresponda, además del identificador en cada objeto se almacena información acerca de su tipología, qué es lo que mide y que unidad de medida emplea (Figura 3-21). También han quedado definidas las estaciones de las que se extraen los datos. Al igual que los parámetros, su identificador consta de una letra, en este caso la "E" de estación (para el caso de las boyas se emplearía la letra "B"), y el número que tenga asignado, en este objeto también quedan definidos los aspectos de tipología, localización de la estación y las coordenadas en las que se encuentra (Figura 3-22).

```
28  Componente_Viento={}
29  Componente_Viento['_id']='p_3'
30  Componente_Viento['Tipo']='Parámetro'
31  Componente_Viento['Medición']='Componente del Viento'
32  Componente_Viento['Unidad']='°'
33  x = mycol.insert_one(Componente_Viento)
```

Figura 3-21 Ejemplo de un objeto de tipo parámetro

```
 99  Estación_Bayona = {}
100  Estación_Bayona['Tipo'] = 'Estación'
101  Estación_Bayona['_id']='E_B'
102  Estación_Bayona['Localización'] = 'Bayona'
103  Estación_Bayona['Latitud'] = '42.1155'
104  Estación_Bayona['Longitud'] = '-8.837179'
105  Estación_Bayona['Unidad'] = 'º'
106  x = mycol.insert_one(Estación_Bayona)
```

Figura 3-22 Ejemplo de objeto de tipo estación

En cuantos a los puntos SIMAR, por ser aproximaciones matemáticas se les considera en todo momento igual que a una boya en cuanto a terminología se refiere. Para estos la estructura seguida es algo distinta. Antes que nada, se han definido las variables que en si no guardan ningún valor pero que son necesarias para relacionarlas con otras más adelante desde nuestro servidor AllegroGraph, los objetos definidos son los siguientes: "`Boya`", "`Altura Ola`", "`Periodo Medio Oleaje`", "`Componente Oleaje`", "`Boya Norte Pontevedra`", "`Boya Sur Pontevedra`", "`Boya Sur Islas Cíes`" y "`Boya Oeste Bayona`".

Al igual que con las estaciones, se emplea un bucle "`for`" para trabajar con los datos de forma individual, la diferencia que tienen con estas es que el archivo JSON viene estructurado de una manera muy distinta. En este a las distintas variables se les asigna números precedidos por dos guiones bajos.

Además, el valor del que se dispone para realizar el condicional es el de la fecha. Por ello como, como se puede apreciar en la Figura 3-23 se definen 3 variables, en cada una de ellas se almacenará la fecha en la que fue tomada la medida, el valor de la misma y que unidades tiene. Y esto se realizará para las 4 boyas disponibles. A la hora de establecer la condición, hay que tener en cuenta que la fecha cambia. Es por ello que, como refleja la Figura 3-23, se ha decidido añadir un contador que vaya sumando uno por cada vuelta que da el bucle y de esta forma vaya avanzando la fecha y se puedan almacenar valores nuevos. Hay que tener en consideración el formato que posee la fecha pues los números hay que escribirlos con dos caracteres de longitud, para ello se utiliza el comando "`+str(i).zfill(2)`", de esta forma convertimos la variable de "`double`" a "`string`" y especificamos la longitud de caracteres que debe tener "`i`", en este caso debido a las exigencias del código, la longitud ha de ser de 2. Como se puede apreciar en la Figura 3-23, por cada boya se almacenan 3 objetos con información acerca de la fecha, el valor de la medición y las unidades en las que se mide.

```
for resultado in data:
        data1 = {}
        data2 = {}
        data3 = {}
        if resultado['Valor nulo: -9999.9'] == ('2022 02 01 '+str(i).zfill(2)):
            data1['_id']='NP_AO_'+str(i)
            data1['Fecha']=resultado['Valor nulo: -9999.9']
            data1['Altura_Ola']=resultado['']
            data1['Unidades']='m'
            data2['_id'] = 'NP_PO_' + str(i)
            data2['Fecha'] = resultado['Valor nulo: -9999.9']
            data2['Periodo_Medio_Oleaje']=resultado['__1']
            data2['Unidades'] = 's'
            data3['_id'] = 'NP_CO_' + str(i)
            data3['Fecha'] = resultado['Valor nulo: -9999.9']
            data3['Componente_Oleaje']=resultado['__2']
            data3['Unidades'] = '°'
            x = mycol.insert_one(data1)
            x = mycol.insert_one(data2)
            x = mycol.insert_one(data3)
            i=i+1
```

Figura 3-23 Código referente a las boyas de Puertos del Estado

A la hora de volcar en MongoDB la información acerca de la batimetría, por cada objeto se seguirá la siguiente estructura: tipo (en este caso batimetría), longitud, latitud y sonda.

En la Figura 3-24 quedan reflejados todos los objetos definidos en la base de datos con sus respectivos identificadores. Así, los verdes corresponden a objetos definidos para poder establecer conexiones en el servidor AllegroGraph, los azules corresponden a los valores de las mediciones volcados en la base de datos, el de color gris corresponde a la batimetría y los de color rojo refieren a boyas.

Figura 3-24 Objetos y sus identificadores

3.4.2 Gestión Semántica

Una vez definido el grafo de relaciones entre conceptos que se va a seguir, tal y como se ha ilustrado en la Figura 3-24,Figura 3-15, Figura 3-16 y Figura 3-17, ya se puede proceder a generarla a través de Python. Para ello hay que empezar por arrancar el servidor AllegroGraph lo cual se realiza ejecutando en el terminal el comando referenciado en la Figura 3-25. Una vez arrancado podemos acceder a él a través de la dirección: http://localhost:10035. Dispone de una interfaz muy intuitiva a través de la cual se puede acceder a todas las tripletas que tenga almacenadas el repositorio.

```
enrique@enrique-VirtualBox:~$ /home/enrique/Documents/ag7.2.0/bin/agraph-control --config
/home/enrique/Documents/ag7.2.0/lib/agraph.cfg start
```

Figura 3-25 Código de arranque del servidor AllegroGraph

Una vez arrancado el servidor, el siguiente paso a seguir es establecer conexión con él y nombrar un repositorio, en este trabajo se le llamará “Gemelo” (Figura 3-26). Con estos pasos el servidor quedaría listo para ser utilizado de forma individual, sin embargo, lo que interesa para este proyecto es que establezca una conexión con una base de datos y trabaje a partir de estos. Por ello, hay que dirigirse a la página del repositorio con el que se establecerá la conexión, y en la parte inferior se encontrará un apartado denominado “Manage external MongoDB connection”, simplemente queda introducir los campos que aparecerán en pantalla y se realizará la conexión.

```
1  from franz.openrdf.connect import ag_connect
2
3  conn = ag_connect('Gemelo', host='localhost', port='10035', user='test', password='test')
```

Figura 3-26 Establecimiento de conexión con AllegroGraph

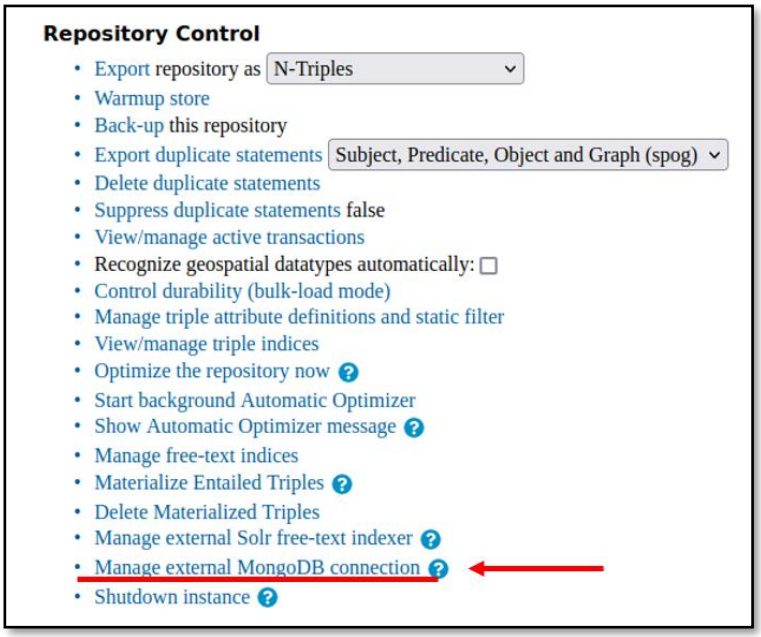


Figura 3-27 Conexión entre AllegroGraph y MongoDB

A continuación, se procede a establecer las relaciones entre los distintos objetos, para ello se empieza con la función referenciada en la Figura 3-28, una vez en ella vamos creando nuestras tripletas. Como se ha mencionado previamente en este trabajo, la estructura de las tripletas es de sujeto-predicado-objeto. Teniendo esto en cuenta, en el repositorio hay que crear tantos objetos como nuestra base de datos disponga. Como se puede observar en la Figura 3-28, la secuencia es la siguiente: primero se crea un

identificador en Allegro y a continuación, se le asigna el valor del identificador del objeto ubicado en la base de datos usando el prefijo `f` del `namespace` indicado en la Figura 3-28 mediante la relación "`f:hasMongoId`" entre el concepto y el identificador. Una vez ya generado el objeto en el repositorio y habiéndole asignado un objeto de la base de datos ya se le pueden definir el resto de relaciones.

```
19    conn.addData("""
20    @prefix f: <http://www.franz.com/> .
21    @prefix id: <http://www.franz.com/id#> .
22
23    id:Meteorología f:hasMongoId "w" .
```

Figura 3-28 Ejemplo de objeto AllegroGraph

A la hora de generar los objetos en el AllegroGraph se plantea la problemática de que en la base de datos existen cientos de objetos debido a los diferentes parámetros y las diferentes medidas tomadas, para solventarlo es necesario automatizarlo mediante un bucle "`for`". Primero se abre el archivo JSON que contenga los objetos que se habían volcado previamente en la base de datos, y al igual que se hacía para generar la base de datos se pone un bucle "`for`" y una condición. La diferencia esta vez es que no se busca almacenar ningún tipo de información de dicho archivo si no que se emplea para llevar el mismo conteo que se llevó a la hora de volcar los datos en la base. En la Figura 3-29 se muestra un ejemplo de cómo se generan en AllegroGraph todos los objetos acerca de la presión que hay en Marín, cómo se les asigna los objetos de MongoDB y cómo establecen relaciones con otros objetos. Esta misma secuencia se repite una y otra vez para todos los parámetros y a su vez para todos los archivos.

```
70  for resultado in data['resultados']:
71
72      if resultado['Parámetro'] == 'Chuvia':
73          conn.addData("""
74          @prefix f: <http://www.franz.com/> .
75          @prefix id: <http://www.franz.com/id#> .
76
77          id:Presión_Marín_"""+str(i_l)+"""    f:hasMongoId \"""""M_p_"+str(i_l)+"""\" ;
78                              f:value_of id:Presión .
79
```

Figura 3-29 Bucle "for" para la generación de objetos en AllegroGraph

4 RESULTADOS

4.1 Captura de datos

Teniendo en cuenta la metodología y la implementación explicadas en el anterior capítulo, resulta importante indicar los datos que se han usado para la elaboración de este TFG. Por tanto, siguiendo con la estructura planteada con anterioridad, creemos que es justo mantenerlo aquí. Comencemos entonces con los datos capturados de MeteoGalicia.

Como se observará más adelante, se vuelca una gran cantidad de datos, es por ello que se tomó la decisión de emplear únicamente los datos recopilados durante un día con la premisa de realizar el trabajo de la forma más fluida posible, no obstante, la estructura de datos es la misma sea el número de datos que sea por lo que en el caso de que se trabajase con un equipo de alto rendimiento capaz de procesar grandes cantidades de datos, se podrían cargar ficheros del tiempo que uno desee.

4.1.1 MeteoGalicia

De MeteoGalicia se han extraídos datos exclusivamente del día 1 de febrero de 2022, se han escogido los parámetros de lluvia, presión, humedad, temperatura e intensidad y componente del viento. Como se ha mencionado previamente se ha optado por trabajar con 6 estaciones meteorológicas diferentes por lo que se ha de realizar una descarga por estación. Todas ellas con la misma configuración a excepción de Cabo Udra, Islas Cíes y Bayona que no disponen de datos acerca de la presión.

A la hora de descargarse los archivos en formato JSON lo hacen todos con el nombre de "`resultadosJSON`" por lo que se ha optado por renombrarlos todos: el archivo referente a la estación de Marín pesa 135 Kb y ha sido renombrado a "`Dia.json`", el referente a la de Sanxenxo pesa 134 Kb y ha sido renombrado a "`Sanxenxo.json`", el referente a la de Cabo Udra pesa 113 Kb y se ha renombrado a "`Cabo_Udra.json`", el referente a la de Vigo pesa 135 Kb ha sido renombrado a "`Vigo.json`", el referente a la de las Islas Cíes pesa 113 Kb y se ha renombrado a "`Islas_Cíes.json`" y por último el referente a la de Bayona que pesa 114 Kb y se renombrado a "`Baiona.json`". A su vez ocurre lo mismo con la descarga de datos de la boya de MeteoGalicia por lo que tras renombrarlo queda un archivo de 168 Kb denominado "`Boya1.json`".

Como se ha visto en el apartado de implementación, se ha creado un objeto por fecha, parámetro y estación. Al tomar valores de cada diez minutos durante un día entero, se recogen un total de 144 mediciones por parámetro, se dispone de tres estaciones que toman datos de seis parámetros distintos y de otras tres estaciones más una boya que toman datos de cinco. Todo esto se traduce en un total de 5.472 objetos generados en la base de datos.

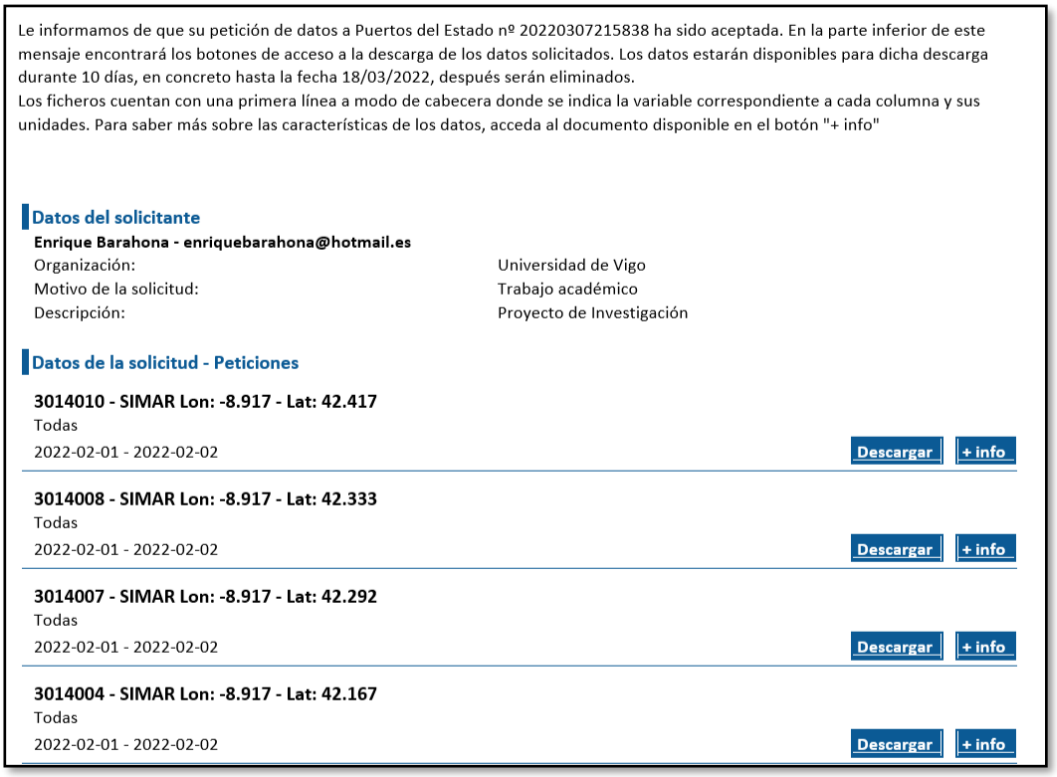
4.1.2 Puertos del Estado

Como se ha mencionado previamente en este trabajo, para de realizar la descarga de datos de Puertos del Estado hay que rellenar una solicitud y a partir de ahí en 24h-48h mandan un correo con los enlaces de descarga.

La Figura 4-1 es una captura del correo que envían, en él advierten que los links de descarga solo estarán disponibles durante diez días, a partir de ahí expirarán y habrá que realizar otra solicitud. En el correo también figura en qué coordenadas se encuentran los puntos de los que se ha pedido información. Los archivos vienen nombrados con una secuencia de números tal que: "`8593_5060_3014010_ALL_20220201215329_20220202215329.csv`" por lo que, al igual que se hizo con los archivos de MeteoGalicia, estos también se han renombrado. Antes de pasar con las capacidades y los nombres de los distintos archivos cabe mencionar que los archivos de Puertos del Estado vienen en formato CSV y no en JSON que es el formato con el que se está trabajando. Es por ello que se vio oportuno realizar una conversión de formato, para ello se accede a un conversor online, se sube el archivo CSV y este ya te lo convierte a formato JSON. Una vez realizada la conversión se procede a renombrarlos resultando en cuatro archivos: "`Entrada_Norte_Pontevedra.json`", "`Entrada_Sur_Pontevedra.json`", "`Sur_Islas_Cíes.json`" y "`Oeste_Bayona.json`" de 6 KB cada uno de ellos.

Estos contienen los siguientes datos: fecha, altura significativa del oleaje, periodo medio, periodo de pico, dirección media de procedencia, altura significativa espectral del mar de viento, dirección media de procedencia del mar de viento, altura significativa espectral del mar de fondo, periodo medio espectral del mar de fondo, dirección media de procedencia del mar de fondo, velocidad del viento y dirección de procedencia del viento.

No obstante, todos los datos referidos al mar de fondo y al mar de viento aparecen como nulos. Por ello se ha optado por recopilar exclusivamente los datos referentes a la altura de la ola, el periodo del oleaje y su componente, excluyendo el viento y su dirección pues para ello ya se dispone de la información proporcionada por las estaciones meteorológicas. Al igual que los datos extraídos de MeteoGalicia, se han cogido datos de día 1 de febrero de 2022 pero, a diferencia de este, los datos son tomados cada hora por lo que se toman 23 medidas por parámetro. Teniendo en cuenta que se han seleccionado cuatro boyas y que cada una toma medidas de 3 parámetros distintos, a la base de datos se vuelcan un total de 276 objetos.


Le informamos de que su petición de datos a Puertos del Estado nº 20220307215838 ha sido aceptada. En la parte inferior de este mensaje encontrará los botones de acceso a la descarga de los datos solicitados. Los datos estarán disponibles para dicha descarga durante 10 días, en concreto hasta la fecha 18/03/2022, después serán eliminados.
Los ficheros cuentan con una primera línea a modo de cabecera donde se indica la variable correspondiente a cada columna y sus unidades. Para saber más sobre las características de los datos, acceda al documento disponible en el botón "+ info"

Datos del solicitante

Enrique Barahona - enriquebarahona@hotmail.es

Organización: Universidad de Vigo
Motivo de la solicitud: Trabajo académico
Descripción: Proyecto de Investigación

Datos de la solicitud - Peticiones

3014010 - SIMAR Lon: -8.917 - Lat: 42.417
Todas
2022-02-01 - 2022-02-02 Descargar + info

3014008 - SIMAR Lon: -8.917 - Lat: 42.333
Todas
2022-02-01 - 2022-02-02 Descargar + info

3014007 - SIMAR Lon: -8.917 - Lat: 42.292
Todas
2022-02-01 - 2022-02-02 Descargar + info

3014004 - SIMAR Lon: -8.917 - Lat: 42.167
Todas
2022-02-01 - 2022-02-02 Descargar + info

Figura 4-1 Correo de Puertos del Estado

4.1.3 EMODnet

EMODnet permite la descarga de datos en distintos formatos dependiendo de qué opción se escoja a la hora de seleccionar el área. Para este trabajo el que mejor se adapta es el formato GeoJSON debido que se puede trabajar con él como si de un JSON se tratase, sin embargo, esta no es una opción de descarga. La opción que mejor se adapta al objeto del trabajo es la de selección de área, pero esta solo permite la descarga de archivos en los formatos ESRI ASCII y GeoTiff por lo que se escogerá la opción de descarga de Digital Terrain Model Tiles la cual nos permite descargar el archivo en formato XYZ.

Existe un programa denominado QGis el cual permite trabajar con este tipo de archivos. Originalmente el archivo descargado ocupa 130 MB, pero se trata de una zona que abarca mucho más de lo que se quiere analizar por lo que se ha introducido en QGis donde no solo se le ha cambiado el formato, sino que también se ha reducido la zona de modo que abarque exclusivamente las rías de Pontevedra y Vigo dejando un archivo con un peso de 18,5 MB. Para la batimetría se ha creado un objeto por cada ubicación y sonda por lo que se han volcado 96.390 objetos.

5 CONCLUSIONES Y LÍNEAS FUTURAS

5.1 Conclusiones

Ya finalizado el presente trabajo se puede concluir que se han alcanzado todos los objetivos propuestos. Se ha trabajado con datos extraídos de tres fuentes distintas y, en el caso de dos de ellas, se ha tenido que incluso modificar el formato. Se ha extraído y manejado la información descargada realizando una criba de datos en la que se han desechado aquellos considerados irrelevantes para la navegación y se han incluido en la base de datos aquellos que eran estimados de provecho.

Además, en este trabajo se han analizado distintos tipos de bases de datos optando por una no estructurada en base al potencial que tiene frente a la relacional. También se ha conseguido aprender a trabajar con este tipo de bases de datos mediante la inserción de los datos previamente obtenidos de fuentes externas, así como la realización de consultas.

Otro de los objetivos logrados ha sido el entender las posibles interrelaciones de los datos extraídos de las distintas fuentes, creado un grafo de relaciones semánticas entre cada componente considerado de relevancia. Mediante código de programación Python, se generaron una serie de tripletas con la estructura sujeto-predicado-objeto, de esta forma se establecieron una serie de relaciones entre los distintos objetos contenidos en la base de datos. Para ello se han empleado fundamentalmente relaciones de tipología y de ubicación.

Finalmente, se han realizado pruebas básicas de consultas semánticas SPARQL utilizando las relaciones establecidas previamente. Se han realizado consultas dando un valor y preguntando en qué zona se dio ese fenómeno o bien dando un parámetro y una fecha que salgan los valores de las distintas estaciones, y como estas se podrían hacer infinidad de consultas.

5.1 Líneas futuras

Tras la conclusión de este trabajo se han considerado las siguientes líneas de mejora:

- Información en tiempo real. En este trabajo se han utilizado los datos extraídos de un día en concreto y, como ya se ha mencionado, en él se podrían ejecutar datos de cualquier periodo de tiempo siempre que se disponga de un ordenador lo suficientemente potente. Sin embargo, sería de provecho poder conseguir que la información se fuese extrayendo en tiempo real y volcando de forma automática a la base de datos. Esto evitaría que cada vez que se quiere información hubiese que descargar un archivo configurando en la página web las

preferencias e incluso, en el caso de Puertos del Estado y EMODnet, tener que cambiar el formato.

- Otro aspecto sería la optimización del código. Se trataría de hacer que el código creado fuese más eficiente para que el tiempo de ejecución sea menor.

6 BIBLIOGRAFÍA

[1] M. Batty, «Digital Twins,» de *Environment and Planning B: Urban Analytics and City Science*, 2018, pp. 817-820.

[2] R. Estévez, «Eco Inteligencia,» 5 4 2019. [En línea]. Available: https://www.ecointeligencia.com/2019/04/gemelos-digitales/. [Último acceso: 2 2 2022].

[3] «A transparent & accessible ocean: Towards a Digital Twin of the Ocean,» 2020.

[4] «European Comission,» [En línea]. Available: https://ec.europa.eu/info/index_en. [Último acceso: 2 2 2022].

[5] M. Giménez, «Hiberus,» 15 8 2019. [En línea]. Available: https://www.hiberus.com/crecemos-contigo/que-es-bim-construccion/. [Último acceso: 27 1 2022].

[6] K. Fugas, «Bim Corner,» 9 3 2020. [En línea]. Available: https://bimcorner.com/how-to-present-building-information-in-a-database/. [Último acceso: 28 1 2022].

[7] «EMODnet,» [En línea]. Available: https://emodnet.ec.europa.eu/en/about_emodnet. [Último acceso: 6 2 2022].

[8] «EMODnet,» [En línea]. Available: https://emodnet.ec.europa.eu/en/logos. [Último acceso: 6 2 2022].

[9] «Puertos del Estado,» [En línea]. Available: https://www.puertos.es/es-es/nosotrospuertos/Paginas/Nosotros.aspx . [Último acceso: 09 02 2022].

[10] «Puertos de Tenerife,» [En línea]. Available: https://www.puertosdetenerife.org/. [Último acceso: 09 02 2022].

[11] «Universidad de Santiago de Compostela,» [En línea]. Available: https://www.usc.gal/es/servizos/meteogalicia. [Último acceso: 09 02 2022].

[12] «Xunta de Galicia,» [En línea]. Available: https://www.edu.xunta.gal/portal/es/node/20961. [Último acceso: 09 02 2022].

[13] «Ceweb,» [En línea]. Available: https://ceweb.br/guias/web-semantica/es/capitulo-4/. [Último acceso: 11 03 2022].

[14] «Ontotext,» [En línea]. Available: https://www.ontotext.com/knowledgehub/fundamentals/what-is-sparql/#:~:text=%EF%BB%BF,can%20be%20mapped%20to%20RDF.. [Último acceso: 2022 03 12].

[15] «Ayudaley,» [En línea]. Available: https://ayudaleyprotecciondatos.es/bases-de-datos/diferencias-entre-datos-estructurados-y-no-estructurados/. [Último acceso: 5 2 2022].

[16] «Kyocera,» [En línea]. Available: https://www.kyoceradocumentsolutions.es/es/smarter-workspaces/insights-hub/articles/diferencia-entre-datos-estructurados-y-no-estructurados.html. [Último acceso: 5 2 2022].

[17] «Lawtomated,» 7 4 2019. [En línea]. Available: https://lawtomated.com/structured-data-vs-unstructured-data-what-are-they-and-why-care/. [Último acceso: 6 2 2022].

[18] «MongoDB,» [En línea]. Available: https://www.mongodb.com/databases/mongodb-graph-database. [Último acceso: 10 2 2022].

[19] «MDN Web Docs,» [En línea]. Available: https://developer.mozilla.org/es/docs/Learn/JavaScript/Objects/JSON. [Último acceso: 12 03 2022].

[20] «mongoDB,» [En línea]. Available: https://www.mongodb.com/es/what-is-mongodb. [Último acceso: 8 2 2022].

[21] «mongoDB,» [En línea]. Available: https://www.mongodb.com/json-and-bson. [Último acceso: 2 8 2022].

[22] «muylinux,» [En línea]. Available: https://www.muylinux.com/2019/01/17/mongodb-rechazo-nueva-licencia/. [Último acceso: 8 2 2022].

[23] R. Fernández, «Genbeta,» 3 2 2014. [En línea]. Available: https://www.genbeta.com/desarrollo/mongodb-que-es-como-funciona-y-cuando-podemos-usarlo-o-no. [Último acceso: 8 2 2022].

[24] A. Robledano, «OpenWebinars,» 28 10 2019. [En línea]. Available: https://openwebinars.net/blog/que-es-mongodb/. [Último acceso: 8 2 22].

[25] «MeteoGalicia,» [En línea]. Available: https://www.meteogalicia.gal/observacion/rede/redeIndex.action. [Último acceso: 09 02 2022].

[26] «Puertos del Estado,» [En línea]. Available: https://www.puertos.es/es-es/oceanografia/Paginas/portus.aspx. [Último acceso: 09 02 2022].

ANEXO I: INSTALACIÓN

A.1 Instalación

A.1.1 Máquina virtual

AllegroGraph funciona con el sistema operativo Linux por lo que, en caso de utilizar Windows, como es este caso, habrá que empezar por instalar una máquina virtual que nos permita simularlo. Debido a ello utilizaremos también el resto de herramientas en Linux. Para ello vamos a instalar la máquina virtual VirtualBox la cual nos permite simular distintos sistemas operativos y guardarlos como si fuesen ordenadores independientes.

Para su instalación hay que dirigirse a su página web oficial[19] donde se puede descargar la última versión del programa. La instalación es bastante sencilla y no difiere de la de un programa cualquiera. Una vez finalizada la instalación hay que proceder a crear la simulación de Linux. Existen muchos sistemas operativos basados en Linux, en este proyecto emplearemos Ubuntu debido a su gran estabilidad y a la gran comunidad que lo respalda. Para poder simular Ubuntu en nuestra máquina virtual es necesario instalar la imagen ISO del mismo. Para ello accedemos a la página oficial de Ubuntu[20], una vez dentro seleccionamos la sección "Downloads" situada en la parte de arriba, a continuación, seleccionamos donde pone "Ubuntu Desktop", esto debería llevarnos a la página con las distintas versiones de Ubuntu, escogemos la última versión y la descargamos. Una vez tenemos descargada la imagen ISO pasamos a crear nuestro entorno virtual.

Abrimos el programa VirtualBox, una vez dentro vemos que existen distintas opciones, para crear nuestra máquina virtual tendremos que ir a donde pone "Nueva". Esto desplegará una nueva ventana la cual le pedirá que introduzca un nombre para la máquina en si y que establezca el tipo y la versión de la misma, para el caso de nuestro proyecto utilizaremos una de tipo Linux y de versión Ubuntu (64-bit), una vez rellenados estos datos el proceso continúa pidiéndonos la cantidad de memoria RAM que queremos dedicarle a nuestra máquina virtual. En este proyecto se trabaja con una gran cantidad de datos por lo que le dedicaremos cuatro gigabytes (4096 MB). En el siguiente paso se presenta la opción a crear un disco duro virtual, tras escoger la opción de crearlo se abre otra ventana para definir el tipo de archivo de disco duro, dejaremos las opciones que vienen por defecto marcadas, VDI y reservado dinámicamente. En cuanto a la capacidad del disco la pondremos en 30 GB para asegurarnos que no encontraremos problemas más adelante con el tamaño y la cantidad de datos. Una vez realizados todos estos pasos se nos creará nuestra máquina virtual la cual aparecerá en la columna de la izquierda, sin embargo, esta no está todavía configurada.

Para ellos encendemos nuestra máquina virtual haciendo doble clic sobre ella, a continuación, nos pedirá que seleccionemos un disco de inicio, esto es la imagen ISO de Ubuntu que hemos descargado antes por lo que clicamos en el icono de la carpeta y en la nueva ventana le damos a añadir, nos dirigimos a la carpeta donde tenemos la imagen ISO y la seleccionamos. Una vez le demos a iniciar arranca la máquina virtual y aparecen los pasos de configuración del supuesto equipo. Primero escogemos nuestro idioma y clicamos en instalar Ubuntu, a continuación, le vamos dando a continuar dejando todo lo que está marcado por defecto. Por último, hay que escoger un usuario y una contraseña que serán los que sirvan para iniciar sesión en Ubuntu. Y así ya tendríamos instalado nuestra máquina virtual con Ubuntu instalado.

[19] https://www.virtualbox.org/
[20] https://ubuntu.com/

A.1.2 Pycharm

Para poder configurar nuestra base semántica necesitamos emplear un entorno en el que programar en Python, en este proyecto utilizaremos el entorno de trabajo Pycharm. Para instalar Pycharm necesitaremos acceder a su página web oficial, en el apartado descargas encontraremos dos opciones: la Professional y la Community. En este caso utilizaremos la Community ya que la Professional es una versión de pago y la Community ya nos brinda todas las herramientas que necesitaremos.

A continuación, debemos descomprimir el archivo descargado, esto nos creará una carpeta. Una vez dentro de la carpeta veremos que hay diferentes archivos y carpetas, para iniciar la aplicación Pycharm bastará con abrir la carpeta "`bin`" en el terminal de Ubuntu y escribir un comando de arranque, para ello haremos clic derecho sobre la carpeta "`bin`" y dentro de las opciones seleccionamos la que dice "Open in Terminal", una vez dentro de la consola de comandos escribimos la línea de código que muestra la Figura A-1, le damos a la tecla Intro y el programa se iniciará.

```
enrique@enrique-VirtualBox:~/Downloads/pycharm-community-2021.3.2/bin$ ./pycharm.sh
```

Figura A-1 Código de instalación de Pycharm

A.1.3 AllegroGraph

Para instalar AllegroGraph accedemos al apartado descargas de su página web y nos bajamos el archivo correspondiente a Ubuntu. A continuación, se descargará un archivo que debemos descomprimir, abrimos la carpeta resultante de la descompresión en el terminal y escribimos la línea de código que aparece en la Figura A-2 siendo la primera parte de este código el nombre del archivo a instalar y la segunda la ubicación donde queremos que se instale. Durante el proceso de instalación se pedirá un usuario y una contraseña para acceder al respositorio de AllegroGraph.

```
enrique@enrique-VirtualBox:~$ agraph-7.2.0/install-agraph /home/enrique/Documents/ag7.2.0
```

Figura A-2 Instalación AllegroGraph

Una vez ya tenemos AllegroGraph instalado pasamos a arrancar el servidor, para ello escribimos el código referenciado en la Figura A-3. A continuación podremos entrar en el repositorio iniciando sesión con el usuario y contraseña que establecimos en la instalación accediendo a través de nuestro navegador a la siguiente dirección: http://localhost:10035 (ver Figura A-4).

```
enrique@enrique-VirtualBox:~$ /home/enrique/Documents/ag7.2.0/bin/agraph-control --config
/home/enrique/Documents/ag7.2.0/lib/agraph.cfg start
```

Figura A-3 Arranque del servido de AllegroGraph

Figura A-4 Inicio de sesión del servidor AllegroGraph

A.1.4 Python

La instalación de Python es bastante sencilla, basta con ir a su página web oficial y descargar la versión compatible con Linux, esto nos descargará un archivo comprimido, lo descomprimimos, abrimos el terminal y escribimos las líneas de código en él orden en que aparecen en la Figura A-5 pulsando la tecla intro entre cada una de ellas. Una vez hayan terminado los procesos Python estará instalado y podremos emplearlo en nuestra aplicación Pycharm.

```
enrique@enrique-VirtualBox:~$ ./configure
enrique@enrique-VirtualBox:~$ make
enrique@enrique-VirtualBox:~$ make test
enrique@enrique-VirtualBox:~$ sudo make install
```

Figura A-5 Código de instalación de Python

A.1.5 MongoDB Compass

Para instalar el MongoDB Compass primero tenemos que instalar previamente el MongoDB 3.6 o superior. Para ello necesitaremos abrir nuestro terminal de Ubuntu e ir ejecutando de forma individual los códigos referenciados en la Figura A-6, una vez finalizado este proceso procedemos a instalar el MongoDB Compass. Al igual que la instalación previa, la instalación solo requiere de ejecutar una serie de comando en nuestro terminal, estos quedan reverenciados en la Figura A-7.

```
enrique@enrique-VirtualBox:~$ wget -q0 - https://www.mongodb.org/static/pgp/serv
er-5.0.asc | sudo apt-key add -
enrique@enrique-VirtualBox:~$ echo "deb [ arch=amd54, arm64 ] https://repo.mongo
db.org/apt/ubuntu focal/mongodb-org/5.0 multiverse" | sudo tee /etc/apt/sources.
list.d/mongodb-org-5.0.list
enrique@enrique-VirtualBox:~$ sudo apt-get update
enrique@enrique-VirtualBox:~$ sudo apt-get install -y mongodb-org
```

Figura A-6 Código de instalación de MongoDB

```
enrique@enrique-VirtualBox:~$ wget https://downloads.mongodb.com/compass/mongodb
-compass_1.30.1_amd64.deb
enrique@enrique-VirtualBox:~$ sudo dpkg -i mongodb-compass_1.30.1_amd64.deb
enrique@enrique-VirtualBox:~$ mongodb-compass
```

Figura A-7 Código de instalación Compass

ANEXO II: CÓDIGO MONGODB

```
import json
import pymongo

myclient = pymongo.MongoClient("localhost")
mydb = myclient["++"]
mycol = mydb["TFG"]

Weather={}
Weather['_id']='w'
Weather['Tipo']='Meteorología'
x = mycol.insert_one(Weather)

Estación={}
Estación['_id']='e'
Estación['Tipo']='Estación'
x = mycol.insert_one(Estación)

Boya={}
Boya['_id']='b'
Boya['Tipo']='Boya'
x = mycol.insert_one(Boya)



Temperatura={}
Temperatura['_id']='p_1'
Temperatura['Tipo']='Parámetro'
Temperatura['Medición']='Temperatura'
Temperatura['Unidad']='°'
x = mycol.insert_one(Temperatura)

Lluvia={}
Lluvia['_id']='p_2'
Lluvia['Tipo']='Parámetro'
Lluvia['Medición']='Lluvia'
Lluvia['Unidad']='L/m²'
x = mycol.insert_one(Lluvia)

Componente_Viento={}
Componente_Viento['_id']='p_3'
Componente_Viento['Tipo']='Parámetro'
Componente_Viento['Medición']='Componente del Viento'
Componente_Viento['Unidad']='°'
x = mycol.insert_one(Componente_Viento)

Intensidad_Viento={}
Intensidad_Viento['_id']='p_4'
Intensidad_Viento['Tipo']='Parámetro'
Intensidad_Viento['Medición']='Intensidad del Viento'
Intensidad_Viento['Unidad']='km/h'
x = mycol.insert_one(Intensidad_Viento)

Presión={}
Presión['_id']='p_5'
Presión['Tipo']='Parámetro'
Presión['Medición']='Presión'
```

```
Presión['Unidad']='hPa'
x = mycol.insert_one(Presión)

Humedad_Relativa={}
Humedad_Relativa['_id']='p_6'
Humedad_Relativa['Tipo']='Parámetro'
Humedad_Relativa['Medición']='Humedad Relativa'
Humedad_Relativa['Unidad']='%'
x = mycol.insert_one(Humedad_Relativa)

Temperatura_Agua={}
Temperatura_Agua['_id']='p_7'
Temperatura_Agua['Tipo']='Parámetro'
Temperatura_Agua['Medición']='Temperatura del agua'
Temperatura_Agua['Unidad']='º'
x = mycol.insert_one(Temperatura_Agua)

Longitud_Latitud={}
Longitud_Latitud['_id']='p_8'
Longitud_Latitud['Tipo']='Parámetro'
Longitud_Latitud['Medición']='Longitud y Latitud'
Longitud_Latitud['Unidad']='º'
x = mycol.insert_one(Longitud_Latitud)

Altura_Ola={}
Altura_Ola['_id']='p_9'
Altura_Ola['Tipo']='Parámetro'
Altura_Ola['Medición']='Altura de ola'
Altura_Ola['Unidad']='m'
x = mycol.insert_one(Altura_Ola)

Periodo_Medio_Oleaje={}
Periodo_Medio_Oleaje['_id']='p_10'
Periodo_Medio_Oleaje['Tipo']='Parámetro'
Periodo_Medio_Oleaje['Medición']='Periodo medio del oleaje'
Periodo_Medio_Oleaje['Unidad']='s'
x = mycol.insert_one(Periodo_Medio_Oleaje)

Componente_Oleaje={}
Componente_Oleaje['_id']='p_11'
Componente_Oleaje['Tipo']='Parámetro'
Componente_Oleaje['Medición']='Componente del oleaje'
Componente_Oleaje['Unidad']='º'
x = mycol.insert_one(Componente_Oleaje)

Estación_Marín = {}
Estación_Marín['_id']='E_M'
Estación_Marín['Tipo'] = 'Estación'
Estación_Marín['Localización'] = 'Marín'
Estación_Marín['Latitud'] = '42.39707'
Estación_Marín['Longitud'] = '-8.69362'
Estación_Marín['Unidad'] = 'º'
x = mycol.insert_one(Estación_Marín)

Estación_Sanxenxo = {}
Estación_Sanxenxo['_id']='E_S'
Estación_Sanxenxo['Tipo'] = 'Estación'
Estación_Sanxenxo['Localización'] = 'Sanxenxo'
Estación_Sanxenxo['Latitud'] = '42.40308'
Estación_Sanxenxo['Longitud'] = '-8.798792'
Estación_Sanxenxo['Unidad'] = 'º'
x = mycol.insert_one(Estación_Sanxenxo)
```

```
Estación_Cabo_Udra={}
Estación_Cabo_Udra['_id']='E_CU'
Estación_Cabo_Udra['Tipo'] = 'Estación'
Estación_Cabo_Udra['Localización'] = 'Cabo Udra'
Estación_Cabo_Udra['Latitud'] = '42.3396'
Estación_Cabo_Udra['Longitud'] = '-8.833798'
Estación_Cabo_Udra['Unidad'] = 'º'
x = mycol.insert_one(Estación_Cabo_Udra)

Estación_Vigo = {}
Estación_Vigo['_id']='E_V'
Estación_Vigo['Tipo'] = 'Estación'
Estación_Vigo['Localización'] = 'Vigo'
Estación_Vigo['Latitud'] = '42.24167'
Estación_Vigo['Longitud'] = '-8.727587'
Estación_Vigo['Unidad'] = 'º'
x = mycol.insert_one(Estación_Vigo)

Estación_Islas_Cíes = {}
Estación_Islas_Cíes['_id']='E_IC'
Estación_Islas_Cíes['Tipo'] = 'Estación'
Estación_Islas_Cíes['Localización'] = 'Islas Cíes'
Estación_Islas_Cíes['Latitud'] = '42.211796'
Estación_Islas_Cíes['Longitud'] = '-8.908423'
Estación_Islas_Cíes['Unidad'] = 'º'
x = mycol.insert_one(Estación_Islas_Cíes)

Estación_Bayona = {}
Estación_Bayona['Tipo'] = 'Estación'
Estación_Bayona['_id']='E_B'
Estación_Bayona['Localización'] = 'Bayona'
Estación_Bayona['Latitud'] = '42.1155'
Estación_Bayona['Longitud'] = '-8.837179'
Estación_Bayona['Unidad'] = 'º'
x = mycol.insert_one(Estación_Bayona)

Boya_Islas_Cíes = {}
Boya_Islas_Cíes['Tipo'] = 'Boya'
Boya_Islas_Cíes['_id']='B_IC'
Boya_Islas_Cíes['Localización'] = 'Islas Cíes'
x = mycol.insert_one(Boya_Islas_Cíes)

Boya_Norte_Pontevedra = {}
Boya_Norte_Pontevedra['Tipo'] = 'Boya'
Boya_Norte_Pontevedra['_id']='B_N_P'
Boya_Norte_Pontevedra['Localización'] = 'Entrada Norte Ría Pontevedra'
Boya_Norte_Pontevedra['Latitud'] = '42.417'
Boya_Norte_Pontevedra['Longitud'] = '-8.917'
Boya_Norte_Pontevedra['Unidad'] = 'º'
x = mycol.insert_one(Boya_Norte_Pontevedra)

Boya_Sur_Pontevedra = {}
Boya_Sur_Pontevedra['Tipo'] = 'Boya'
Boya_Sur_Pontevedra['_id']='B_S_P'
Boya_Sur_Pontevedra['Localización'] = 'Entrada Sur Ría Pontevedra'
Boya_Sur_Pontevedra['Latitud'] = '42.333'
Boya_Sur_Pontevedra['Longitud'] = '-8.917'
Boya_Sur_Pontevedra['Unidad'] = 'º'
x = mycol.insert_one(Boya_Sur_Pontevedra)

Boya_Sur_Islas_Cíes = {}
Boya_Sur_Islas_Cíes['Tipo'] = 'Boya'
```

```
Boya_Sur_Islas_Cíes['_id']='B_S_IC'
Boya_Sur_Islas_Cíes['Localización'] = 'Sur Islas Cíes'
Boya_Sur_Islas_Cíes['Latitud'] = '42.167'
Boya_Sur_Islas_Cíes['Longitud'] = '-8.917'
Boya_Sur_Islas_Cíes['Unidad'] = 'º'
x = mycol.insert_one(Boya_Sur_Islas_Cíes)

Boya_Oeste_Bayona = {}
Boya_Oeste_Bayona['Tipo'] = 'Boya'
Boya_Oeste_Bayona['_id']='B_S_IC'
Boya_Oeste_Bayona['Localización'] = 'Sur Islas Cíes'
Boya_Oeste_Bayona['Latitud'] = '42.125'
Boya_Oeste_Bayona['Longitud'] = '-8.917'
Boya_Oeste_Bayona['Unidad'] = 'º'
x = mycol.insert_one(Boya_Oeste_Bayona)



################---MARÍN--####################

with open('Dia.json') as file:
    data = json.load(file)

    ###############---M_LLUVIA---#########################
    i_l=1
    for resultado in data['resultados']:
        data3 = {}
        if resultado['Parámetro'] == 'Chuvia':
            data3['_id']="M_l_"+str(i_l)
            data3['Tipo'] = 'Lluvia'
            i_l=i_l+1
            data3['Fecha']=resultado['Data']
            data3['Valor']=resultado['Valor']
            data3['Unidad']= resultado['Unidades']
            x = mycol.insert_one(data3)

    ####################---M_TEMPERATURA---#########################
    i_t = 1
    for resultado in data['resultados']:
        data4 = {}
        if resultado['Parámetro'] == 'Temperatura media a 1.5m':
            data4['_id'] = "M_t_"+str(i_t)
            data4['Tipo'] = 'Temperatura'
            i_t = i_t + 1
            data4['Fecha'] = resultado['Data']
            data4['Valor'] = resultado['Valor']
            data4['Unidad'] = resultado['Unidades']
            x = mycol.insert_one(data4)

    ####################---M_PRESIÓN---#################################
    i_p = 1
    for resultado in data['resultados']:
        data5 = {}
        if resultado['Parámetro'] == 'Presión':
            data5['_id'] = "M_p_"+str(i_p)
            data5['Tipo'] = 'Presión'
            i_p = i_p + 1
            data5['Fecha'] = resultado['Data']
            data5['Valor'] = resultado['Valor']
            data5['Unidad'] = resultado['Unidades']
            x = mycol.insert_one(data5)
```

```
    ####################---M_COMPONENTE DEL VIENTO---####################

    i_cv = 1
    for resultado in data['resultados']:
        data6 = {}
        if resultado['Parámetro'] == 'Dirección do vento a 10m':
            data6['_id'] = "M_cv_"+str(i_cv)
            data6['Tipo'] = 'Componente del Viento'
            i_cv = i_cv + 1
            data6['Fecha'] = resultado['Data']
            data6['Valor'] = resultado['Valor']
            data6['Unidad'] = resultado['Unidades']
            x = mycol.insert_one(data6)

    ####################---M_INTENSIDAD DEL VIENTO---#########################

    i_iv = 1
    for resultado in data['resultados']:
        data7 = {}
        if resultado['Parámetro'] == 'Velocidade do vento a 10m':
            data7['_id'] = "M_iv_"+str(i_iv)
            data7['Tipo'] = 'Intensidad del Viento'
            i_iv = i_iv + 1
            data7['Fecha'] = resultado['Data']
            data7['Valor'] = resultado['Valor']
            data7['Unidad'] = resultado['Unidades']
            x = mycol.insert_one(data7)

    ####################---M_HUMEDAD RELATIVA---##############################
    i_h = 1
    for resultado in data['resultados']:
        data8 = {}
        if resultado['Parámetro'] == 'Humidade relativa media a 1.5m':
            data8['_id'] = "M_h_"+str(i_h)
            data8['Tipo'] = 'Humedad relativa'
            i_h = i_h + 1
            data8['Fecha'] = resultado['Data']
            data8['Valor'] = resultado['Valor']
            data8['Unidad'] = resultado['Unidades']
            x = mycol.insert_one(data8)



##################---CABO_UDRA--################################
with open('Cabo_Udra.json') as file:
    data = json.load(file)

    ##################---CU_LLUVIA---##############################
    i_l=1
    for resultado in data['resultados']:
        data3 = {}
        if resultado['Parámetro'] == 'Chuvia':
            data3['_id']="CU_l_"+str(i_l)
            data3['Tipo'] = 'Lluvia'
            i_l=i_l+1
            data3['Fecha']=resultado['Data']
            data3['Valor']=resultado['Valor']
            data3['Unidad']= resultado['Unidades']
            x = mycol.insert_one(data3)
```

```
####################---CU_TEMPERATURA---##############################
    i_t = 1
    for resultado in data['resultados']:
        data4 = {}
        if resultado['Parámetro'] == 'Temperatura media a 1.5m':
            data4['_id'] = "CU_t_"+str(i_t)
            data4['Tipo'] = 'Temperatura'
            i_t = i_t + 1
            data4['Fecha'] = resultado['Data']
            data4['Valor'] = resultado['Valor']
            data4['Unidad'] = resultado['Unidades']
            x = mycol.insert_one(data4)

    ############---CU_COMPONENTE DEL VIENTO---##################
    i_cv = 1
    for resultado in data['resultados']:
        data6 = {}
        if resultado['Parámetro'] == 'Dirección do vento a 10m':
            data6['_id'] = "CU_cv_"+str(i_cv)
            data6['Tipo'] = 'Componente del Viento'
            i_cv = i_cv + 1
            data6['Fecha'] = resultado['Data']
            data6['Valor'] = resultado['Valor']
            data6['Unidad'] = resultado['Unidades']
            x = mycol.insert_one(data6)

    ################---CU_INTENSIDAD DEL VIENTO---####################
    i_iv = 1
    for resultado in data['resultados']:
        data7 = {}
        if resultado['Parámetro'] == 'Velocidade do vento a 10m':
            data7['_id'] = "CU_iv_"+str(i_iv)
            data7['Tipo'] = 'Intensidad del Viento'
            i_iv = i_iv + 1
            data7['Fecha'] = resultado['Data']
            data7['Valor'] = resultado['Valor']
            data7['Unidad'] = resultado['Unidades']
            x = mycol.insert_one(data7)

    ##################---CU_HUMEDAD RELATIVA---##############################
    i_h = 1
    for resultado in data['resultados']:
        data8 = {}
        if resultado['Parámetro'] == 'Humidade relativa media a 1.5m':
            data8['_id'] = "CU_h_"+str(i_h)
            data8['Tipo'] = 'Humedad relativa'
            i_h = i_h + 1
            data8['Fecha'] = resultado['Data']
            data8['Valor'] = resultado['Valor']
            data8['Unidad'] = resultado['Unidades']
            x = mycol.insert_one(data8)



################---SANXENXO--##############################
with open('Sanxenxo.json') as file:
    data = json.load(file)

    #############---S_LLUVIA---############################
    i_l=1
    for resultado in data['resultados']:
        data3 = {}
```

```
        if resultado['Parámetro'] == 'Chuvia':
            data3['_id']="S_l_"+str(i_l)
            data3['Tipo'] = 'Lluvia'
            i_l=i_l+1
            data3['Fecha']=resultado['Data']
            data3['Valor']=resultado['Valor']
            data3['Unidad']= resultado['Unidades']
            x = mycol.insert_one(data3)

    ####################---S_TEMPERATURA---#############################
    i_t = 1
    for resultado in data['resultados']:
        data4 = {}
        if resultado['Parámetro'] == 'Temperatura media a 1.5m':
            data4['_id'] = "S_t_"+str(i_t)
            data4['Tipo'] = 'Temperatura'
            i_t = i_t + 1
            data4['Fecha'] = resultado['Data']
            data4['Valor'] = resultado['Valor']
            data4['Unidad'] = resultado['Unidades']
            x = mycol.insert_one(data4)

    ###################---S_PRESIÓN---#############################
    i_p = 1
    for resultado in data['resultados']:
        data5 = {}
        if resultado['Parámetro'] == 'Presión':
            data5['_id'] = "S_p_"+str(i_p)
            data5['Tipo'] = 'Presión'
            i_p = i_p + 1
            data5['Fecha'] = resultado['Data']
            data5['Valor'] = resultado['Valor']
            data5['Unidad'] = resultado['Unidades']
            x = mycol.insert_one(data5)

##################---S_COMPONENTE DEL VIENTO---#########################
    i_cv = 1
    for resultado in data['resultados']:
        data6 = {}
        if resultado['Parámetro'] == 'Dirección do vento a 10m':
            data6['_id'] = "S_cv_"+str(i_cv)
            data6['Tipo'] = 'Componente del Viento'
            i_cv = i_cv + 1
            data6['Fecha'] = resultado['Data']
            data6['Valor'] = resultado['Valor']
            data6['Unidad'] = resultado['Unidades']
            x = mycol.insert_one(data6)

    ###############---S_INTENSIDAD DEL VIENTO---#############################
    i_iv = 1
    for resultado in data['resultados']:
        data7 = {}
        if resultado['Parámetro'] == 'Velocidade do vento a 10m':
            data7['_id'] = "S_iv_"+str(i_iv)
            data7['Tipo'] = 'Intensidad del Viento'
            i_iv = i_iv + 1
            data7['Fecha'] = resultado['Data']
            data7['Valor'] = resultado['Valor']
            data7['Unidad'] = resultado['Unidades']
            x = mycol.insert_one(data7)
```

```
##################---S_HUMEDAD RELATIVA---#############################
    i_h = 1
    for resultado in data['resultados']:
        data8 = {}
        if resultado['Parámetro'] == 'Humidade relativa media a 1.5m':
            data8['_id'] = "S_h_"+str(i_h)
            data8['Tipo'] = 'Humedad relativa'
            i_h = i_h + 1
            data8['Fecha'] = resultado['Data']
            data8['Valor'] = resultado['Valor']
            data8['Unidad'] = resultado['Unidades']
            x = mycol.insert_one(data8)




##################---VIGO--#########################
with open('Vigo.json') as file:
    data = json.load(file)

    ##################---V_LLUVIA---#####################
    i_l=1
    for resultado in data['resultados']:
        data3 = {}
        if resultado['Parámetro'] == 'Chuvia':
            data3['_id']="V_l_"+str(i_l)
            data3['Tipo'] = 'Lluvia'
            i_l=i_l+1
            data3['Fecha']=resultado['Data']
            data3['Valor']=resultado['Valor']
            data3['Unidad']= resultado['Unidades']
            x = mycol.insert_one(data3)

    ################---V_TEMPERATURA---##################
    i_t = 1
    for resultado in data['resultados']:
        data4 = {}
        if resultado['Parámetro'] == 'Temperatura media a 1.5m':
            data4['_id'] = "V_t_"+str(i_t)
            data4['Tipo'] = 'Temperatura'
            i_t = i_t + 1
            data4['Fecha'] = resultado['Data']
            data4['Valor'] = resultado['Valor']
            data4['Unidad'] = resultado['Unidades']
            x = mycol.insert_one(data4)

    ##################---V_PRESIÓN---#####################
    i_p = 1
    for resultado in data['resultados']:
        data5 = {}
        if resultado['Parámetro'] == 'Presión':
            data5['_id'] = "V_p_"+str(i_p)
            data5['Tipo'] = 'Presión'
            i_p = i_p + 1
            data5['Fecha'] = resultado['Data']
            data5['Valor'] = resultado['Valor']
            data5['Unidad'] = resultado['Unidades']
            x = mycol.insert_one(data5)

    ##################---V_COMPONENTE DEL VIENTO---#####################
    i_cv = 1
```

```
    for resultado in data['resultados']:
        data6 = {}
        if resultado['Parámetro'] == 'Dirección do vento a 10m':
            data6['_id'] = "V_cv_"+str(i_cv)
            data6['Tipo'] = 'Componente del Viento'
            i_cv = i_cv + 1
            data6['Fecha'] = resultado['Data']
            data6['Valor'] = resultado['Valor']
            data6['Unidad'] = resultado['Unidades']
            x = mycol.insert_one(data6)

    ####################---V_INTENSIDAD DEL VIENTO---####################
    i_iv = 1
    for resultado in data['resultados']:
        data7 = {}
        if resultado['Parámetro'] == 'Velocidade do vento a 10m':
            data7['_id'] = "V_iv_"+str(i_iv)
            data7['Tipo'] = 'Intensidad del Viento'
            i_iv = i_iv + 1
            data7['Fecha'] = resultado['Data']
            data7['Valor'] = resultado['Valor']
            data7['Unidad'] = resultado['Unidades']
            x = mycol.insert_one(data7)

    ##################---V_HUMEDAD RELATIVA---#########################
    i_h = 1
    for resultado in data['resultados']:
        data8 = {}
        if resultado['Parámetro'] == 'Humidade relativa media a 1.5m':
            data8['_id'] = "V_h_"+str(i_h)
            data8['Tipo'] = 'Humedad relativa'
            i_h = i_h + 1
            data8['Fecha'] = resultado['Data']
            data8['Valor'] = resultado['Valor']
            data8['Unidad'] = resultado['Unidades']
            x = mycol.insert_one(data8)



##################---ISLAS_CÍES--########################
with open('Islas_Cíes.json') as file:
    data = json.load(file)

    ################---IC_LLUVIA---####################
    i_l=1
    for resultado in data['resultados']:
        data3 = {}
        if resultado['Parámetro'] == 'Chuvia':
            data3['_id']="IC_l_"+str(i_l)
            data3['Tipo'] = 'Lluvia'
            i_l=i_l+1
            data3['Fecha']=resultado['Data']
            data3['Valor']=resultado['Valor']
            data3['Unidad']= resultado['Unidades']
            x = mycol.insert_one(data3)

   ###################---IC_TEMPERATURA---####################
    i_t = 1
    for resultado in data['resultados']:
        data4 = {}
        if resultado['Parámetro'] == 'Temperatura media a 1.5m':
```

```
            data4['_id'] = "IC_t_"+str(i_t)
            data4['Tipo'] = 'Temperatura'
            i_t = i_t + 1
            data4['Fecha'] = resultado['Data']
            data4['Valor'] = resultado['Valor']
            data4['Unidad'] = resultado['Unidades']
            x = mycol.insert_one(data4)

    ################---IC_COMPONENTE DEL VIENTO---#######################
    i_cv = 1
    for resultado in data['resultados']:
        data6 = {}
        if resultado['Parámetro'] == 'Dirección do vento a 10m':
            data6['_id'] = "IC_cv_"+str(i_cv)
            data6['Tipo'] = 'Componente del Viento'
            i_cv = i_cv + 1
            data6['Fecha'] = resultado['Data']
            data6['Valor'] = resultado['Valor']
            data6['Unidad'] = resultado['Unidades']
            x = mycol.insert_one(data6)

    ################---IC_INTENSIDAD DEL VIENTO---#########################
    i_iv = 1
    for resultado in data['resultados']:
        data7 = {}
        if resultado['Parámetro'] == 'Velocidade do vento a 10m':
            data7['_id'] = "IC_iv_"+str(i_iv)
            data7['Tipo'] = 'Intensidad del Viento'
            i_iv = i_iv + 1
            data7['Fecha'] = resultado['Data']
            data7['Valor'] = resultado['Valor']
            data7['Unidad'] = resultado['Unidades']
            x = mycol.insert_one(data7)

    ##############---IC_HUMEDAD RELATIVA---#########################
    i_h = 1
    for resultado in data['resultados']:
        data8 = {}
        if resultado['Parámetro'] == 'Humidade relativa media a 1.5m':
            data8['_id'] = "IC_h_"+str(i_h)
            data8['Tipo'] = 'Humedad relativa'
            i_h = i_h + 1
            data8['Fecha'] = resultado['Data']
            data8['Valor'] = resultado['Valor']
            data8['Unidad'] = resultado['Unidades']
            x = mycol.insert_one(data8)



###################---BAYONA--#########################
with open('Baiona.json') as file:
    data = json.load(file)

    ################---B_LLUVIA---#####################
    i_l=1
    for resultado in data['resultados']:
        data3 = {}
        if resultado['Parámetro'] == 'Chuvia':
            data3['_id']="B_l_"+str(i_l)
            data3['Tipo'] = 'Lluvia'
            i_l=i_l+1
            data3['Fecha']=resultado['Data']
```

```
            data3['Valor']=resultado['Valor']
            data3['Unidad']= resultado['Unidades']
            x = mycol.insert_one(data3)

    ################---B_TEMPERATURA---####################
    i_t = 1
    for resultado in data['resultados']:
        data4 = {}
        if resultado['Parámetro'] == 'Temperatura media a 1.5m':
            data4['_id'] = "B_t_"+str(i_t)
            data4['Tipo'] = 'Temperatura'
            i_t = i_t + 1
            data4['Fecha'] = resultado['Data']
            data4['Valor'] = resultado['Valor']
            data4['Unidad'] = resultado['Unidades']
            x = mycol.insert_one(data4)

    ####################---B_COMPONENTE DEL VIENTO---#########################
    i_cv = 1
    for resultado in data['resultados']:
        data6 = {}
        if resultado['Parámetro'] == 'Dirección do vento a 10m':
            data6['_id'] = "B_cv_"+str(i_cv)
            data6['Tipo'] = 'Componente del Viento'
            i_cv = i_cv + 1
            data6['Fecha'] = resultado['Data']
            data6['Valor'] = resultado['Valor']
            data6['Unidad'] = resultado['Unidades']
            x = mycol.insert_one(data6)

    #################---B_INTENSIDAD DEL VIENTO---###################
    i_iv = 1
    for resultado in data['resultados']:
        data7 = {}
        if resultado['Parámetro'] == 'Velocidade do vento a 10m':
            data7['_id'] = "B_iv_"+str(i_iv)
            data7['Tipo'] = 'Intensidad del Viento'
            i_iv = i_iv + 1
            data7['Fecha'] = resultado['Data']
            data7['Valor'] = resultado['Valor']
            data7['Unidad'] = resultado['Unidades']
            x = mycol.insert_one(data7)

    #################---B_HUMEDAD RELATIVA---#########################
    i_h = 1
    for resultado in data['resultados']:
        data8 = {}
        if resultado['Parámetro'] == 'Humidade relativa media a 1.5m':
            data8['_id'] = "B_h_"+str(i_h)
            data8['Tipo'] = 'Humedad relativa'
            i_h = i_h + 1
            data8['Fecha'] = resultado['Data']
            data8['Valor'] = resultado['Valor']
            data8['Unidad'] = resultado['Unidades']
            x = mycol.insert_one(data8)
```

```
##########################################################################################
##########################################################################################
##########################################################################################
                                    ---BOYA METEOGALICIA--
##########################################################################################
##########################################################################################
##########################################################################################
with open('Boya1.json') as file:
    data = json.load(file)

    #########--Tª AGUA--###################
    i =1
    for resultado in data['Valores']:
        data3 = {}
        data3['Fecha']=resultado['@Data']
        for resultado in resultado['Medida']:
            data2 = {}
            if resultado['@Variable']== 'Temperatura (Superficie)':
                data2['_id']= 'B_IC_TA'+str(i)
                data2['Tipo']= 'Temperatura del Agua'
                data3['Valor']= resultado['@Valor']
                data2['Medidas']=data3
                i = i + 1
                x = mycol.insert_one(data2)

    #########--HUMEDAD--#########################
    i = 1
    for resultado in data['Valores']:
        data3 = {}
        data3['Fecha'] = resultado['@Data']
        for resultado in resultado['Medida']:
            data2 = {}
            if resultado['@Variable']== 'Humidade Relativa Media':
                data2['_id']= 'B_IC_H'+str(i)
                data2['Tipo']= 'Humedad relativa'
                data3['Valor']= resultado['@Valor']
                data2['Medidas']=data3
                i = i + 1
                x = mycol.insert_one(data2)

    ################--TEMPERATURA--#####################
    i = 1
    for resultado in data['Valores']:
        data3 = {}
        data3['Fecha'] = resultado['@Data']
        for resultado in resultado['Medida']:
            data2 = {}
            if resultado['@Variable']== 'Temperatura Media do aire':
                data2['_id']= 'B_IC_T'+str(i)
                data2['Tipo']= 'Temperatura'
                data3['Valor']= resultado['@Valor']
                data2['Medidas']=data3
                i = i + 1
                x = mycol.insert_one(data2)

    ###########--COMPONENTE DEL VIENTO--#################
    i = 1
    for resultado in data['Valores']:
        data3 = {}
        data3['Fecha'] = resultado['@Data']
        for resultado in resultado['Medida']:
            data2 = {}
            if resultado['@Variable']== 'Dirección do Vento':
```

```
                data2['_id']= 'B_IC_CV'+str(i)
                data2['Tipo']= 'Componente del Viento'
                data3['Valor']= resultado['@Valor']
                data2['Medidas']=data3
                i = i + 1
                x = mycol.insert_one(data2)

    #############--INTENSIDAD DEL VIENTO--####################
    i=1
    for resultado in data['Valores']:
        data3 = {}
        data3['Fecha'] = resultado['@Data']
        for resultado in resultado['Medida']:
            data2 = {}
            if resultado['@Variable']== 'Velocidade do Vento':
                data2['_id']= 'B_IC_IV'+str(i)
                data2['Tipo']= 'Intensidad del Viento'
                data3['Valor']= resultado['@Valor']
                data2['Medidas']=data3
                i=i+1
                x = mycol.insert_one(data2)


################################################################################
################################################################################
################################################################################
                                ---BATIMETRÍA--
################################################################################
################################################################################
################################################################################

with open('Rias_Baixas_2.json') as file:
    data = json.load(file)
    i =1
    for resultado in data['features']:
        data1={}
        data1=resultado['properties']
        data2={}
        data2['_id'] = 'BT_' + str(i)
        data2['Tipo'] = 'Batimería'
        data2['Longitud'] = data1['-16.25156250']
        data2['Latitud'] = data1['43.12656250']
        data2['Sonda'] = data1['-5135.80']
        i = i + 1
        x = mycol.insert_one(data2)

################################################################################
################################################################################
################################################################################
                        ---BOYAS_PUERTOS_DEL_ESTADO--
################################################################################
################################################################################
################################################################################
with open('Entrada_Norte_Pontevedra.json') as file:
    data = json.load(file)
    i= 0

    for resultado in data:
        data1 = {}
        data2 = {}
        data3 = {}
        if resultado['Valor nulo: -9999.9'] == ('2022 02 01 '+str(i).zfill(2)):
```

```
            data1['_id']='NP_AO_'+str(i)
            data1['Fecha']=resultado['Valor nulo: -9999.9']
            data1['Altura_Ola']=resultado['']
            data1['Unidades']='m'
            data2['_id'] = 'NP_PO_' + str(i)
            data2['Fecha'] = resultado['Valor nulo: -9999.9']
            data2['Periodo_Medio_Oleaje']=resultado['__1']
            data2['Unidades'] = 's'
            data3['_id'] = 'NP_CO_' + str(i)
            data3['Fecha'] = resultado['Valor nulo: -9999.9']
            data3['Componente_Oleaje']=resultado['__2']
            data3['Unidades'] = '°'
            x = mycol.insert_one(data1)
            x = mycol.insert_one(data2)
            x = mycol.insert_one(data3)
            i=i+1


with open('Entrada_Sur_Pontevedra.json') as file:
    data = json.load(file)
    i= 0
    for resultado in data:
        data1 = {}
        data2 = {}
        data3 = {}
        if resultado['Valor nulo: -9999.9'] == ('2022 02 01 '+str(i).zfill(2)):
            data1['_id'] = 'SP_AO_' + str(i)
            data1['Fecha'] = resultado['Valor nulo: -9999.9']
            data1['Altura_Ola'] = resultado['']
            data1['Unidades'] = 'm'
            data2['_id'] = 'SP_PO_' + str(i)
            data2['Fecha'] = resultado['Valor nulo: -9999.9']
            data2['Periodo_Medio_Oleaje'] = resultado['__1']
            data2['Unidades'] = 's'
            data3['_id'] = 'SP_CO_' + str(i)
            data3['Fecha'] = resultado['Valor nulo: -9999.9']
            data3['Componente_Oleaje'] = resultado['__2']
            data3['Unidades'] = '°'
            x = mycol.insert_one(data1)
            x = mycol.insert_one(data2)
            x = mycol.insert_one(data3)
            i = i + 1


with open('Sur_Islas_Cíes.json') as file:
    data = json.load(file)
    i= 0
    for resultado in data:
        data1 = {}
        data2 = {}
        data3 = {}
        if resultado['Valor nulo: -9999.9'] == ('2022 02 01 '+str(i).zfill(2)):
            data1['_id'] = 'SIC_AO_' + str(i)
            data1['Fecha'] = resultado['Valor nulo: -9999.9']
            data1['Altura_Ola'] = resultado['']
            data1['Unidades'] = 'm'
            data2['_id'] = 'SIC_PO_' + str(i)
            data2['Fecha'] = resultado['Valor nulo: -9999.9']
            data2['Periodo_Medio_Oleaje'] = resultado['__1']
            data2['Unidades'] = 's'
            data3['_id'] = 'SIC_CO_' + str(i)
            data3['Fecha'] = resultado['Valor nulo: -9999.9']
            data3['Componente_Oleaje'] = resultado['__2']
```

```
            data3['Unidades'] = 'º'
            x = mycol.insert_one(data1)
            x = mycol.insert_one(data2)
            x = mycol.insert_one(data3)
            i = i + 1


with open('Oeste_Bayona.json') as file:
    data = json.load(file)
    i= 0
    for resultado in data:
        data1 = {}
        data2 = {}
        data3 = {}
        if resultado['Valor nulo: -9999.9'] == ('2022 02 01 '+str(i).zfill(2)):
            data1['_id'] = 'OB_AO_' + str(i)
            data1['Fecha'] = resultado['Valor nulo: -9999.9']
            data1['Altura_Ola'] = resultado['']
            data1['Unidades'] = 'm'
            data2['_id'] = 'OB_PO_' + str(i)
            data2['Fecha'] = resultado['Valor nulo: -9999.9']
            data2['Periodo_Medio_Oleaje'] = resultado['__1']
            data2['Unidades'] = 's'
            data3['_id'] = 'OB_CO_' + str(i)
            data3['Fecha'] = resultado['Valor nulo: -9999.9']
            data3['Componente_Oleaje'] = resultado['__2']
            data3['Unidades'] = 'º'
            x = mycol.insert_one(data1)
            x = mycol.insert_one(data2)
            x = mycol.insert_one(data3)
            i = i + 1
```

ANEXO III: CÓDIGO ALLEGROGRAPH

```
from franz.openrdf.connect import ag_connect

conn = ag_connect('Gemelo', host='localhost', port='10035', user='test',
password='test')

# import os.path

# We assume that our data files live in this directory.
# DATA_DIR = '/home/mfgavilanes/PyCharm/GemeloTFG/AllegroGraph/data'
# path2 = os.path.join(DATA_DIR, 'mongo_tripletas.ntriples')


# from franz.openrdf.rio.rdfformat import RDFFormat

# conn.add(path2, base=None, format=RDFFormat.NTRIPLES, contexts=None)

print('Mongo triples (default graph): {count}'.format(count=conn.size('null')))


conn.addData("""
@prefix f: <http://www.franz.com/> .
@prefix id: <http://www.franz.com/id#> .

id:Meteorología f:hasMongoId "w" .
id:Temperatura  f:hasMongoId "p_1";
                f:subtype_of id:Meteorología .
id:Lluvia f:hasMongoId "p_2";
            f:subtype_of id:Meteorología  .
id:Componente_Viento f:hasMongoId "p_3";
                        f:subtype_of id:Meteorología .
id:Intensidad_Viento f:hasMongoId "p_4" ;
                    f:subtype_of id:Meteorología  .
id:Presión f:hasMongoId "p_5" ;
            f:subtype_of id:Meteorología .
id:Humedad_Relativa f:hasMongoId "p_6" ;
                    f:subtype_of id:Meteorología .
id:Temperatura_Agua f:hasMongoId "p_7" ;
                    f:subtype_of id:Meteorología .
id:Altura_Ola       f:hasMongoId "p_9" ;
                    f:subtype_of id:Meteorología .
id:Periodo_Medio_Oleaje f:hasMongoId "p_10" ;
                        f:subtype_of id:Meteorología .
id:Componente_Oleaje    f:hasMongoId "p_11" ;
                        f:subtype_of id:Meteorología .

id:Estación     f:hasMongoId "e".
id:Boya         f:hasMongoId "b" .
id:Estación_Marín f:hasMongoId "E_M".
id:Estación_Cabo_Udra f:hasMongoId "E_CU".
id:Estación_Sanxenxo f:hasMongoId "E_S".
id:Estación_Vigo f:hasMongoId "E_V".
id:Estación_Islas_Cíes f:hasMongoId "E_IC".
id:Estación_Bayona f:hasMongoId "E_B".
id:Boya_Islas_Cíes f:hasMongoID "B_IC".
id:Boya_Norte_Pontevedra f:hasMongoID "B_N_P".
id:Boya_Sur_Pontevedra f:hasMongoID "B_S_P".
id:Boya_Sur_Islas_Cíes f:hasMongoID "B_S_IC".
id:Boya_Oeste_Bayona f:hasMongoID "B_O_B".
```

```
""")


import json

####################---MARÍN--###############
with open('Dia.json') as file:
    data = json.load(file)

    i_l = 1
    for resultado in data['resultados']:

        if resultado['Parámetro'] == 'Chuvia':
            conn.addData("""
            @prefix f: <http://www.franz.com/> .
            @prefix id: <http://www.franz.com/id#> .

            id:Presión_Marín_"""+str(i_l)+"""    f:hasMongoId
\""""M_p_"+str(i_l)+"""\" ;
                                f:value_of id:Presión .

            id:Lluvia_Marín_"""+str(i_l)+""" f:hasMongoId
\""""M_l_"+str(i_l)+"""\" ;
                                f:value_of id:Lluvia .

            id:Componente_Viento_Marín_"""+str(i_l)+""" f:hasMongoId
\""""M_cv_"+str(i_l)+"""\" ;
                                f:value_of id:Componente_Viento .

            id:Intensidad_Viento_Marín_"""+str(i_l)+""" f:hasMongoId
\""""M_iv_"+str(i_l)+"""\" ;
                                f:value_of id:Intensidad_Viento .

            id:Temperatura_Marín_"""+str(i_l)+""" f:hasMongoId
\""""M_t_"+str(i_l)+"""\" ;
                                f:value_of id:Temperatura .

            id:Humedad_Relativa_Marín_"""+str(i_l)+""" f:hasMongoId
\""""M_h_"+str(i_l)+"""\" ;
                                f:value_of id:Humedad_Relativa .

            id:Estación_Marín
                                    f:location_of id:Presión_Marín_"""+str(i_l)+""";
                                    f:location_of id:Lluvia_Marín_"""+str(i_l)+""";
                                     f:location_of
id:Componente_Viento_Marín_"""+str(i_l)+""";
                                       f:location_of
id:Intensidad_Viento_Marín_"""+str(i_l)+""";
                                        f:location_of
id:Temperatura_Marín_"""+str(i_l)+""";
                                         f:location_of
id:Humedad_Relativa_Marín_"""+str(i_l)+""".
                                """)
            i_l=i_l+1



######################---CABO_UDRA--#######################
with open('Cabo_Udra.json') as file:
    data = json.load(file)

    i_l = 1
```

```
    for resultado in data['resultados']:

        if resultado['Parámetro'] == 'Chuvia':
            conn.addData("""
            @prefix f: <http://www.franz.com/> .
            @prefix id: <http://www.franz.com/id#> .
            id:Presión_Cabo_Udra_"""+ str(i_l) + """   f:hasMongoId
\"""""CU_p_"+str(i_l)+"""\" ;
                                    f:value_of id:Presión .

            id:Lluvia_Cabo_Udra_"""+ str(i_l) + """ f:hasMongoId
\"""""CU_l_"+str(i_l)+"""\" ;
                                    f:value_of id:Lluvia .

            id:Componente_Viento_Cabo_Udra_"""+ str(i_l) + """ f:hasMongoId
\"""""CU_cv_"+str(i_l)+"""\" ;
                                    f:value_of id:Componente_Viento .

            id:Intensidad_Viento_Cabo_Udra_"""+ str(i_l) + """ f:hasMongoId
\"""""CU_iv_"+str(i_l)+"""\" ;
                                    f:value_of id:Intensidad_Viento .

            id:Temperatura_Cabo_Udra_"""+ str(i_l) + """ f:hasMongoId
\"""""CU_t_"+str(i_l)+"""\" ;
                                    f:value_of id:Temperatura .

            id:Humedad_Relativa_Cabo_Udra_"""+ str(i_l) + """ f:hasMongoId
\"""""CU_h_"+str(i_l)+"""\" ;
                                    f:value_of id:Humedad_Relativa .

            id:Estación_Cabo_Udra
                                    f:location_of id:Presión_Cabo_Udra_"""+str(i_l)+""";
                                    f:location_of id:Lluvia_Cabo_Udra_"""+str(i_l)+""";
                                     f:location_of
id:Componente_Viento_Cabo_Udra_"""+str(i_l)+""";
                                      f:location_of
id:Intensidad_Viento_Cabo_Udra_"""+str(i_l)+""";
                                       f:location_of id:Temperatura_Cabo_Udra_"""+str(i_l)+""";
                                        f:location_of
id:Humedad_Relativa_Cabo_Udra_"""+str(i_l)+""".
                                    """)
            i_l=i_l+1



######################---SANXENXO--#################
with open('Sanxenxo.json') as file:
    data = json.load(file)

    i_l = 1
    for resultado in data['resultados']:
        if resultado['Parámetro'] == 'Chuvia':
            conn.addData("""
            @prefix f: <http://www.franz.com/> .
            @prefix id: <http://www.franz.com/id#> .
            id:Presión_Sanxenxo        f:hasMongoId \"""""S_p_"+str(i_l)+"""\" ;
                                        f:value_of id:Presión .

            id:Lluvia_Sanxenxo_"""+ str(i_l) + """ f:hasMongoId
\"""""S_l_"+str(i_l)+"""\" ;
                                    f:value_of id:Lluvia .

            id:Componente_Viento_Sanxenxo_"""+ str(i_l) + """  f:hasMongoId
\"""""S_cv_"+str(i_l)+"""\" ;
                                    f:value_of id:Componente_Viento .
```

```
            id:Intensidad_Viento_Sanxenxo_"""+ str(i_l) + """  f:hasMongoId
\"""""S_iv_"+str(i_l)+"""\" ;
                                f:value_of id:Intensidad_Viento .

             id:Temperatura_Sanxenxo_"""+ str(i_l) + """  f:hasMongoId
\"""""S_t_"+str(i_l)+"""\" ;
                                f:value_of id:Temperatura .

            id:Humedad_Relativa_Sanxenxo_"""+ str(i_l) + """  f:hasMongoId
\"""""S_h_"+str(i_l)+"""\" ;
                                f:value_of id:Humedad_Relativa .

            id:Estación_Sanxenxo
                                f:location_of id:Presión_Sanxenxo"""+str(i_l)+""";
                                f:location_of id:Lluvia_Sanxenxo_"""+str(i_l)+""";
                                 f:location_of
id:Componente_Viento_Sanxenxo_"""+str(i_l)+""";
                                  f:location_of
id:Intensidad_Viento_Sanxenxo_"""+str(i_l)+""";
                                   f:location_of
id:Temperatura_Sanxenxo_"""+str(i_l)+""";
                                    f:location_of
id:Humedad_Relativa_Sanxenxo_"""+str(i_l)+""".
                                """)
            i_l=i_l+1



######################---VIGO--####################

with open('Vigo.json') as file:
    data = json.load(file)

    i_l = 1
    for resultado in data['resultados']:
        data3 = {}
        if resultado['Parámetro'] == 'Chuvia':
            conn.addData("""
            @prefix f: <http://www.franz.com/> .
            @prefix id: <http://www.franz.com/id#> .
            id:Presión_Vigo_"""+ str(i_l) + """     f:hasMongoId
\"""""V_p_"+str(i_l)+"""\" ;
                                f:value_of id:Presión .

            id:Lluvia_Vigo_"""+ str(i_l) + """      f:hasMongoId
\"""""V_l_"+str(i_l)+"""\" ;
                                f:value_of id:Lluvia .

            id:Componente_Viento_Vigo_"""+ str(i_l) + """      f:hasMongoId
\"""""V_cv_"+str(i_l)+"""\" ;
                                f:value_of id:Componente_Viento .

            id:Intensidad_Viento_Vigo_"""+ str(i_l) + """      f:hasMongoId
\"""""V_iv_"+str(i_l)+"""\" ;
                                f:value_of id:Intensidad_Viento .

            id:Temperatura_Vigo_"""+ str(i_l) + """      f:hasMongoId
\"""""V_t_"+str(i_l)+"""\" ;
                                f:value_of id:Temperatura .

            id:Humedad_Relativa_Vigo_"""+ str(i_l) + """      f:hasMongoId
\"""""V_h_"+str(i_l)+"""\" ;
                                f:value_of id:Humedad_Relativa .
```

```
            id:Estación_Vigo
                                    f:location_of id:Presión_Vigo_"""+str(i_l)+""";
                                    f:location_of id:Lluvia_Vigo_"""+str(i_l)+""";
                                     f:location_of
id:Componente_Viento_Vigo_"""+str(i_l)+""";
                                      f:location_of
id:Intensidad_Viento_Vigo_"""+str(i_l)+""";
                                       f:location_of
id:Temperatura_Vigo_"""+str(i_l)+""";
                                        f:location_of
id:Humedad_Relativa_Vigo_"""+str(i_l)+""".
                                    """)
            i_l=i_l+1



############################---ISLAS_CÍES--##########################
with open('Islas_Cíes.json') as file:
    data = json.load(file)

    i_l = 1
    for resultado in data['resultados']:
        data3 = {}
        if resultado['Parámetro'] == 'Chuvia':
            conn.addData("""
            @prefix f: <http://www.franz.com/> .
            @prefix id: <http://www.franz.com/id#> .
            id:Presión_Islas_Cíes_"""+ str(i_l) + """    f:hasMongoId
\"""""IC_p_"+str(i_l)+"""\" ;
                                     f:value_of id:Presión .

            id:Lluvia_Islas_Cíes_"""+ str(i_l) + """    f:hasMongoId
\"""""IC_l_"+str(i_l)+"""\" ;
                                    f:value_of id:Lluvia .

            id:Componente_Viento_Islas_Cíes_"""+ str(i_l) + """    f:hasMongoId
\"""""IC_cv_"+str(i_l)+"""\" ;
                                    f:value_of id:Componente_Viento .

            id:Intensidad_Viento_Islas_Cíes_"""+ str(i_l) + """    f:hasMongoId
\"""""IC_iv_"+str(i_l)+"""\" ;
                                    f:value_of id:Intensidad_Viento .

            id:Temperatura_Islas_Cíes_"""+ str(i_l) + """    f:hasMongoId
\"""""IC_t_"+str(i_l)+"""\" ;
                                    f:value_of id:Temperatura .

            id:Humedad_Relativa_Islas_Cíes_"""+ str(i_l) + """    f:hasMongoId
\"""""IC_h_"+str(i_l)+"""\" ;
                                    f:value_of id:Humedad_Relativa .

            id:Estación_Islas_Cíes
                                    f:location_of
id:Presión_Islas_Cíes_"""+str(i_l)+""";
                                    f:location_of
id:Lluvia_Islas_Cíes_"""+str(i_l)+""";
                                     f:location_of
id:Componente_Viento_Islas_Cíes_"""+str(i_l)+""";
                                      f:location_of
id:Intensidad_Viento_Islas_Cíes_"""+str(i_l)+""";
                                       f:location_of
```

```
id:Temperatura_Islas_Cíes_"""+str(i_l)+""";
                                    f:location_of
id:Humedad_Relativa_Islas_Cíes_"""+str(i_l)+""".
                                    """)
            i_l=i_l+1


######################---BAYONA--#####################
with open('Baiona.json') as file:
    data = json.load(file)

    i_l = 1
    for resultado in data['resultados']:
        if resultado['Parámetro'] == 'Chuvia':
            conn.addData("""
            @prefix f: <http://www.franz.com/> .
            @prefix id: <http://www.franz.com/id#> .
            id:Presión_Baiona_"""+ str(i_l) + """    f:hasMongoId
\""""B_p_"+str(i_l)+"""\" ;
                                f:value_of id:Presión .

            id:Lluvia_Baiona_"""+ str(i_l) + """    f:hasMongoId
\""""B_l_"+str(i_l)+"""\" ;
                                f:value_of id:Lluvia .

            id:Componente_Viento_Baiona_"""+ str(i_l) + """    f:hasMongoId
\""""B_cv_"+str(i_l)+"""\" ;
                                f:value_of id:Componente_Viento .

            id:Intensidad_Viento_Baiona_"""+ str(i_l) + """    f:hasMongoId
\""""B_iv_"+str(i_l)+"""\" ;
                                f:value_of id:Intensidad_Viento .

            id:Temperatura_Baiona_"""+ str(i_l) + """    f:hasMongoId
\""""B_t_"+str(i_l)+"""\" ;
                                f:value_of id:Temperatura .

            id:Humedad_Relativa_Baiona_"""+ str(i_l) + """    f:hasMongoId
\""""B_h_"+str(i_l)+"""\" ;
                                f:value_of id:Humedad_Relativa .

            id:Estación_Bayona
                                f:location_of id:Presión_Baiona_"""+str(i_l)+""";
                                f:location_of id:Lluvia_Baiona_"""+str(i_l)+""";
                                 f:location_of
id:Componente_Viento_Baiona_"""+str(i_l)+""";
                                   f:location_of
id:Intensidad_Viento_Baiona_"""+str(i_l)+""";
                                    f:location_of
id:Temperatura_Baiona_"""+str(i_l)+""";
                                     f:location_of
id:Humedad_Relativa_Baiona_"""+str(i_l)+""".
                                    """)
            i_l=i_l+1

######################---BOYA_ISLAS_CÍES--####################
with open('Boya1.json') as file:
    data = json.load(file)
    i_l=1
    for resultado in data['Valores']:

        for resultado in resultado['Medida']:
```

```
            if resultado['@Variable'] == 'Temperatura Media do aire':
                conn.addData("""
                @prefix f: <http://www.franz.com/> .
                @prefix id: <http://www.franz.com/id#> .
                id:T_Agua_Boya_Cíes_""" + str(i_l) + """    f:hasMongoId
\"""""B_IC_TA" + str(i_l) + """\" ;
                                     f:value_of id:Temperatura_Agua .

                id:Componente_Viento_Boya_Cíes""" + str(i_l) + """    f:hasMongoId
\"""""B_IC_CV" + str(i_l) + """\" ;
                                    f:value_of id:Componente_Viento .

                id:Intensidad_Viento_Boya_Cíes""" + str(i_l) + """    f:hasMongoId
\"""""B_IC_IV" + str(i_l) + """\" ;
                                    f:value_of id:Intensidad_Viento .

                id:Temperatura_Boya_Cíes""" + str(i_l) + """    f:hasMongoId
\"""""B_IC_T" + str(i_l) + """\" ;
                                    f:value_of id:Temperatura .

                id:Humedad_Relativa_Boya_Cíes""" + str(i_l) + """    f:hasMongoId
\"""""B_IC_H" + str(i_l) + """\" ;
                                    f:value_of id:Humedad_Relativa .

                id:Boya_Islas_Cíes
                                    f:location_of id:T_Agua_Boya_Cíes_""" +
str(i_l) + """;
                                    f:location_of
id:Componente_Viento_Boya_Cíes""" + str(i_l) + """;
                                     f:location_of
id:Intensidad_Viento_Boya_Cíes""" + str(i_l) + """;
                                       f:location_of id:Temperatura_Boya_Cíes""" +
str(i_l) + """;
                                         f:location_of
id:Humedad_Relativa_Boya_Cíes""" + str(i_l) + """.

                                    """)
                i_l = i_l + 1


conn.addData("""
            @prefix f: <http://www.franz.com/> .
@prefix id: <http://www.franz.com/id#> .
id:Estación_Marín      f:type_of id:Estación.
id:Estación_Cabo_Udra  f:type_of id:Estación.
id:Estación_Sanxenxo   f:type_of id:Estación.
id:Estación_Vigo       f:type_of id:Estación.
id:Estación_Islas_Cíes f:type_of id:Estación.
id:Estación_Bayona     f:type_of id:Estación.
id:Boya_Islas_Cíes     f:type_of id:Boya.
id:Boya_Norte_Pontevedra    f:type_of id:Boya.
id:Boya_Sur_Pontevedra      f:type_of id:Boya.
id:Boya_Sur_Islas_Cíes      f:type_of id:Boya.
id:Boya_Oeste_Bayona        f:type_of id:Boya.

""")
```

```
##############################################################################################
##############################################################################################
##############################################################################################
                                    ---BATIMETRÍA--
##############################################################################################
##############################################################################################
##############################################################################################
with open('Rias_Baixas_2.json') as file:
    data = json.load(file)
    i =1

    for resultado in data['features']:
        conn.addData("""
        @prefix f: <http://www.franz.com/> .
        @prefix id: <http://www.franz.com/id#> .
        id:Batimetría_"""+str(i)+"""    f:hasMongoID \"""""BT_" + str(i) +
"""\"."""")
        #i=i+1



##############################################################################################
##############################################################################################
##############################################################################################
                          ---BOYAS_PUERTOS_DEL_ESTADO--
##############################################################################################
##############################################################################################
##############################################################################################
with open('Entrada_Norte_Pontevedra.json') as file:
    data = json.load(file)
    i = 0

    for resultado in data:

        if resultado['Valor nulo: -9999.9'] == ('2022 02 01 ' + str(i).zfill(2)):
            conn.addData("""
            @prefix f: <http://www.franz.com/> .
            @prefix id: <http://www.franz.com/id#> .

            id:Norte_Pontevedra_Amplitud_Ola_""" + str(i) + """    f:hasMongoId
\""""""NP_AO_" + str(i) + """\" ;
                                    f:value_of id:Altura_Ola .

            id:Norte_Pontevedra_Periodo_Medio_Oleaje_""" + str(i) + """
f:hasMongoId \""""""NP_PO_" + str(i) + """\" ;
                                f:value_of id:Periodo_Medio_Oleaje .

            id:Norte_Pontevedra_Componente_Oleaje_""" + str(i) + """
f:hasMongoId \""""""NP_CO_" + str(i) + """\" ;
                                    f:value_of id:Componente_Oleaje .



            id:Boya_Norte_Pontevedra
                                    f:location_of id:Norte_Pontevedra_Amplitud_Ola_"""
+ str(i) + """;
                                    f:location_of
id:Norte_Pontevedra_Periodo_Medio_Oleaje_""" + str(i) + """;
                                     f:location_of
id:Norte_Pontevedra_Componente_Oleaje_""" + str(i) + """.
                                """)
            i = i + 1
```

```
with open('Entrada_Sur_Pontevedra.json') as file:
    data = json.load(file)
    i = 0

    for resultado in data:

        if resultado['Valor nulo: -9999.9'] == ('2022 02 01 ' + str(i).zfill(2)):
            conn.addData("""
            @prefix f: <http://www.franz.com/> .
            @prefix id: <http://www.franz.com/id#> .

            id:Sur_Pontevedra_Amplitud_Ola_""" + str(i) + """    f:hasMongoId
\"""""SP_AO_" + str(i) + """\" ;
                                f:value_of id:Altura_Ola .

            id:Sur_Pontevedra_Periodo_Medio_Oleaje_""" + str(i) + """ f:hasMongoId
\"""""SP_PO_" + str(i) + """\" ;
                            f:value_of id:Periodo_Medio_Oleaje .

            id:Sur_Pontevedra_Componente_Oleaje_""" + str(i) + """    f:hasMongoId
\"""""SP_CO_" + str(i) + """\" ;
                                f:value_of id:Componente_Oleaje .



            id:Boya_Sur_Pontevedra
                                f:location_of id:Norte_Pontevedra_Amplitud_Ola_"""
+ str(i) + """;
                                f:location_of
id:Norte_Pontevedra_Periodo_Medio_Oleaje_""" + str(i) + """;
                                 f:location_of
id:Norte_Pontevedra_Componente_Oleaje_""" + str(i) + """.
                            """)
            i = i + 1

with open('Sur_Islas_Cíes.json') as file:
    data = json.load(file)
    i = 0

    for resultado in data:

        if resultado['Valor nulo: -9999.9'] == ('2022 02 01 ' + str(i).zfill(2)):
            conn.addData("""
            @prefix f: <http://www.franz.com/> .
            @prefix id: <http://www.franz.com/id#> .

            id:Sur_Islas_Cíes_Amplitud_Ola_""" + str(i) + """    f:hasMongoId
\"""""SIC_AO_" + str(i) + """\" ;
                                f:value_of id:Altura_Ola .

            id:Sur_Islas_Cíes_Periodo_Medio_Oleaje_""" + str(i) + """ f:hasMongoId
\"""""SIC_PO_" + str(i) + """\" ;
                            f:value_of id:Periodo_Medio_Oleaje .

            id:Sur_Islas_Cíes_Componente_Oleaje_""" + str(i) + """    f:hasMongoId
\"""""SIC_CO_" + str(i) + """\" ;
                                f:value_of id:Componente_Oleaje .



            id:Boya_Sur_Islas_Cíes
```

```
                                f:location_of id:Sur_Islas_Cíes_Amplitud_Ola_""" +
str(i) + """;
                                f:location_of
id:Sur_Islas_Cíes_Periodo_Medio_Oleaje_""" + str(i) + """;
                                 f:location_of
id:Sur_Islas_Cíes_Componente_Oleaje_""" + str(i) + """.
                            """)
            i = i + 1

with open('Oeste_Bayona.json') as file:
    data = json.load(file)
    i = 0

    for resultado in data:

        if resultado['Valor nulo: -9999.9'] == ('2022 02 01 ' + str(i).zfill(2)):
            conn.addData("""
            @prefix f: <http://www.franz.com/> .
            @prefix id: <http://www.franz.com/id#> .

            id:Oeste_Bayona_Amplitud_Ola_""" + str(i) + """    f:hasMongoId
\"""""OB_AO_" + str(i) + """\" ;
                                f:value_of id:Altura_Ola .

            id:Oeste_Bayona_Periodo_Medio_Oleaje_""" + str(i) + """ f:hasMongoId
\"""""OB_PO_" + str(i) + """\" ;
                            f:value_of id:Periodo_Medio_Oleaje .

            id:Oeste_Bayona_Componente_Oleaje_""" + str(i) + """    f:hasMongoId
\"""""OB_CO_" + str(i) + """\" ;
                                f:value_of id:Componente_Oleaje .



            id:Boya_Oeste_Bayona
                                f:location_of id:Sur_Islas_Cíes_Amplitud_Ola_""" +
str(i) + """;
                                f:location_of
id:Sur_Islas_Cíes_Periodo_Medio_Oleaje_""" + str(i) + """;
                                f:location_of
id:Sur_Islas_Cíes_Componente_Oleaje_""" + str(i) + """.
                            """)
            i = i + 1



print('Mongo triples (default graph): {count}'.format(count=conn.size('null')))
```