UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica

Grado en Ingeniería Informática en Ingeniería del Software

Trabajo Fin de Grado

Aplicación Web para el Almacenamiento y Visualización de Geodatos Meteorológicos mediante Spring y MongoDB. Análisis de Técnicas de Indexación NoSQL.

DAVID SANJUÁN LÓPEZ

Julio, 2015

UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica

Grado en Ingeniería Informática en Ingeniería del Software

Trabajo Fin de Grado

Aplicación Web para el Almacenamiento y Visualización de Geodatos Meteorológicos mediante Spring y MongoDB. Análisis de Técnicas de Indexación NoSQL.

Autor: David Sanjuán López

Fdo.:

Director: Félix Rodríguez Rodríguez

Fdo.:

Tribunal Calificador

Presidente: Antonio Polo Márquez

Fdo.:

Secretario: Antonio J. Plaza Miguel

Fdo.:

Vocal: Miryam J. Salas Sánchez

Fdo.:

CALIFICACIÓN:

FECHA:

Eternamente agradecido a mi profesor, tutor y maestro Félix Rodriguez, por la dedicación, comprensión, ayuda y apoyo moral brindado y el cual me ha permitido no solo estar aquí, sino además elaborar este proyecto. Esperando volver a encontrarnos en un futuro.

A todos aquellos profesores, que de alguna manera, sembraron en mí la semilla del conocimiento.

Y sin duda alguna, a ellos, mis padres, porque sin ellos nada de esto tendría sentido y a los cuales les adeudo la paciencia de tolerarme las espinas más agudas, las negligencias, las vanidades, los temores y la dudas.

¿Y por qué no? A ella, en ocasiones musa de mi inspiración.

ÍNDICE DE CONTENIDO

Ilustración 1: Fotografía de la escuela politécnica - Universidad de Extremadura

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

1 RESUMEN

Palabras clave: Aplicación Web 2.0, Spring framework, MongoDB, técnicas de indexación, Openlayers, Modelo Vista Controlador, geodatos QuikSCAT, Spring Tool Suite.

La realización de este proyecto de fin de grado pretende, por un lado, plasmar los conocimientos adquiridos en los cuatro años de estudios en el grado de Ingeniería Informática en Ingeniería del Software, y así como en lo posible, tratar de explorar y añadir nuevos conocimientos en materia de desarrollo de aplicaciones Web y estrategias de optimización e indexación en bases de datos. El objetivo principal del trabajo es el desarrollo de una aplicación y estudiar las distintas **estrategias de indexación** para datos geoespaciales. Para ello se han empleado tecnologías inscritas dentro del marco conocido como Web 2.0 y las bases de datos definidos como **NoSQL**.

Por un lado, el proyecto que he realizado facilita la **exploración y visualización** de datos geoespaciales. El proyecto se trata de una **aplicación Web** accesible desde cualquier tipo de navegador y equipo, incluyendo dispositivos móviles. El objetivo es proporcionar una **herramienta gráfica**, sencilla y con gran poder computacional que pueda ser usada por la comunidad científica para el estudio de los datos marítimos.

Por otro lado, los datos geoespaciales se procesan, se almacenan y se indexan en una base de datos de tipo NoSQL implementada; esta base de datos se trata de **MongoDB**. MongoDB es el repositorio y el corazón de la aplicación. Sobre este almacén de datos se describen las diversas estrategias de optimización del rendimiento en la consulta de los datos. Para ello, se definirán y estudiarán las posibles formas de indexación, ofreciendo una comparativa detallada de cada una de las estrategias posibles.

La aplicación Web permite la visualización de datos geoespaciales de manera gráfica, pudiendo realizar exploraciones por rango de fecha y regiones terrestres. Además, clasifica los resultados mostrados en categorías de acuerdo a la naturaleza de los datos para mejorar la visualización.

Los datos geoespaciales de los que hace uso la aplicación son propiedad de la **NASA** y fueron generados por el satélite ***QuikSCAT***. De toda la información recogida por el satélite, y para simplificar, se ha seleccionado la información relativa a la velocidad del

viento, a su dirección y al grado de precipitaciones sobre puntos de observación marinos separados cada veinticinco kilómetros cuadrados.

Para el desarrollo de la aplicación Web se han seguido diversos patrones y esquemas de diseño. Como patrón de diseño principal se ha utilizado el patrón **Modelo Vista Controlador** (MVC); se trata de una arquitectura estándar para aplicaciones interactivas, que separa los datos y la lógica de negocio de una aplicación de las vistas del usuario. Este patrón de diseño facilita el desarrollo y posterior mantenimiento de las aplicaciones gracias a la reutilización de código y a la separación de conceptos.

Además se han utilizado diversos *frameworks* de trabajo, entre los que destaca **Spring Framework**. Spring Framework es un marco de aplicaciones Java que facilita enormemente la implementación de distintos patrones de diseño y la integración de una gran cantidad de tecnologías de gran relevancia.

Con este trabajo de fin de grado, he querido construir una aplicación Web que fuera original y útil, cuyos procesos de negocio ayudaran en la medida de lo posible a entender algo tan importante como el planeta Tierra donde vivimos, dando oportunidad a los usuarios de la aplicación Web, tanto públicos como privados, científicos o curiosos, de obtener información meteorológica terrestre sobre una situación geográfica marítima y, por qué no, de recuperar información que pueda ayudar a entender y prevenir desastres naturales, entre otras posibilidades.

2 ABSTRACT

Keywords: Application Web 2.0, Spring framework, MongoDB, Indexing techniques, Openlayers, Model View Controller, geodates QuikSCAT, Spring Tool Suite.

This degree dissertation aims, to make use of the knowledge acquired all the four years of Computer Science Engineering degree, and also to explore new knowledge for Web applications development and database optimization strategies and indexing. The main goal of this work is to develop a Web site studying different **indexing strategies** for geospatial data. Thus, both a framework of Web 2.0 and **NoSQL** database technologies have been developed.

This work facilitates, on the one hand, both the **exploration and visualization** of geospatial data. The project is about an accessible **web application** from any browser and equipment, including mobile devices. The aim is to provide a simple **graphical tool** with great computational power to be used by the scientific community on marine data studies.

On the other hand, the geospatial data are stored and indexed in a NoSQL; this database is **MongoDB**. MongoDB is both the data repository and the application core. According to this kind of data storing, some optimization strategies of data access performance are described. Thus, practically all possible indexing ways have been explored, providing a detailed comparison of each possible strategy.

The Web application allows the graphical visualization of the geospatial data, and it can perform scans by both date range and regions of earth. In addition, and in order to improve the visualization process, the query results are classified by means of its spatial range, according to the nature of the data.

The **NASA QuikSCAT** satellite is the source of our geospatial data. Based on the main idea of simplicity, the QuikSCAT ocean data has been filtered in order to select three basic attributes every 25 square kilometers: the wind speed (meters per second), the wind direction (degrees) and rain rate (millimeters per hour).

Regarding to the application development, different patterns and design schemes have been performed. The **Model View Controller** (MVC) schema has been used as main design pattern to implement the Web application; this is a standard architecture for

interactive applications, which separates data and business logic of an application user views. This design pattern facilitates the development and subsequent maintenance of applications through code reuse and concepts division.

Furthermore, diverse frameworks have been used. Among them, the **Spring Framework**. Spring Framework is a Java application framework that greatly facilitates both the implementation and integration of some design patterns with lots of technologies.

A full implementation of all the constituent parts has been developed.

With this degree dissertation I have built an original and useful Web application, whose business processes help us to understand, as far as possible, something as important as the planet Earth where we live, giving the opportunity to users, both public and private, scientists, or curious people, to obtain information about ocean geospatial data by means of a Web application, and, why not, to retrieve information that could help to understand and prevent natural disasters, among others possibilities.

3 INTRODUCCIÓN

3.1 FINALIDAD DEL PROYECTO

El objetivo principal del trabajo es el desarrollo de un portal Web siguiendo las directrices Web 2.0 y estudiar las distintas estrategias de indexación para datos geoespaciales.

La Web 2.0 es una "denominación de origen" que se refiere a una segunda generación en la historia de los sitios Web. Cuando las páginas ofrecen un nivel considerable de interacción y se actualizan con los aportes de los usuarios, se habla de Web 2.0. Sin duda, la consideración principal de Web 2.0 es que la aplicación debe estar orientada por y para los usuarios, ofreciendo contenido de forma interactiva.

Para cumplir con esta especificación se utilizará, en la parte servidora, el patrón de diseño Modelo Vista Controlador implementado con Spring MVC. Desarrollando las vistas mediante paginas JSP (Java Server Pages), una tecnología que ayuda a los desarrolladores de software a crear páginas Web dinámicas basadas en HTML y otros formatos.

En la parte cliente se usará lenguaje HTML en su última especificación (HTML5) y las tecnologías apropiadas para crear una aplicación RIA (Aplicaciones de Internet Enriquecidas), las cuales utilizan un navegador Web estandarizado para ejecutarse. Algunas de estas tecnologías utilizadas son CSS 3 y JavaScript.

Para el almacenamiento de los datos se ha utilizado MongoDB. Se trata de un sistema de base de datos NoSQL orientado a documentos, desarrollado bajo el concepto de código abierto. MongoDB forma parte de la nueva familia de sistemas de base de datos NoSQL. En vez de guardar los datos en tablas como se hace en las bases de datos relacionales, MongoDB guarda las estructuras de datos en documentos tipo JSON con un esquema dinámico y representadas en binario (MongoDB llama a ese formato BSON), haciendo que la integración de los datos en ciertas aplicaciones sea más fácil y rápida.

MongoDB cuenta con técnicas de optimización de consultas, entre estas técnicas se encuentra la indexación. MongoDB cuenta con diversos tipos de índices, a fin de proveer índices adaptados a los diferentes modelos de datos que pueden existir. En este trabajo no solo se establecerá un índice o índices para mejorar el rendimiento, sino que además se realizará un estudio detallado de los distintos tipos de estrategias de indexación.

3.2 FUENTE DE LOS DATOS

El origen de los datos geoespaciales utilizados para este proyecto proviene del satélite *QuikSCAT* (contracción de Quick Scatterometer). Se trata de un satélite que se ha encargado durante más de una década en recolectar datos meteorológicos marinos. El proyecto se llevó a cabo con el trabajo colaborativo de la NASA y la NOAA (Administración Nacional Oceánica y Atmosférica, del inglés *National Oceanic & Atmospheric Administration)* en una misión conjunta.

Los objetivos de la misión se pueden generalizar en dos vertientes: científicos y operativos. Entre los objetivos científicos se encuentran:

- Combinar la información recogida sobre los vientos con mediciones de instrumentos científicos de otras disciplinas que ayuden a entender mejor los mecanismos del cambio climático.
- Estudio de los cambios anuales y semestrales de la lluvia.
- Estudio de los cambios diarios y estacionales del hielo del mar Ártico y el movimiento de los glaciares.
- Determinar la fuerza atmosférica, la respuesta de los océanos y los mecanismos de interacción aire-mar en varias escalas espaciales y temporales.

Entre los objetivos operativos, los datos obtenidos del satélite QuikSCAT, se encuentran:

- Ayudar a mejorar el pronóstico del tiempo cerca de las costas mediante el uso de los datos del viento en la intemperie.
- Obtener información para realizar modelos de predicción de olas.
- Ayudar a mejorar el pronóstico de tormentas y la vigilancia de su evolución.

El satélite QuikSCAT fue precedido por el satélite NSCAT (NASA *Scatterometer*). La misión del NSCAT, cuyo lanzamiento fue en 1996 tenía previsiones de tener una duración de varios años, pero un fallo inesperado en 1997 obligó a cancelar la misión. Debido al éxito del NSCAT, la NASA empezó a construir un nuevo satélite para sustituirlo rápidamente y tener en el menor espacio de tiempo posible los datos del nuevo satélite *QuikSCAT*.

Ilustración 2: Imagen artificial del satélite QuikSCAT

El corazón de QuikSCAT es un instrumento llamado *scatterometer*. Se trata de un radar especialmente diseñado que opera a una frecuencia de microondas que penetra las nubes. Esto, unido a la órbita polar del satélite, hace que los sistemas de vientos sobre los océanos sean visibles para QuikSCAT en el mundo entero. Las mediciones dan una información detallada acerca de los vientos de los océanos, olas, corrientes, características polares de hielo y otros fenómenos, en beneficio de meteorólogos, climatólogos, oceanógrafos y marinos.

Los haces del radar no son parados por las nubes; sin embargo, la observación queda modificada por todo fenómeno que altere las ondas: la lluvia, los vientos muy débiles o los vientos fuertes.

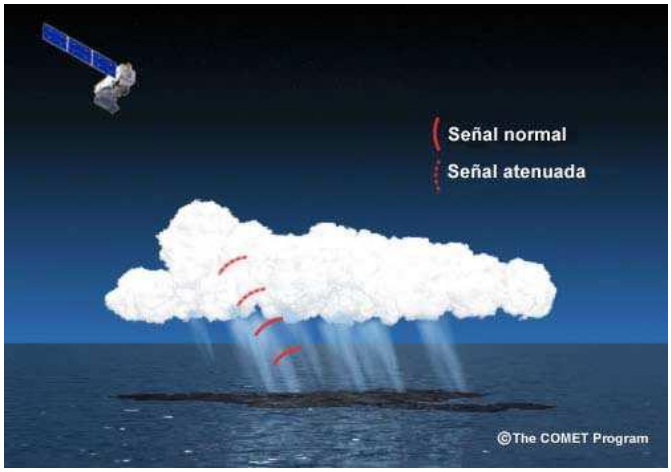


Ilustración 3: Señales del satélite QuikSCAT

Técnicamente el instrumento usado en el QuikSCAT es un radar de alta frecuencia de microondas (de 13,4 GHz) diseñado específicamente para medir la velocidad y la dirección del viento próximas a la superficie del océano. El radar está montado en un plato giratorio de 1 metro de diámetro que produce dos haces de cobertura, barriendo en forma circular. Este instrumento explora una andana de 1.800 kilómetros de ancho y realiza aproximadamente 400.000 medidas, cubriendo el 90% de la superficie de la tierra en un día, y proporcionándonos dos tipos de resoluciones: cada 25 km la más gruesa, y cada 12,5 km la más fina.

El satélite QuikSCAT fue lanzado el 19 de junio del 1999 desde la base de la fuerza aérea Vandenberg, EE.UU., montado en un cohete *Titan II*. Dieciocho días después del lanzamiento, la instrumentación del satélite empezó a recoger datos. La duración de la misión estaba estimada en 2 o 3 años pero la misión se alargó hasta 2009 (10 años). El 23 de noviembre del 2009 una de las antenas del satélite, cuya función era recoger las señales en tiempo real, dejó de girar debido al desgaste de los rodamientos. Las piezas tenían una vida esperada de alrededor de 5 años. Esta avería provocó el final de la misión.

El satélite realizaba 14 orbitas diarias alrededor del planeta. Cada orbita queda registrada en un fichero. Los datos están organizados por un tipo de ficheros llamados HDF (Hierarchical Data Format). La principal ventaja de estos ficheros es que son capaces de almacenar una gran cantidad de datos numéricos referidos a multitud de variables. La versión de los ficheros que utiliza es HDF4, la cual puede soportar diferentes modelos de datos incluyendo matrices multidimensionales, imágenes de tipo *raster* y tablas. El formato HDF es auto descriptivo, lo que permite que una aplicación pueda interpretar los datos sin ningún tipo de información externa.

3.3 MONGODB

3.3.1 VISIÓN GENERAL DE MONGODB

Para la realización del proyecto se ha optado por usar una base de datos NoSQL de código abierto, MongoDB. MongoDB es un sistema de base de datos NoSQL de tipo orientado a documentos, desarrollado bajo el concepto de código abierto. Forma parte de la nueva familia de sistemas de base de datos denominados “NoSQL”.

Ilustración 4: Logotipo de la base de datos NoSQL MongoDB

NoSQL (a veces llamado "No sólo SQL") es una amplia clase de sistemas de gestión de bases de datos que difieren del modelo clásico de gestión de bases de datos relacionales (RDBMS) en aspectos relevantes. El más destacado es que no siempre usan SQL como principal lenguaje de consultas, asimismo, en algunos casos si pueden soportar lenguajes de consultas de tipo SQL, de ahí el nombre común de "no sólo SQL" para subrayar dicho hecho.

Sin embargo, la opcionalidad de no tener que utilizar un lenguaje de consultas como SQL, no es su única característica importante; entre ellas se encuentran las siguientes: permite que los datos almacenados no requieran estructuras fijas como tablas; normalmente no soportan operaciones JOIN; no garantizan completamente las propiedades ACID (atomicidad, consistencia, aislamiento y durabilidad); y de manera general escalan bien horizontalmente.

Dentro de las bases de datos NoSQL, existen diferentes tipos en función del modelo de datos que representan: orientadas a documentos, orientadas a columnas, de clave valor, grafos...

MongoDB se trata de una base de datos orientada a documentos o de tipo documental, que en lugar de registrar relaciones de datos, almacena documentos en estructuras dinámicas. Sus principales características definitorias son, por un lado, la enorme flexibilidad que brinda en los esquemas de datos a la hora de almacenar la información

y su simplicidad de uso, pero también su altísimo rendimiento y escalabilidad, la facilidad de mantenimiento y la capacidad de funcionar sin puntos de fallo, pues, pueden recuperarse aunque se caiga cualquiera de sus nodos.

El tipo de modelo de datos que implemente una base de datos como NoSQL es importante, pero también lo son otros factores. Las bases de datos NoSQL están pensadas para ser escalables y distribuidas. En cambio, el teorema CAP o teorema Brewer, dice que en sistemas distribuidos es imposible garantizar a la vez: consistencia, disponibilidad y tolerancia a particiones (*Consistency-Availability-Partition Tolerance*). Cada carácter:

- Consistencia: al realizar una consulta o inserción siempre se tiene que recibir la misma información, con independencia del nodo o servidor que procese la petición.
- Disponibilidad: que todos los clientes puedan leer y escribir, aunque se haya caído uno de los nodos.
- Tolerancia a particiones (a veces traducido como tolerancia a fallos): los sistemas distribuidos, incluido MongoDB, pueden estar divididos en particiones (generalmente de forma geográfica). Esta condición implica, que el sistema tiene que seguir funcionando aunque existan fallos o caídas parciales que dividan el sistema.

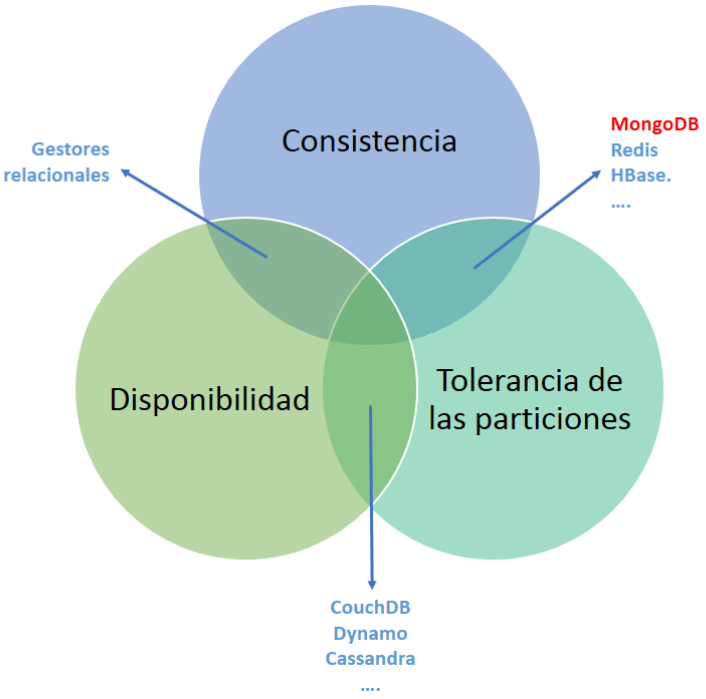


Ilustración 5: Esquema teorema CAP

MongoDB garantiza la consistencia y la tolerancia a particiones. No obstante, para lograr la consistencia en todo momento debe replicar los datos a través de los nodos, sacrificando la disponibilidad. En MongoDB, para logar la consistencia, se establece un servidor principal en entornos distribuidos, que es el único que acepta inserciones o modificaciones.

Hay que tener en cuenta que la clasificación de una base de datos NoSQL no es siempre definitiva, ya que algunos de estos sistemas NoSQL pueden configurarse para cambiar su comportamiento, aun perdiendo en parte la esencia de los mismos. En la base de datos que nos atañe, MongoDB, se puede configurar el nivel de consistencia, eligiendo el número de nodos en los que se replicarán los datos. También podemos configurar si se pueden leer datos de los nodos secundarios. Si permitimos leer de un nodo secundario sacrificamos consistencia, pero ganamos disponibilidad.

De manera general, se enumeran algunas de las características más relevantes de MongoDB:

- De propósito general, casi tan rápida como las bases de datos NoSQL de tipo clave-valor, y con casi todas las funcionalidades de las bases de datos relacionales.

- Alta disponibilidad.
- Ofrece una alta escalabilidad, desde un servidor aislado a arquitecturas distribuidas de grandes clústers.
- Permite particionar, de manera transparente para el usuario, nuestra base de datos en tantas como shards tengamos. Esto aumenta el rendimiento del procesado de los datos al ser cada una de ellas más pequeña que la original.
- Es capaz de actualizarse sin dejar de dar servicio.
- No tiene los cuellos de botella que se producen en las bases de datos relacionales (RDBMS) al procesar grandes cantidades de información.
- Utiliza objetos JSON para guardar y transmitir la información. JSON es el estándar Web hoy en día, por lo que supone una gran ventaja que Web y base de datos hablen el mismo lenguaje.
- Permite utilizar *Map-Reduce* para el procesado de la información a través de funciones JavaScript que se ejecutan en los servidores.
- Puede almacenar y ejecutar funciones JavaScript en el servidor.
- Ofrece para almacenamientos en la nube una potente herramienta Web que nos permite monitorizar todo lo que pasa en nuestras bases de datos y en nuestras máquinas, MongoDB Management *Service* (MMS).
- Replicación nativa: sincronización de datos entre servidores.
- Seguridad: autenticación, autorización, etc.
- Gestión avanzada de usuarios.
- *Automatic -fail-over*: elección automática de un nuevo primario cuando este se ha caído.
- Aggregation Framework, Procesamiento *batch* de datos para cálculos agrupados usando operaciones nativas de MongoDB.
- Auto balanceado de carga a través de los distintos shards. El balanceador decide cuándo migrar los datos, y a qué shard, para que estén uniformemente distribuidos entre todos los servidores del clúster. Cada shard aloja los datos correspondientes a un rango de la clave escogida para particionar nuestra colección.

3.3.2 MODELO DE DATOS DE MONGODB

MongoDB se trata de una base de datos de tipo NoSQL. Como ya hemos visto, este término cobija varios productos, pero todos comparten ciertos conceptos relacionados con el almacenamiento, la gestión de datos y el tratamiento de datos voluminosos.

El modelo que ofrece MongoDB para los datos es el denominado "orientado a documentos", se trata de almacenar los datos en una estructura de "documentos" que MongoDB define como "documentos" o también denominado "*things*" (cosas) que básicamente son elementos de tipo JSON con un esquema dinámico representados en forma binaria, al que MongoDB denomina formato BSON.

Este modelo de datos implica que no existe un esquema predefinido para los datos, ni sea necesario declararlo a priori, con las ventajas y desventajas que ello conlleva.

Los elementos de los datos se denominan documentos y se guardan en colecciones. Una colección puede tener un número indeterminado de documentos. Comparando con una base de datos relacional, se puede decir que las colecciones son como tablas y los documentos son registros en la tabla. La diferencia es que en una base de datos relacional cada registro en una tabla tiene la misma cantidad de campos, mientras que en MongoDB cada documento en una colección puede tener diferentes campos.

En un documento, se pueden agregar, eliminar, modificar o renombrar nuevos campos en cualquier momento, ya que no existe un esquema predefinido. La estructura de un documento es simple y compuesta por "*key-value pairs*" (pares de clave-valor) parecido a las matrices asociativas en un lenguaje de programación, esto es debido a que MongoDB sigue el formato de JSON. En MongoDB la clave es el nombre del campo y el valor es su contenido, los cuales se separan mediante el uso de ":". Como valor se pueden usar números, fechas, cadenas de textos o datos binarios como imágenes o cualquier otro "*key-value pairs*".

3.3.3 ADAPTACION DE LOS DATOS GEOESPACIALES

La fuente de datos de la aplicación proviene del satélite QuikSCAT, el cual generó mediciones espaciales sobre la Tierra, en concreto sobre zonas marítimas. Estos datos geoespaciales, como ya hemos visto, recogen información específica y ordenada, de la cual para simplificar el proyecto, se ha seleccionado la velocidad y dirección del viento y el grado de precipitaciones. Estas mediciones se realizan en un punto localizado cerca de la superficie de océanos y mares en un momento concreto y se almacenan.

Los datos geoespaciales, tras realizar su descompresión y posterior descompactación, a través de su interpretación como ficheros HDF4 a ficheros de formato CSV, presentan el siguiente modelo de datos:

3	2009001005418,-66.40,119.72, 4.51, 0.00, 0.00
4	2009001005418,-66.29,120.15, 4.35, 0.00, 0.00
5	2009001005421,-66.35,118.87, 13.70, 0.00, 0.00
6	2009001005421,-66.26,119.37, 11.22, 0.00, 0.00
7	2009001005421,-66.11,119.77, 12.25, 0.00, 0.00
8	2009001005421,-65.95,120.10, 12.41, 0.00, 0.00
9	2009001005425,-66.37,118.10, 32.92, 0.00, 0.00
10	2009001005425,-66.23,118.46, 23.97, 0.00, 0.00
11	2009001005425,-66.10,118.96, 21.71, 0.00, 0.00
12	2009001005425,-65.94,119.37, 18.37, 0.00, 0.00
13	2009001005425,-65.79,119.81, 17.83, 0.00, 0.00
14	2009001005425,-65.63,120.16, 17.43, 0.00, 0.00
15	2009001005425,-65.45,120.50, 11.91, 0.00, 0.00
16	2009001005429,-66.19,117.82, 38.29, 0.00, 0.00
17	2009001005429,-66.04,118.18, 26.58, 0.00, 0.00
18	2009001005429,-65.92,118.59, 24.30, 0.00, 0.00

Ilustración 6: Extracto de datos geoespaciales en formato CSV

El formato CSV (*comma-separated values*) se trata de un tipo de documento de formato abierto y sencillo para representar datos en forma de tabla, en las que las columnas se separan por comas y las filas por saltos de línea. Para el modelo de datos geoespacial utilizado, cada fila representa una medición sobre un punto terrestre y en cada fila, existen elementos separados mediante comas cuya estructura ordenada es la siguiente:

Tiempo, latitud, longitud, velocidad del viento, dirección del viento, grado de precipitaciones.

El tiempo se representa en formato juliano (yyyydddHHmmss), donde los cuatro primeros dígitos forman el año, los tres siguientes el día a partir del comienzo del año, los dos siguientes la hora, los dos siguientes el minuto y los dos siguientes y últimos los segundos. El resto de elementos se tratan de elementos numéricos simples.

Representados los datos de forma textual presentarían el siguiente formato:

- Para la fecha día 1 de enero a las 0 horas 54 minutos y 21 segundos en el punto con latitud -66.35 y longitud 118.87 la dirección del viento era 13.70 m/s, la dirección era de 0 grados y la lluvia era 0 mm/h
- Para la fecha día 1 de enero a las 0 horas 54 minutos y 21 segundos en el punto con latitud -66.26 y longitud 119.37 la dirección del viento era 11.22 m/s, la dirección era de 0 grados y la lluvia era 0 mm/h
- Para la fecha día 1 de enero a las 0 horas 54 minutos y 21 segundos en el punto con latitud -66.11 y longitud 119.77 la dirección del viento era 12.25 m/s, la dirección era de 0 grados y la lluvia era 0 mm/h
- Para la fecha día 1 de enero a las 0 horas 54 minutos y 21 segundos en el punto con latitud -66.95 y longitud 120.10 la dirección del viento era 12.41 m/s, la dirección era de 0 grados y la lluvia era 0 mm/h

Para almacenar en MongoDB los datos geoespaciales descritos, se ha utilizado el siguiente modelo como estructura:

```
{
  "_id" : ObjectId("558fc2c239fce70c1486c30a"),
  "_class" : "com.geospatial.model.SpatialPoint",
  "date" : ISODate("2005-01-13T09:08:40Z"),
  "location" : [112.739990234375,  -4.4099998474121094],
  "windSpeed" : 5.679999828338623,
  "windDir" : 0.0,
  "rainRate" : 0.0,
  "longitude" : 112.739990234375,
  "latitude" : -4.4099998474121094
}
```

El modelo utilizado, utiliza las siguientes claves para almacenar los valores:

- -id: se trata de un identificar único y necesario para todos los elementos almacenados en MongoDB. Se crea automáticamente.
- _class: se trata de una clave generada automáticamente por Spring Framework. Hace referencia al objeto de la clase Java generado y transmitido a la base de datos para almacenarse.
- Date: clave para almacenar la fecha de generación de la medición en formato ISODate, un formato para fechas ampliamente utilizados.
- Location: se trata de una clave que almacena un *array*, que alberga; como primer elemento la longitud; y como segundo elemento la latitud.
- WindSpeed: clave para almacenar la medición correspondiente a la velocidad del viento.
- WindDir: clave para almacenar la medición correspondiente a la dirección del viento.
- RainRate: clave para almacenar la medición correspondiente al grado de precipitaciones.

Para la elaboración de diferentes pruebas sobre técnicas de indexación en MongoDB se declaran de forma redundante las siguientes claves:

- Longitude: clave para almacenar la longitud donde se ha realizado la medición.
- Latitude: clave para almacenar la latitud donde se ha realizado la medición.

3.3.4 OPERACIONES BÁSICAS REALIZADAS EN MONGODB

MongoDB, al igual que una base de datos tradicional, establece las operaciones básicas CRUD para la inserción, modificación o borrado de los datos. A continuación se establecen las operaciones básicas realizadas en MongoDB.

MongoDB no utiliza SQL como lenguaje de consultas. Por ello la sintaxis de las operaciones difieren notablemente de una base de datos relacional tradicional. Más adelante, se acompaña una tabla donde se establecen las sintaxis de las operaciones CRUD en MongoDB frente a una base de datos SQL tradicional:

	MONGODB	SQL
INSERCIÓN	Insert	Insert
CONSULTA	Find	Select
ACTUALIZACION	Update	Update
BORRADO	Remove / Drop	Delete

Tabla 1: Operaciones básicas en MongoDB respecto a SQL tradicional

Las operaciones se tratan de funciones JavaScript, al realizar las operaciones se tiene que llamar al objeto base de datos, "db"; y seguidamente, precedido de un punto, el nombre de la colección sobre la cual queremos realizar la operación pertinente, y, nuevamente precedido de un punto, el nombre de la operación a realizar. A continuación, se detallan las distintas operaciones CRUD correspondientes a los datos geoespaciales utilizados en el proyecto:

INSERCIÓN DE ELEMENTOS - INSERT()

Para la inserción de elementos, en este caso datos geoespaciales, es necesario crear primeramente una colección donde albergarlos, lo que se asemejaría al concepto de tabla en SQL.

MongoDB, al realizar la inserción de un documento (se asemejaría a la entrada de una tabla en SQL), obliga a declarar la colección en la que se insertará. En caso de que no exista la colección que indicamos, se creará automáticamente. Por tanto, para la creación de una colección, basta con nombrar el nombre de la colección sobre la que queremos almacenar el documento.

Para la inserción de documentos, en este caso datos geoespaciales, deben de declararse los elementos en formato JSON para posteriormente ser almacenados.

Para almacenar los elementos en MongoDB existe la función "insert", pudiendo especificarse también como "save". Esta función se asemeja al "INSERT INTO" de SQL e inserta el elemento declarado en la base de datos.

Asimismo, se acompaña un ejemplo correspondiente a los datos geoespaciales de este proyecto:

```
db.datosgeoespaciales.insert({
        Date: '2013-01-13T12:53:09.186Z',
        Longitude: 10,
        Latitude: 10,
        windSpeed: 4.6,
        windDir: 10,
        rainRate: 5
});
```

Los documentos, se pueden definir como objetos JSON o también como objetos JavaScript. Gracias a esta última definición, es posible declarar el documento como un objeto JavaScript, almacenarlo en una variable y posteriormente ser insertado:

```
Var spatialPoint = {
        Date: '2013-01-13T12:53:09.186Z',
        Longitude: 55,
        Latitude: 10,
        Longitude: -5,
        windSpeed: 0,
        WindDir: 10,
        RainRate: 5
};

db.datosgeoespaciales.insert(spatialPoint);
```

MongoDB permite la inserción de documentos con estructuras dinámicas. Como ya se ha explicado, se trata de una de las características principales de la base de datos, esto le permite a MongoDB mantener colecciones con diversidad de documentos. A continuación, vamos a realizar la inserción de un nuevo dato geoespacial, pero con la diferencia de que este nuevo elemento no seguirá la estructura de los elementos previamente almacenados y contará con un campo redundante llamado "location". De esta manera, en MongoDB la inserción de elementos es completamente posible y esto conforma una de sus ventajas.

```
db.datosgeoespaciales.insert({
        Date: '2013-01-13T12:53:09.186Z',
        Location: [{
```

```
                    Latitude: 10,
                    Longitude: 10
                }],
        windSpeed: 0,
        windDir: 10,
        rainRate: 5
});
```

Para cualquier tipo de inserción, en caso de no declarar previamente un valor para dicha clave, MongoDB genera un identificador único, "_id" de tipo *Object* (objeto). Tomando como ejemplo una inserción realizada previamente y realizando su consulta retorna:

```
db.datosgeoespaciales.insert({
        "_id" : ObjectId("558fc2c239fce70c1486c30a"),
         "Date": '2013-01-13T12:53:09.186Z',
         "Location ": [{
                    "Latitude" : 10,
                    "Longitude" : 10
                  }],
         "windSpeed ": 0,
         "windDir ": 10,
         "rainRate" : 5
});
```

El objeto identificador consiste en un número hexadecimal de 12 bytes formado a partir de la combinación del momento *timestamp* con una precisión de un segundo, el id de la máquina donde se está ejecutando MongoDB, el id del proceso y, finalmente, tres bytes autoincrementales.

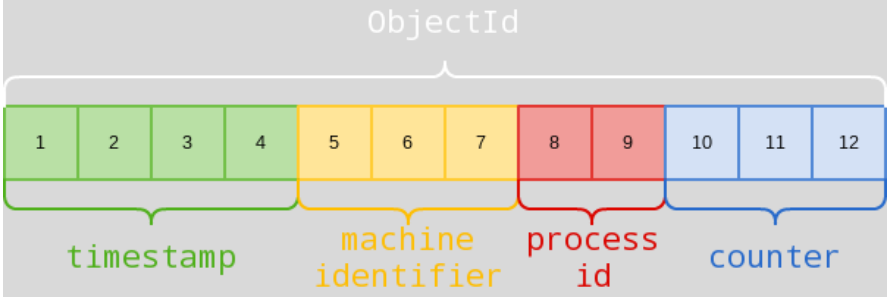


Ilustración 7: Composición del valor ID de un documento en MongoDB

El resultado es un identificador único para cada elemento que, además de servir para identificar de forma exclusiva un elemento, podemos utilizar para listar los registros ordenados cronológicamente.

Si se desea se puede especificar manualmente el valor del campo _id cuando se inserta el elemento, en cambio, es primordial asegurar que este valor sea único, de lo contrario el registro con dicho campo duplicado no será insertado, proporcionado un error por clave primaria duplicada.

BÚSQUEDA DE REGISTROS - FIND()

MongoDB permite la búsqueda de elementos por uno o más elementos y de acuerdo a valores únicos o por rango.

La sintaxis para las búsquedas difieren enormemente de la bases de datos tradicionales de tipo SQL. En MongoDB para realizar búsquedas se utiliza la sentencia "find()". Incluyendo dentro de la sentencia *find* la sentencia de las diferentes consultas que queremos realizar.

En primer lugar, supondremos que existe una colección previamente creada y con documentos insertados llamada "datosgeoespaciales". Para obtener todos los documentos bastará con declarar la sentencia find() sin más:

```
db.datosgeoespaciales.find();
```

Una búsqueda como la anterior sería similar en SQL a "SELECT * FROM geoespaciales". De dicha operación se obtendrán todos los documentos existentes en la colección:

```
{ "_id" : ObjectId("5232344a2ad290346881464a"), "Date" : "'2013-01-13T12:53:09.186Z", "Longitude" : "10", "Latitude": "10", "WindDir": "10", "RainRate": "10"}

{ "_id" : ObjectId("5232344a2ad290346881464b"), "Date" : "'2013-01-13T12:51:09.186Z", "Longitude" : "10", "Latitude": "10.1", "WindDir": "10.1", "RainRate": "10"}
```

{ "_id" : ObjectId("5232344a2ad290346881464c"), "Date" : "'2013-01-13T12:50:09.186Z", "Longitude" : "10", “Latitude”: “10.2”, “WindDir”: “10.2”, “RainRate”: “10”}

…

…

Para la obtención de documentos específicos, es necesario realizar una búsqueda con filtros por uno o más parámetros. Para ello sólo debemos insertar el filtro deseado a la función find().

En este caso vamos a realizar una consulta por rango temporal y espacial, se trata de una consulta utilizada para llevar a cabo consultas por rango en la aplicación Web desarrollada. Para ello declaramos la sentencia *find* e insertamos el filtro, dicho filtro permite la declaración de distintos operadores. En la siguiente tabla se indican los operadores y su función:

OPERADOR	FUNCION
$eq	Valores coincidentes que son igual que el valor especificado
$gt	Valores coincidentes que son mayores que el valor especificado
$gte	Valores coincidentes que son mayores o igual que el valor especificado
$lt	Valores coincidentes que son menores que el valor especificado
$lte	Valores coincidentes que son menores o iguales que el valor especificado
$ne	Valores coincidentes que no son iguales al valor especificado
$in	Valores coincidentes que son especificados en un array
$nin	Valores coincidentes que no son especificados en un array

Tabla 2: Operadores de consultas en MongoDB

A continuación se indica la sentencia utilizada para realizar la consulta por rango espacial y temporal. En ella se utilizan las claves longitud, latitud y fecha y los operadores gte (mayor o igual que) y lte (menor o igual que) para establecer el rango de búsqueda.

```
db.datosgeoespaciales.find(
longitude : { $gte : "0", $lte : "10" },
latitude : { $gte : "0", $lte : "10" },
date: {$gte: {$date : "'2013-01-13T12:53:09.186Z
"}, $lte: {$date: "'2013-02-13T12:53:09.186Z"}}
);
```

La sentencia anterior sería similar en SQL a "SELECT * FROM datosgeoespaciales WHERE longitude BETWEEN 0 AND 10, LATITUDE BETWEEN 0 AND 10 AND DATE BETWEEN '2013-01-13T12:53:09.186Z' AND '2013-02-13T12:53:09.186Z'".

Si quisiéramos limitar los resultados a un número máximo especificado de documentos, de forma similar en SQL a la sentencia "SELECT * FROM datosgeoespaciales LIMIT 30". Es necesario agregar .limit(#) al final del comando .find():

```
db.datosgeoespaciales.find().limit(30);
```

La misma modalidad se utiliza para la operación de ordenación de los documentos por un campo en particular. Para ello es necesario indicar el operador ".sort()" como argumento a la consulta. Luego se declaró una consulta para obtener todos los datos geoespaciales ordenados por fecha de medición de forma inversa (aquellos elementos con fecha más antigua primero):

```
db.datosgeoespaciales.find().sort({date: -1});
```

El número 1 que acompaña al argumento de ordenamiento es el tipo de ordenación; 1 para descendiente y -1 para ascendente. En SQL sería similar a "SELECT * FROM autores ORDER BY apellido DESC/ASC".

ELIMINACIÓN DE REGISTROS - REMOVE() Y DROP()

Para la eliminación de colecciones y documentos en MongoDB se establecen dos sintaxis *.drop* y *.remove*. Esta última sentencia se utiliza para borrar contenidos y "*drop*", por su parte, para eliminar colecciones y sus documentos asociados.

Para eliminar los documentos de una colección que cumplan alguna condición se utiliza la sentencia *remove* especificando el o los filtros deseados. En dichos filtros se utilizan los operadores de consultas vistos anteriormente. Aquí se muestra un ejemplo:

```
db.datosgeoespaciales.remove(
        longitude : { $gte : "0", $lte : "10" }
)
```

Dicha sentencia correspondería a la sentencia SQL de una base de datos tradicional "DELETE FROM datosgeoespaciales WHERE longitude BETWEEN 0 AND 10".

Para eliminar todos los documentos de una colección se utiliza la sentencia remove, sin ningún tipo de condición.

```
db.datosgeoespaciales.remove();
```

En SQL sería similar a "DELETE FROM datosgeoespaciales".

Por último, para la eliminar la colección completa, incluyendo los documentos existentes, se utiliza la sentencia "*drop*".

```
db.datosgeoespaciales.drop();
```

En SQL sería similar a "DROP TABLE autores".

3.3.5 TÉCNICAS DE INDEXACIÓN SOBRE EL MODELO DE DATOS

Debido al elevado volumen de datos de los que tiene que hacer uso la aplicación Web desarrollada y el carácter del modelo de datos (datos geoespaciales con valores dimensionales sobre los que realizar la búsqueda: tiempo y zona (longitud y latitud)), se vuelve fundamental utilizar estrategias de indexación que disminuyan los tiempos y complejidades en las búsquedas.

MongoDB proporciona diferentes tipos de índices para adaptarse a distintos modelos de datos. El objetivo que se pretende con todo índice es la creación de una estructura de datos de tipo árbol B, que, dado el carácter de la estructura, ofrezca una complejidad logarítmica en las búsquedas.

MongoDB ofrece los siguientes tipos de índices:

- Índice simple: se trata de un índice que se define para un único campo de una colección. Genera una estructura de tipo árbol B+.
- Índice compuesto: se trata de un índice que se define para múltiples campos de una colección. Genera una estructura de tipo árbol B+ compuesta por los dos o más índices definidos.
- Índice geoespacial: para soportar de forma eficiente las consultas a base de datos en las que intervengan datos geolocalizados por coordenadas geográficas. Genera una estructura de tipo R-tree. Sobre este índice se declaran dos subtipos:
    - 2d, Se utilizan para indexar puntos definidos en un plano de 2 dimensiones. Los Índices 2d soportan cálculos en un Plano Euclidiano (superficie plana).
    - Los índices 2d esféricos son un tipo de Índices que soportan consultas de cálculo de geometrías en una esfera similar a la de La Tierra. Los índices 2d esféricos solo están disponibles a partir de la versión 2.4 de MongoDB.

En la aplicación Web desarrollada para este proyecto, todas las búsquedas en la base de datos de MongoDB se realizan por rango temporal y espacial. La manera ideal, por tanto, sería indexar el espacio y el tiempo, es decir, los atributos correspondientes al tiempo, la latitud y la longitud. Estos tres atributos conforman un espacio tridimensional, ya que permiten localizar cualquier elemento existente en el espacio con los valores de esos tres elementos. Para indexar estos elementos la opción eficaz sería la utilización de un tipo de índice 3D o de tres dimensiones, de manera que se pudiera indexar de forma conjunta el tiempo, la longitud y la latitud mediante la construcción de una estructura de Árbol R de tres valores.

A pesar de la gran ventaja que supondría para este proyecto, MongoDB (actualmente versión 3.0) no ofrece un índice de tres dimensiones. Por consiguiente, es necesario adaptar otras técnicas de indexación que permitan agilizar las consultas.

A priori, se puede plantear que lo más obvio sería la construcción de un índice 2D o de dos dimensiones sobre el espacio (longitud y latitud) y pronto nos daríamos cuenta de que podría mejorar si incluyésemos además un índice simple sobre el tiempo. Sin embargo, ¿se podría afirmar que se trata de la mejor opción? Indudablemente no, pues fruto de la elaboración de este proyecto podemos comprobar que existen otras técnicas iguales o mejores. Ahora se recogen las técnicas de indexación para datos geoespaciales estudiadas en este proyecto.

ÍNDICE TEMPORAL SIMPLE + ÍNDICE LONGITUD SIMPLE + INDICE LATITUD SIMPLE

Como la gran mayoría de bases de datos, MongoDB permite la indexación simple por atributos. Como ya hemos aprendido, como ya se ha explicado previamente, MongoDB genera una estructura de árbol B sobre el atributo que queremos indexar, a fin de realizar búsquedas con complejidad logarítmica.

Para esta estrategia vamos a crear un índice simple sobre el atributo del tiempo date y otros dos índices simples sobre los atributos de localización: longitud y latitud.

```
db.geoespatial.ensureIndex({ “date” : 1 },{ “name” : “dateIndex” });
db.geoespatial.ensureIndex({ “longitude” : 1 },{ “name” : “longitudeIndex” });
db.geoespatial.ensureIndex({ “latitude” : 1 },{ “name” : “latitudeIndex” });
```

ÍNDICE GEOESPACIAL (LONGITUD, LATITUD) + ÍNDICE TEMPORAL SIMPLE

El índice geoespacial es una de las herramientas más potentes de MongoDB. Es un índice sobre coordenadas geográficas, que además de aumentar la eficiencia sobre los datos, incluye funciones de consultas especiales para datos geográficos.

Para esta técnica de indexación se declara un índice geoespacial sobre el espacio (longitud y latitud) y además se declara un índice simple para el tiempo.

```
db.geoespatial.ensureIndex({ “date” : 1 },{ “name” : “dateIndex” });
db.geoespatial.ensureIndex({ “location” : “2d” },{ “name” : “GeoespacialIndex” });
```

De esta manera tendremos la ventaja de contar con un índice geoespacial y además contar con un índice sobre el tiempo.

ÍNDICE COMPUESTO (TIEMPO, LONGITUD, LATITUD)

El índice compuesto es de uno de los índices más completos y de más complejidad de MongoDB. Se trata de un índice sobre múltiples valores. Su uso es muy variado y su eficiencia dependerá de la estrategia y orden en la declaración del propio índice.

El índice genera un árbol B único sobre los atributos declarados para mejorar la eficiencia en las búsquedas. En función del orden en la declaración del índice puede variar significativamente su rendimiento, ya que al generar un único árbol B a partir de diversos atributos, en los primeros nodos del árbol se situarán los elementos correspondientes al orden del primer atributo declarado y así sucesivamente con el resto de elementos.

Para las consultas realizadas en este proyecto, al tratarse de consultas por rango para todos los atributos del espacio y el tiempo, pondremos primero aquellos atributos que restrinjan más. Por consiguiente, primero se restringe por fecha, disminuyendo significativamente el número de elementos, del mismo modo posteriormente se restringe para la longitud y por último para la latitud.

```
db.collection.ensureIndex({“date”: 1, “longitude”: 1, “latitude”})
```

ÍNDICE GEOESPACIAL (LONGITUD, LATITUD) + ÍNDICE COMPUESTO (TIEMPO, LONGITUD, LATITUD)

Se trata de la técnica de indexación más compleja. Por un lado se declara un índice geoespacial para las coordenadas geográficas (espacio) y por otro lado se declara un índice compuesto. En el índice compuesto selecciona como primer elemento para su declaración el tiempo y en segundo lugar el espacio

```
db.geoespatial.ensureIndex({ “location” : “2d” },{ “name” : “GeoespacialIndex” });
db.collection.ensureIndex({“date”: 1, “location”: 1})
```

El objetivo de esta técnica de indexación es aprovechar las ventajas del índice geoespacial y que a su vez este sirva de forma auxiliar al índice compuesto. De manera que al realizar una consulta, en primer lugar haga uso del índice compuesto y posteriormente, de forma auxiliar, actúe el índice geoespacial para mejorar la consulta para la parte espacial.

LÍMITES DE LOS ÍNDICES

Como se ha descrito MongoDB ofrece múltiples herramientas para la indexación. Sin embargo, la declaración de índices también conlleva alguna serie de desventajas:

- Solo se permiten un máximo de 40 índices por colección. Es un número muy grande, pero está bien indicarlo.
- No se permiten índices lógicamente equivalentes (por ejemplo, {x:1} y {x:-1}). Esto quiere decir que no podemos generar dos índices sobre el mismo valor organizados en distinto orden (ascendente, descendente).
- Lo índices mejoran las consultas pero crean inserciones lentas. En MongoDB las inserciones no son precisamente su mejor característica, si se declaran multitud de índices se pueden ralentizar en gran medida las inserciones.
- Los índices más específicos, como puede tratarse de un índice compuesto sobre los elementos ({a:1, b:1, c:1}) pueden ser más útiles que los menos específicos ({a:1} , {b:1}). Sin embargo, la ordenación de los más específicos puede no ser tan rápida como en los menos específicos. Esto se traduce en pérdida de rendimiento.

3.4 ALCANCE Y OBJETIVOS

El presente trabajo tiene como alcance desarrollar una aplicación Web que sirva como marco para el estudio de los datos geoespaciales generados por el satélite QuikSCAT y que a su vez permita realizar búsquedas y consultas sobre los datos.

La aplicación Web debe proporcionar un mecanismo de visualización de los datos realizados por un rango temporal y espacial de búsqueda y posteriormente llevar a cabo

una clasificación. Para la clasificación se establecerán una serie de colores a mostrar sobre un mapa gráfico de acuerdo a los valores de la dirección del viento.

Además se plantea como alcance del proyecto, entender la base de datos de MongoDB la cual deberá tratarse del corazón de la aplicación Web, debiendo proporcionar datos de forma rápida y eficiente. A fin de mejorar las consultas en la base de datos de MongoDB, se estudiarán diferentes estrategias de indexación sobre los datos geoespaciales, realizando un estudio detallado y comparativo sobre el rendimiento de las diferentes estrategias de indexación.

Para ello, como objetivos específicos para este trabajo de fin de grado me planteo:

- Adquirir manejo en el desarrollo de aplicaciones RIA: HTML5, CSS 3 y JavaScript en la profundidad necesaria para afrontar desarrollos de portales Web de la especificación Web 2.0.
- Estudiar el patrón de diseño MVC implementado mediante Spring MVC 3.
- Adquirir competencias en Spring, para la integración de diversas tecnologías.
- Persistir un modelo de datos mediante MongoDB.
- Establecer, analizar y estudiar diversas estrategias en la indexación de datos geoespaciales en MongoDB.
- Diseñar una Web que permita a los usuarios interaccionar con la Web utilizando una serie de servicios propios, así como obtener una información real, eficaz y de gran valor.
- Producir un manual de usuario detallado sobre la instalación y funcionamiento de la Web.
- Redactar documentación o memoria final que sirva como objeto de estudio y desarrollo en posteriores Trabajos Fin de Grados.

A lo largo de este documento se describirán todos los procesos llevados a cabo para conseguir estos objetivos y alcances planteados.

4 MATERIALES: ENTORNO, HERRAMIENTAS, ESTRUCTURAS Y DATOS

4.1 PROGRAMACION ORIENTADA A OBJETOS

Como metodología de desarrollo se ha utilizado **programación orientada a objetos**. Esta metodología se basa en la declaración de "objetos" que permiten abstraer elementos de la realidad y simplificar la programación del software.

Un objeto contiene toda la información que permite definirlo e identificarlo frente a otros objetos pertenecientes a otras clases e incluso frente a objetos de una misma clase, al poder tener valores bien diferenciados en sus atributos.

A su vez, los objetos disponen de mecanismos de interacción llamados métodos, que favorecen la comunicación entre ellos. Esta comunicación favorece también el cambio de estado en los propios objetos. Esta característica lleva a tratarlos como unidades indivisibles, en las que no se separa el estado y el comportamiento.

Los **métodos** (comportamiento) y **atributos** (estado) están estrechamente relacionados por la propiedad de conjunto. Esta propiedad destaca que una clase requiere de métodos para poder tratar los atributos con los que cuenta. El programador debe pensar indistintamente en ambos conceptos, sin separar ni darle mayor importancia a alguno de ellos. Hacerlo podría producir el hábito erróneo de crear clases contenedoras de información por un lado y clases con métodos que manejen a las primeras por el otro. De esta manera se estaría realizando una programación estructurada camuflada en un lenguaje de programación orientado a objetos.

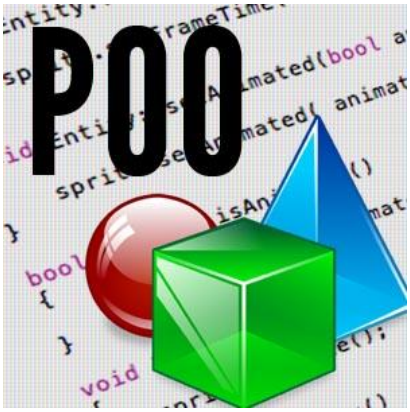


Ilustración 8: Logotipo de Programación Orientada a Objetos

4.2 HERRAMIENTA DE ANÁLISIS Y DISEÑO UML

Para la programación de este proyecto, como ya se ha descrito, se ha aplicado una Ingeniería del Software Orientada a Objetos, usando para su fase de análisis y el diseño, el lenguaje que mejor se ajusta a este entorno tecnológico, el Lenguaje Unificado de Modelado, UML en su versión 2.0.

UML es un lenguaje gráfico para visualizar, especificar, construir y documentar software. UML ofrece un estándar para describir arquitecturas y sistemas software (modelo), incluyendo aspectos conceptuales tales como procesos de negocio, funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y compuestos reciclados.

Ilustración 9: Logotipo de Unified Modeling Language (UML)

4.3 TECNOLOGÍAS BASADAS EN EL CONCEPTO WEB 2.0

Para la elaboración de este proyecto, se han utilizado tecnologías que se sitúan dentro del marco de la Web 2.0. La Web 2.0 es un esquema que marca una tendencia hacía la facilidad de interacción del usuario con la interfaz Web, el denominado front-end.

Aunque no existen unos requisitos específicos para clasificar un portal Web en el marco de la Web 2.0, si bien de forma general, deben de cumplir los siguientes requisitos:

- Dar funcionalidad y aspecto de la plataforma Web como si fuera software de escritorio.
- Respetar a los estándares del World Wide Web Consortium (W3C).

- Separar el contenido del portal Web del diseño mediante el uso de hojas de estilo en cascada (CSS).
- Utilizar HTML en su última versión operativa, actualmente HTML5.
- Utilización de JavaScript o tecnologías similares para proporcionar *front-End* "vivo" que mejore el aspecto de las interfaces del Cliente.
- Facilitar el posicionamiento con URL sencillos.

Para la construcción del *front-end* de la aplicación Web desarrollado para este proyecto, se ha utilizado HTML como estándar para la elaboración de la parte visual y como definición de la estructura y contenido. Se ha utilizado CSS, un lenguaje usado para definición y creación de estilos que mejoren el diseño. Asimismo, se ha utilizado JavaScript, el cual se trata de un lenguaje de programación interpretado y utilizado normalmente en el lado del cliente (*client-side*), para mejorar la interfaz de usuario y realizar la construcción de una aplicación Web dinámica.

Ilustración 10: Logotipo de entornos de programación utilizados: HTML, CSS y JS

4.4 APLICACIÓN WEB

Como ya se expresado en este documento, el objetivo de este proyecto ha sido la construcción de una aplicación Web accesible desde cualquier tipo de navegador y equipo, incluyendo dispositivos móviles. El objetivo es proporcionar una herramienta gráfica, sencilla y con gran poder computacional que pueda ser usada por la comunidad científica para el estudio de los datos marítimos.

En la ingeniería de software se denomina aplicación Web a aquellas herramientas que los usuarios pueden utilizar accediendo a un servidor Web a través de Internet o de una

intranet mediante un navegador. En otras palabras, es una aplicación software que se codifica en un lenguaje soportado por los navegadores Web en la que se confía la ejecución al navegador.

Las aplicaciones Web son populares debido a lo práctico del navegador Web como cliente ligero, a la independencia del sistema operativo, así como a la facilidad para actualizar y mantener aplicaciones Web sin distribuir e instalar software a miles de usuarios potenciales.

Es importante mencionar que una página Web puede contener elementos que permiten una comunicación activa entre el usuario y la información. Esto permite que el usuario acceda a los datos de modo interactivo, gracias a que la página responderá a cada una de sus acciones, como por ejemplo rellenar y enviar formularios, participar en juegos diversos y acceder a gestores de base de datos de todo tipo.

A continuación se exponen las ventajas que conlleva una aplicación Web frente a otros tipos de tecnológicas:

- Ahorra tiempo: Se pueden realizar tareas sencillas sin necesidad de descargar ni instalar ningún programa.
- No hay problemas de compatibilidad: Basta tener un navegador actualizado para poder utilizarlas.
- No ocupan espacio en el disco duro del usuario.
- Actualizaciones inmediatas: Como el software lo gestiona el desarrollador, siempre accedemos la última versión que haya lanzado.
- Consumo de recursos bajo: Dado que toda (o gran parte) de la aplicación no se encuentra en el lado del usuario, muchas de las tareas que realiza el software no consumen recursos porque se realizan desde el equipo del servidor que aloja la aplicación Web.
- Multiplataforma: Se pueden usar desde cualquier sistema operativo porque sólo es necesario tener un navegador.
- Portables: Es independiente del ordenador donde se utilice (un PC de sobremesa, un portátil...) porque se accede a través de una página Web (sólo es necesario disponer de acceso a Internet).
- La disponibilidad suele ser alta porque el servicio se ofrece desde múltiples localizaciones para asegurar la continuidad del mismo.

- Los virus no dañan los datos porque éstos están guardados en el servidor de la aplicación.
- Colaboración: Gracias a que el acceso al servicio se realiza desde una única ubicación es sencillo el acceso y compartición de datos por parte de varios usuarios. Tiene mucho sentido, por ejemplo, en aplicaciones online de calendarios u oficina.
- Los navegadores ofrecen cada vez más y mejores funcionalidades para crear Aplicaciones Web Enriquecidas (RIAs).

4.5 PLATAFORMA J2EE

Para la elaboración de este proyecto se ha utilizado la plataforma tecnológica Java2EE. Se trata de una tecnología destinada a desarrollar aplicaciones empresariales distribuidas, con una arquitectura multi-capa, escritas en lenguaje de programación Java y que se ejecutan en un servidor de aplicaciones. Es de una tecnología de código abierto de las que existe gran cantidad de herramientas y *frameworks* gratuitos, existiendo también multitud de documentaciones y APIS de desarrollo muy superiores a otras tecnologías, como puede ser NET.

Entre los motivos por lo que se ha escogido la plataforma tecnológica J2EE para la elaboración de este proyecto se encuentran:

- Permite desarrollar proyectos en lenguaje de programación Java, el cual se trata de uno de los lenguajes más utilizados en el ámbito empresarial.
- Es una tecnología muy potente.
- Cubre la mayoría de necesidades tecnológicas.
- Interoperable con otras tecnologías: XML, JavaScript, HTML, etc.
- Gran cantidad de soporte en la red: APIs, manuales, etc.
- De código abierto y con multitud de herramientas de desarrollo gratuitas (Eclipse, Spring Tool Suite, etc.).
- Muchas utilidades ya creadas y fáciles de integrar.
- Fácil conectividad con Base de Datos: driver JDBC, Spring Data...

- Y el motivo principal, la existencia de *frameworks* de gran calidad para desarrollo basados en el patrón Modelo Vista Controlador, como Spring Framework.

4.6 MODELO DE CAPAS

Para el desarrollo de aplicaciones Web modernas es indispensable realizar una abstracción en capas que permita llevar a cabo un desarrollo limpio y ordenado, y que a su vez permita realizar mantenimientos con bajo esfuerzo.

En una arquitectura por capas, cada capa o nivel se le confía una misión simple, lo que permite el diseño de arquitecturas escalables que pueden ampliarse con facilidad en caso de que las necesidades aumenten. Cada grupo o persona de trabajo durante un mantenimiento o desarrollo puede estar totalmente abstraído del resto de capas y centrarse únicamente en la capa sobre la que está trabajando.

Para este proyecto se ha elegido, dentro de la arquitectura cliente/servidor, un modelo de tres capas. El modelo de capas es una técnica software para separar las diferentes partes de la aplicación, con el objetivo de mejorar su rendimiento, mantenimiento y su funcionalidad.

Esta separación de las diferentes partes hace que los cambios en cualquiera de las capas no afecten o afecten mínimamente a otras capas en las que está dividida la aplicación.

Para la elaboración de este proyecto se ha realizado la abstracción en un modelo de tres capas. Las cuales se tratan:

- Capa de presentación. Es la capa que el usuario visualiza, encargada de presentar y capturar la información correspondiente al usuario que la visualiza. Esta capa se comunica únicamente con la capa de negocio. Es la interfaz gráfica de la aplicación Web.
- Capa de negocio o lógica. Esta capa recibe las peticiones del usuario y envía las respuestas tras el proceso. En esta capa se establecen todas las reglas que deben cumplirse. Se comunica con la capa de presentación para recibir las solicitudes y presentar los resultados, también se comunica con la capa de persistencia o de

datos para solicitar al gestor de la base de datos para recuperar, modificar o insertar datos en la base de datos.

- Capa de persistencia o de datos. Es donde residen los datos y la encargada de acceder a ellos. Está formada por uno o más gestores de datos que realizan todo el almacenamiento de datos, reciben solicitudes de almacenamiento o recuperación desde la capa de negocio.

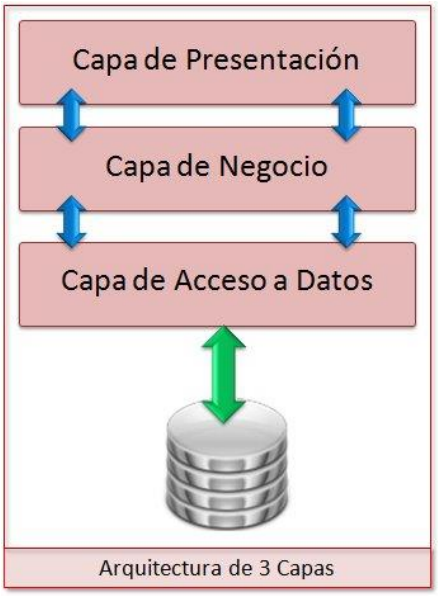


Ilustración 11: Esquema de la arquitectura de tres capas

4.7 PATRÓN MODELO VISTA CONTROLADOR

Dentro del modelo de arquitectura en tres capas uno de los patrones de diseño de software más utilizado es el patrón Modelo Vista Controlador (MVC). Su característica primordial es la separación de la aplicación en tres capas distintas, el Modelo, la Vista y el Controlador.

Se define como un estilo de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos. Se trata de una guía para el diseño de arquitecturas de aplicaciones que mantienen una gran interactividad con los usuarios.

Este patrón organiza la aplicación en tres modelos separados. El primero es un modelo que representa los datos de la aplicación y sus reglas de negocio, el segundo es un conjunto de vistas que representan los formularios de entrada y salida de información y el tercero es un conjunto de controladores que procesan las peticiones de los usuarios y controla el flujo de ejecución del sistema.

El siguiente gráfico básico, nos muestra la interacción entre las tres capas:

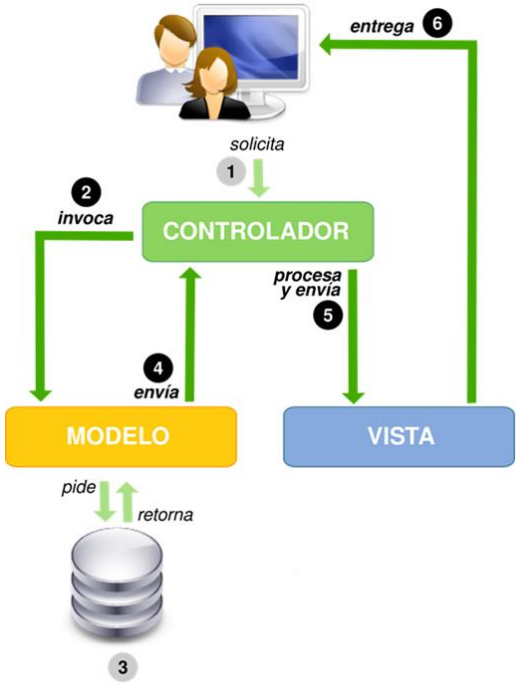


Ilustración 12: Interacción entre capas del patrón MVC

Por tanto, el patrón MVC divide la arquitectura de una aplicación en tres tipos de clases que conforman las tres capas del sistema. Cada clase correspondiente a una capa contará con las siguientes responsabilidades:

- Clase Modelo.
    - Procesa peticiones del sistema.
    - Genera datos de respuesta del sistema.
    - Gestiona y almacena la información.
- Clase Vista.
    - Genera peticiones.
    - Muestra respuestas del sistema.

- Clase Controlador:
    - Gestiona el procesamiento de peticiones.
    - Gestiona las respuestas del sistema.

4.8 SPRING FRAMEWORK

Como tecnología base del proyecto se ha utilizado Spring, el cual se trata de un *framework* o entorno de desarrollo. Un *framework* es una estructura de software compuesta de componentes personalizables e intercambiables para el desarrollo de una aplicación. En otras palabras, un *framework* se puede considerar como una aplicación genérica incompleta y configurable a la que podemos añadirle las últimas piezas para construir una aplicación concreta.

Spring proporciona la base del proyecto, es decir, proporciona los medios para generar una estructura correspondiente, en este caso un patrón Modelo Vista Controlador y además permite la integración de otros *frameworks* y entornos tecnológicos en la estructura formada.

La misión de Spring es doble, por un lado establece la estructura del proyecto (dependencias, clases, configuraciones) y por otro lado permite la integración sencilla de otras tecnologías propias de la organización y ajenas, que complementen la estructura del proyecto.

La organización de Spring ofrece como marco del *framework* que desarrollan diversos módulos acoplables entre sí que aumenten la funcionalidad del sistema que desarrollan. A continuación se detallan aquellos utilizados para el proyecto.

4.8.1 SPRING MVC

Spring MVC es uno de uno de los módulos del Framework de Spring, y como su propio nombre nos indica, que implementa una arquitectura Modelo Vista Controlador previamente explicada.

Ilustración 13: Logotipo de Spring Framework

Spring MVC brinda un patrón Modelo Vista Controlador para entornos Webs bastante flexibles y configurables. Para la configuración de la aplicación y de las propias capas del patrón Spring MVC provee diversas opciones. La forma más popular es usando archivos XML, esta es la forma más tradicional y soportada desde la primera versión de Spring. Con la introducción de las anotaciones Java Bean en Java 5 y a partir de la versión Spring 2.5 se introduce un amplio soporte para configurar una aplicación Web mediante anotaciones. Para este proyecto hacemos uso de ambas formas.

A pesar de la complejidad de las operaciones internas del módulo de Spring MVC, la abstracción que proporciona permite un fácil tratamiento. Más adelante se explica el funcionamiento interno detalladamente.

Spring MVC cuenta con un conjunto de clases que implementan y proporcionan de manera totalmente intrínseca al usuario el patrón Modelo Vista Controlador. En la siguiente ilustración se pueden observar las clases que conforman Spring MVC:

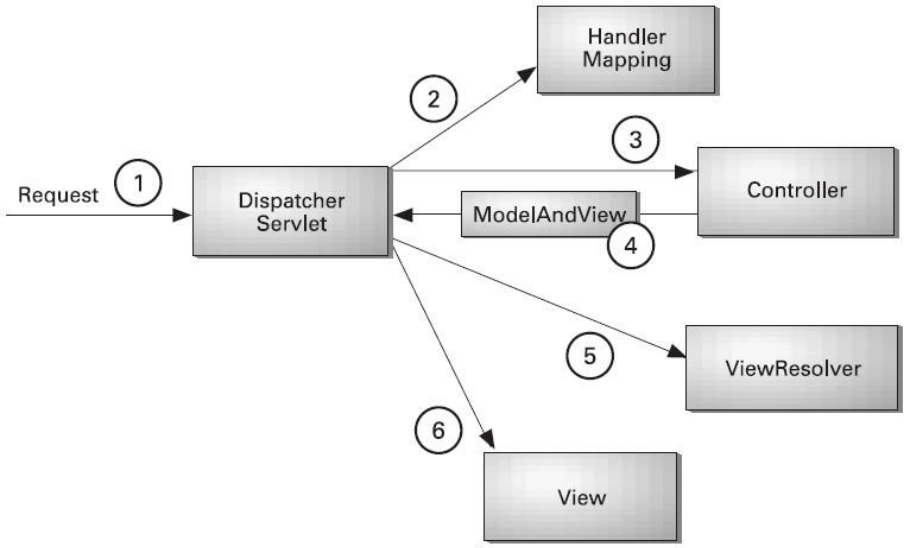


Ilustración 14: Clases e interacción en Spring MVC

Ahora se describirá el funcionamiento de las diferentes clases, a través de un ejemplo de petición.

La clase DispatcherServlet está en el *front controller* y es responsable de delegar y coordinar el control entre varias interfaces en la fase de ejecución durante una petición Http. Es el distribuidor de *servlets*. Cuando el navegador envía una petición se encarga de recoger esta petición (paso 1) para pasárselo al controlador de mapeos (paso 2). Este comprueba que dicha petición este mapeada y retorna el controlador asociado a dicha petición al distribuidor de *servlets*.

Una vez que sabemos qué controlador necesita, el distribuidor de *servlets* pasa el control a dicho controlador (paso 3) para que este se encargue de realizar toda la lógica de negocio de nuestra aplicación. Este controlador devolverá un objeto "Modelo y Vista" (paso 4), el modelo son la información que deseamos mostrar y la vista donde deseamos mostrar dicha información.

Una vez el distribuidor de *servlets* tiene el objeto modelo y vista tendrá que asociar el nombre de la vista retornada por el controlador, con una vista concreta, es decir una página JSP o HTML. (Paso 5). Una vez resulto esto, el distribuidor de *servlets* tendrá que servir la vista. Por último el distribuidor de *servlets* tendrá que pasar a la vista el modelo, es decir los datos a presentar, y mostrar la vista (paso 6).

En el mercado actual existen multitud de *frameworks* para implementar el patrón Modelo Vista Controlador, Spring MVC proporciona algunas ventajas frente a otros *frameworks*:

- Ofrece una división limpia entre controlador, modelo y vista.
- Es muy flexible, ya que implementa toda su estructura mediante interfaces.
- Provee interceptores y controladores que permiten interpretar y adaptar el comportamiento común en el manejo de múltiples peticiones.
- Los controladores de Spring MVC se configuran mediante líneas de código como los demás objetos, lo cual los hace fácilmente *testeables* e integrables con otros objetos que estén en el contexto de Spring, y por tanto sean manejables por éste.
- Las partes de Spring MVC son más fácilmente *testeables*, debido a que evita la herencia de una clase de manera forzosa y una dependencia directa con el *controller* del *servlet* que despacha las peticiones.
- Cuenta con una interfaz bien definida para la capa de negocio.

4.8.2 SPRING DATA

Spring Data es otro de los módulos de Spring. Se trata de proyecto de SpringSource cuyo propósito es unificar y facilitar el acceso a distintos tipos de tecnologías de persistencia, tanto a bases de datos relacionales, como a las del tipo NoSQL utilizada en este proyecto.

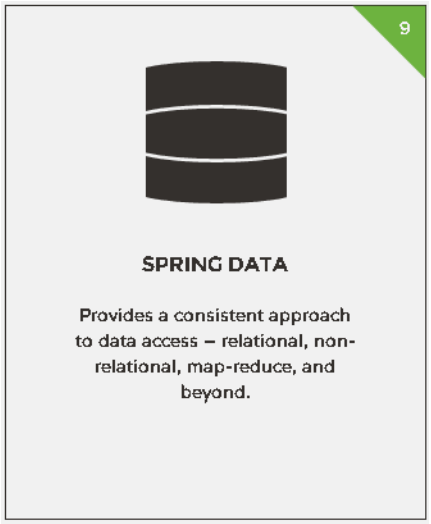


Ilustración 15: Logotipo de Spring Data

Spring Data proporciona un soporte para simplificar implementaciones de la capa de acceso a los datos de un sistema, unificando configuraciones y creando una jerarquía de excepciones común para todas ellas. Recientemente cubre el soporte necesario para distintas tecnologías de bases de datos NoSQL y, además, integra las tecnologías de acceso a datos tradicionales, simplificando el trabajo a la hora de crear implementaciones concretas.

El proyecto de Spring Data cuenta a su vez de diversos módulos, cada uno de ellos especializado para la tecnología de persistencia de datos que queremos integrar. Para este proyecto se utiliza la base de datos de tipo NoSQL, MongoDB, por consiguiente para este proyecto se hace uso del módulo de Spring data denominado **Spring-data-Mongodb**.

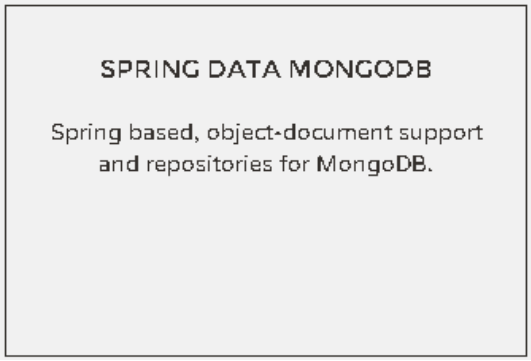


Ilustración 16: Logotipo de Spring Data MongoDB

Spring-data-mongoDB se trata de un proyecto que provee integración con la base de datos NoSQl documental de MongoDB. Ofrece diversas funciones mediante anotaciones y uso de modelos de clases Java denominadas POJO (Plain Old Java Object). Estas son clases que no implementan funciones, únicamente representan objetos. A través de Spring-data-MongoDB y clases Java POJOs pueden desarrollarse capas de acceso a datos, incluyendo su escritura y lectura.

Entre sus características:

- Ofrece soporte a la configuración a través de clases Java, XML y/o anotaciones Java.
- Ayuda a incrementar la productividad de las operaciones de MongoDB, integrando mapeos de objetos entre documentos y POJOs.
- Traslada las excepciones producidas en Spring a excepciones Java.
- Mapeo de objetos entre Java y Spring y conversión de servicios enriquecidos.
- Mapeo y persistencia de todos los eventos del ciclo de vida en Spring.
- Utilización de Java Query, Criteria y DSLs para la construcción de consultas.
- Implementación automática de una interfaz y repositorio con soporte para métodos de búsqueda.
- Almacenamiento de persistencia, soportando entidades JPA.
- Integración geoespacial (para formatos de dos dimensiones).
- Inauguración de Map-Reduce.
- Soporte GridFS.
- Etc.

4.9 PIVOTAL TC SERVER

Como servidor Web para el despliegue de la aplicación Web se ha optado, por su sencillez y compatibilidad de uso con Spring Tool Suite, Pivotal TC Server, en su versión Developer Edition v3.0 como servidor Web.

Pivotal TC Server se trata de una versión empresarial de Apache Tomcat, el servidor de aplicaciones Web más utilizado. Pivotal TC Server está diseñado para convertirse en el

reemplazo de Apache Tomcat, asegurando una ruta de actualizaciones para aplicaciones software a medida y comerciales ya certificadas para Tomcat existente.

La misión de Pivotal TC es mantener este nivel de compatibilidad de Tomcat y permitir a sus usuarios agregar nuevas funcionalidades importantes para el negocio y funcionar de manera más eficaz, gestionando sus aplicaciones con la menor cantidad de esfuerzo.

Respecto al funcionamiento, trabaja como un contenedor de servlets y JSPs (Java Server Pages), al igual que Tomcat. Implementa las especificaciones de los *servlets* y de JavaServer Pages (JSP) de Sun Microsystems. Como contenedor de *servlets* utiliza Catalina.

4.10 OPENLAYERS 3

Para la implementación de mapas gráficos interactivos en la aplicación Web desarrollada, se ha utilizado OpenLayers en su última versión. OpenLayers es una biblioteca de JavaScript de código abierto bajo una derivación de la licencia BSD para mostrar mapas interactivos en los navegadores Web.

Ilustración 17: Logotipo de OpenLayers 3.0

OpenLayers ofrece un API para acceder a diferentes fuentes de información cartográfica en la red: Web Map Services, Mapas comerciales (tipo Google Maps, Bing, Yahoo), Web Features Services, distintos formatos vectoriales, mapas de OpenStreetMap, etc.

Inicialmente fue desarrollado por la organización MetaCarta en junio del 2006. Desde el noviembre del 2007 este proyecto forma parte de los proyectos de Open Source Geospatial Foundation. Actualmente el desarrollo y el soporte corren a cargo de la comunidad de colaboradores existente.

La ventaja de OpenLayers para desarrollar mapas, es que al hacer uso de JavaScript no es necesario llevar a cabo la instalación de ninguna API en nuestro equipo y es compatible con multitud de plataformas al ser fácilmente visualizables desde cualquier navegador.

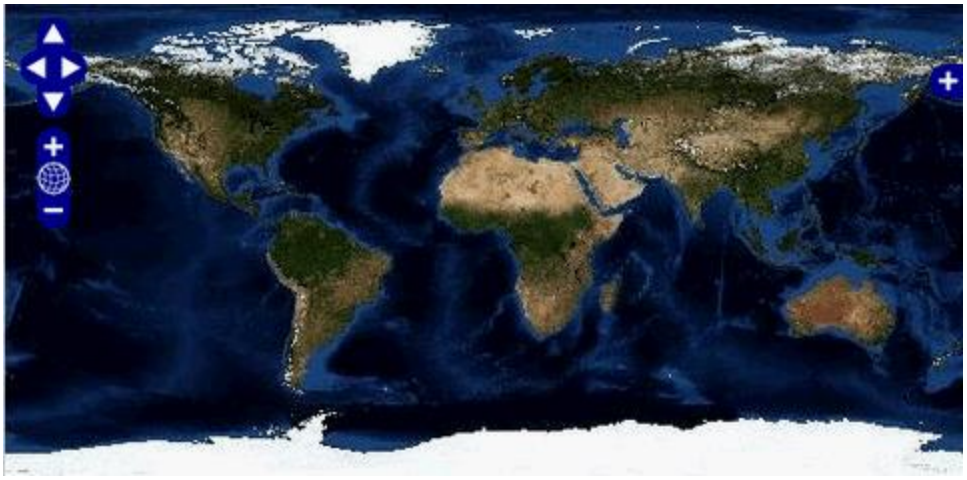


Ilustración 18: Mapa gráfico desarrollado mediante OpenLayers 3.0

Otra de las ventajas de los mapas gráficos desarrollados por OpenLayers es que permiten una alta personalización. Utilizando JavaScript como lenguaje de programación se puede implementar nuevas funcionalidades y elementos.

4.11 SPRING TOOL SUITE

La herramienta IDE seleccionada para la realización de la práctica ha sido **Spring Tool Suite (STS) en su versión 3.6.3**. Se trata de un "todo en uno" que facilita el desarrollo de aplicaciones Web. Proporciona una suite de desarrollo lista para ser usada integrando un soporte para multitud de lenguajes de programación, framework de desarrollo y procesos en general. Combina las herramientas Java, Web y Java EE de Eclipse. Además, de forma predeterminada, integra tecnologías como Gradle o Pivotal TC Server, este último utilizado como servidor Web para el desarrollo.

Ilustración 19: Logotipo de Spring Tool Suite (STS)

Spring Tool Suite además ofrece facilidades a la hora de crear un proyecto dinámico Web basado en Spring MVC. Spring Tool Suite proporciona un entorno *ready-to-use* para implementar, depurar, ejecutar y desplegar las aplicaciones Spring, incluyendo integraciones para Pivotal tc Server, Pivotal Cloud Foundry, Git, Maven, AspectJ, y adaptado a las últimas versiones de Eclipse.

Spring Tool Suite soporta el despliegue de aplicaciones tanto en servidores locales, virtuales y en la nube. Es de libre acceso para el desarrollo y uso en operaciones internas sin límite de tiempo, completamente de código abierto y licenciada bajo los términos de la Licencia Pública Eclipse.

4.12 MAVEN

Para la realización de la aplicación Web se ha usado Maven instalado sobre Spring Tool Suite. Maven es una herramienta que se integra perfectamente con el IDE Spring Tool Suite o Eclipse. Maven utiliza un Project Object Model (POM) para describir sistemas software, sus dependencias de otros módulos, componentes externos, y el orden de construcción de los elementos. Realiza también la compilación del código y su empaquetado.

Ilustración 20: Logotipo de Maven

La principal característica de Maven es que está listo para usar en red. El motor incluido en su núcleo puede dinámicamente descargar librerías y plugins de un repositorio. El repositorio que utiliza es el mismo que provee acceso a muchas versiones de diferentes proyectos Open Source en Java, de Apache y otras organizaciones y desarrolladores. Maven proporciona un gran soporte para las librerías de Spring framework.

Para explicar el funcionamiento de Maven, es necesario entender el concepto de librería, el cual es un concepto en ocasiones limitado. Para el proyecto realizado ha sido necesario integrar múltiples librerías (JSTL, Spring-data-MongoDB, Spring-MVC , Spring-core, etc.). En el caso de necesitar integrar una nueva librería en nuestro proyecto, denominada "librería A". No nos vale simplemente con querer utilizar la librería sino que además necesitamos saber que versión exacta de ella necesitamos.

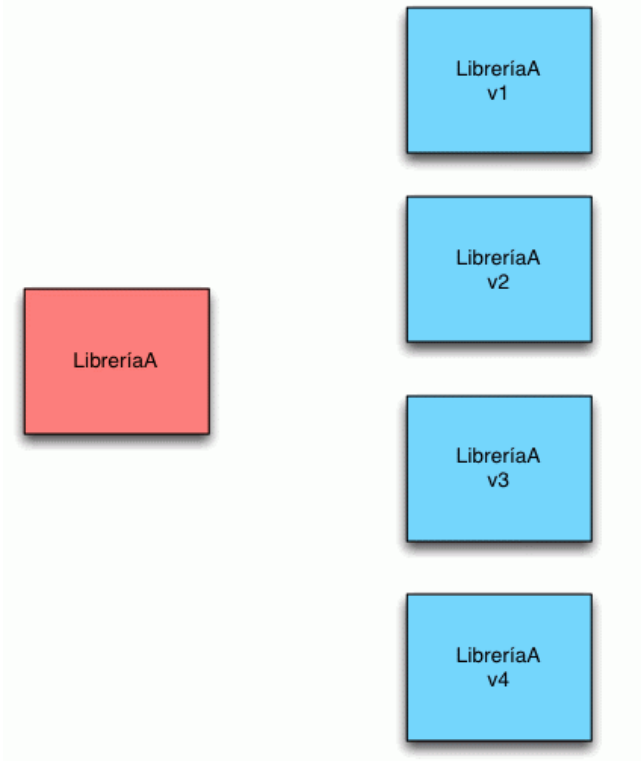


Ilustración 21: Simulación de esquema de dependencias de versiones de librerías

Sin embargo, la librería a añadir puede depender de otras librerías para funcionar de forma óptima o simplemente para funcionar. Así pues necesitamos más información para gestionar todo de forma correcta. Entonces es necesario conocer las dependencias de la librería a añadir en función de su versión.

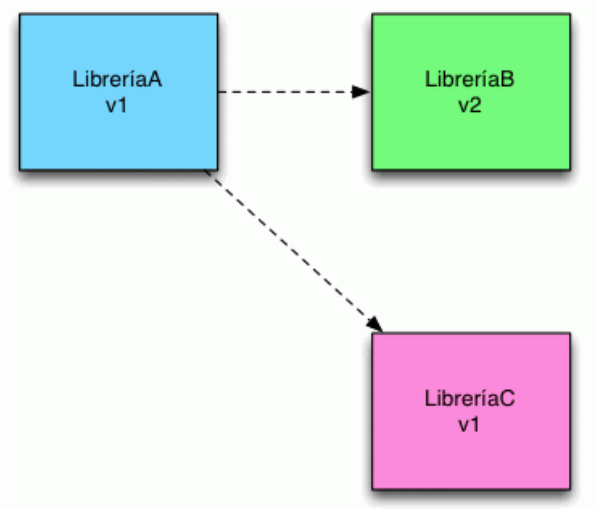


Ilustración 22: Simulación de esquemas de dependencias de librerías

Maven solventa este problema a través del concepto de artefacto. Un artefacto puede verse como una librería con información adicional (aunque agrupa más conceptos). Un artefacto contiene las clases propias de la librería pero además incluye toda la información necesaria para su correcta gestión (grupo, versión, dependencias etc.).

Ilustración 23: Esquema de artefacto

Transformando las librerías en artefactos, podemos conocer exactamente qué librerías debemos implementar, las dependencias de las mismas, las versiones permitidas, su nombre, etc.

Maven ha sido una herramienta vital en este proyecto, pues la Aplicación Web desarrollada integra multitud de tecnologías. Cada tecnología integrada conlleva una librería, con dependencias de otras librerías y a su vez con compatibilidades e incompatibilidades de versiones. Sin un software que gestionara estos elementos, se hubiera convertido en un arduo trabajo la integración de todas las tecnológicas que conforman actualmente el proyecto.

5 DESARROLLO DEL PROYECTO

5.1 ESTUDIO DE VIABILIDAD

La viabilidad del proyecto depende fundamentalmente de dos elementos. Por un lado, la capacidad de desarrollar una aplicación Web partiendo desde una idea y ser capaz de integrar una serie de *frameworks*, entornos y patrones de desarrollo que proporcionen una aplicación Web con alta escalabilidad para futuros proyectos y que a su vez sirva de aprendizaje y estudio para el desarrollo de aplicaciones Web.

Por otro lado, utilizar y estudiar en profundidad MongoDB, a fin de proporcionar una base de datos eficiente que almacene un gran volumen de datos y permita a su vez, mostrar en el menor tiempo posible los datos al usuario que haga uso de la aplicación Web. Asimismo, profundizar en el estudio de bases de datos NoSQL, en concreto en MongoDB, más allá de su usabilidad y características, sino también su modelo, estructura de datos y estratégicas de indexación.

Una de las razones para desarrollar este proyecto es proporcionar información gráfica y ordenada sobre la información recogida por el satélite QuikSCAT mediante el acceso a un portal Web.

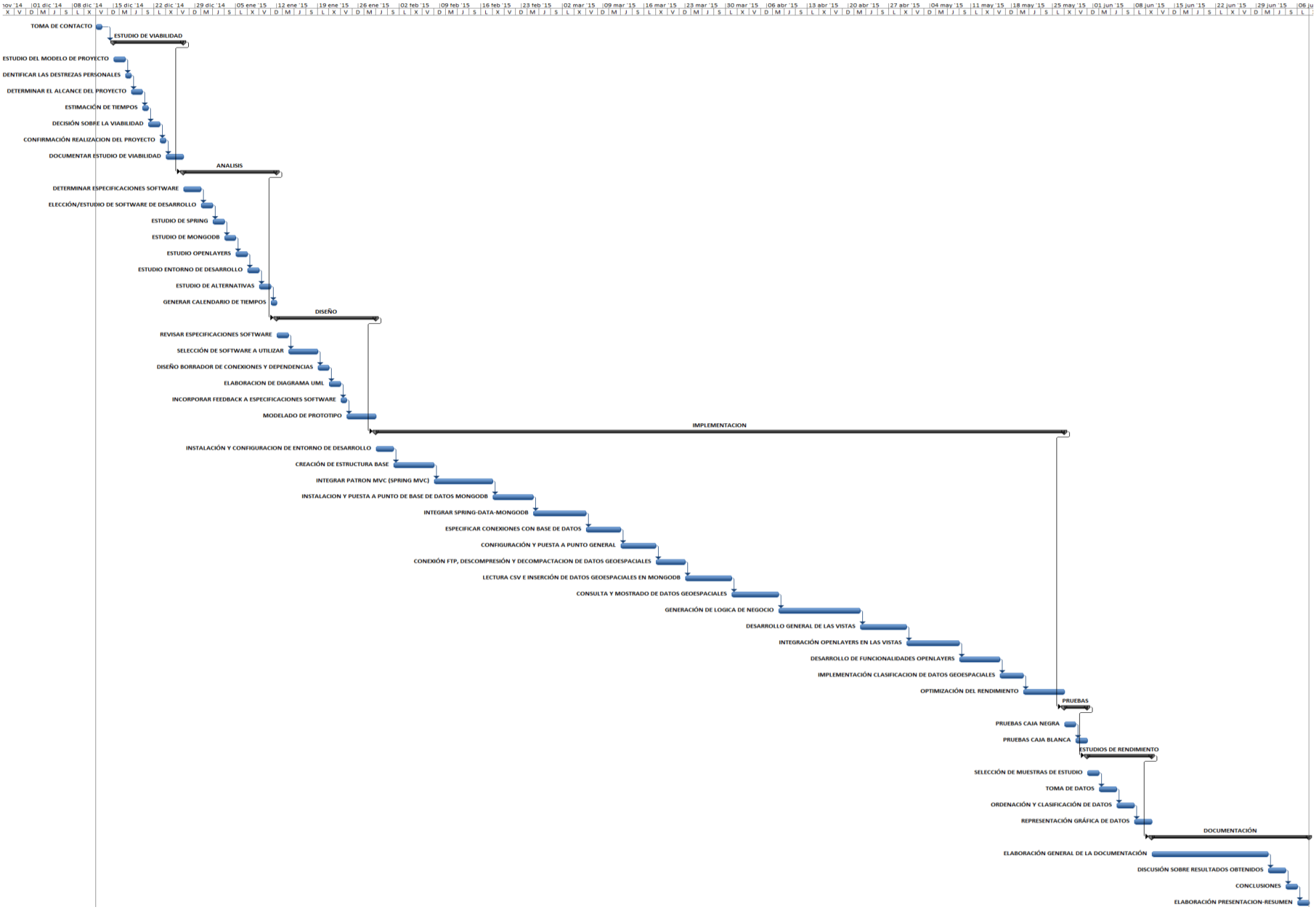
La información existente actualmente y de la cual tendrá que hacer uso la aplicación Web ocupa un gran volumen de datos. Sin duda alguna, la viabilidad del proyecto depende de desarrollar una aplicación Web capaz hacer uso de esa ingente cantidad de datos y ofrecérsela al usuario de forma ordenada y rápida. Razón por la cual será necesario hacer uso de un base de datos orientada a grandes volúmenes de información (Big data) como MongoDB.

5.2 PLANIFICACIÓN

Para la elaboración de una planificación, que permitiese desarrollar en los plazos acordados el proyecto, se ha utilizado un meticuloso cronograma elaborado mediante Microsoft Project. Se trata de un software de administración de proyectos diseñado, desarrollado y comercializado por Microsoft para asistir a administradores de proyectos

en el desarrollo de planes, asignación de recursos a tareas, dar seguimiento al progreso, administrar presupuesto y analizar cargas de trabajo.

Para la elaboración del proyecto se ha establecido la planificación de acuerdo al siguiente diagrama de Gantt.


TOMA DE CONTACTO
ESTUDIO DE VIABILIDAD
ESTUDIO DEL MODELO DE PROYECTO
DENTIFICAR LAS DESTREZAS PERSONALES
DETERMINAR EL ALCANCE DEL PROYECTO
ESTIMACIÓN DE TIEMPOS
DECISIÓN SOBRE LA VIABILIDAD
CONFIRMACIÓN REALIZACION DEL PROYECTO
DOCUMENTAR ESTUDIO DE VIABILIDAD
ANALISIS
DETERMINAR ESPECIFICACIONES SOFTWARE
ELECCIÓN/ESTUDIO DE SOFTWARE DE DESARROLLO
ESTUDIO DE SPRING
ESTUDIO DE MONGODB
ESTUDIO OPENLAYERS
ESTUDIO ENTORNO DE DESARROLLO
ESTUDIO DE ALTERNATIVAS
GENERAR CALENDARIO DE TIEMPOS
DISEÑO
REVISAR ESPECIFICACIONES SOFTWARE
SELECCIÓN DE SOFTWARE A UTILIZAR
DISEÑO BORRADOR DE CONEXIONES Y DEPENDENCIAS
ELABORACION DE DIAGRAMA UML
INCORPORAR FEEDBACK A ESPECIFICACIONES SOFTWARE
MODELADO DE PROTOTIPO
IMPLEMENTACION
INSTALACIÓN Y CONFIGURACION DE ENTORNO DE DESARROLLO
CREACIÓN DE ESTRUCTURA BASE
INTEGRAR PATRON MVC (SPRING MVC)
INSTALACION Y PUESTA A PUNTO DE BASE DE DATOS MONGODB
INTEGRAR SPRING-DATA-MONGODB
ESPECIFICAR CONEXIONES CON BASE DE DATOS
CONFIGURACIÓN Y PUESTA A PUNTO GENERAL
CONEXIÓN FTP, DESCOMPRESIÓN Y DECOMPACTACION DE DATOS GEOESPACIALES
LECTURA CSV E INSERCIÓN DE DATOS GEOESPACIALES EN MONGODB
CONSULTA Y MOSTRADO DE DATOS GEOESPACIALES
GENERACIÓN DE LOGICA DE NEGOCIO
DESARROLLO GENERAL DE LAS VISTAS
INTEGRACIÓN OPENLAYERS EN LAS VISTAS
DESARROLLO DE FUNCIONALIDADES OPENLAYERS
IMPLEMENTACIÓN CLASIFICACION DE DATOS GEOESPACIALES
OPTIMIZACIÓN DEL RENDIMIENTO
PRUEBAS
PRUEBAS CAJA NEGRA
PRUEBAS CAJA BLANCA
ESTUDIOS DE RENDIMIENTO
SELECCIÓN DE MUESTRAS DE ESTUDIO
TOMA DE DATOS
ORDENACIÓN Y CLASIFICACIÓN DE DATOS
REPRESENTACIÓN GRÁFICA DE DATOS
DOCUMENTACIÓN
ELABORACIÓN GENERAL DE LA DOCUMENTACIÓN
DISCUSIÓN SOBRE RESULTADOS OBTENIDOS
CONCLUSIONES
ELABORACIÓN PRESENTACION-RESUMEN

5.3 ANALÍSIS DE LA APLICACIÓN WEB

La aplicación Web se trata de un proyecto elaborado mediante el IDE Spring Tool Suite basado en Eclipse. El proyecto está compuesto por una serie de librerías, archivos fuentes y ficheros Java™.

El proyecto está formado por una totalidad de unas 5000 líneas de código que permiten integrar diversas tecnologías de gran relevancia y desarrollar una gran funcionalidad.

Los código fuentes que conforman el proyecto se encuentran estructurados en paquetes, clases y ficheros, de acuerdo al estándar de programación estructurada en Java™. Para mantener una estructura que ayude a la organización del proyecto, se han estructurado los elementos de acuerdo al patrón Modelo Vista Controlador previamente explicado y utilizado en este proyecto.

La organización de los ficheros fuentes se organiza en:

- Elementos correspondientes al modelo:
    - Paquete correspondiente al modelo.
        - Modelo referente a un punto geoespacial (SpatialPoint.class)
    - Paquete correspondiente a servicios del modelo.
        - Ficheros que implementan funciones referentes a servicios FTP (FTPService.class).
        - Ficheros que implementan funciones referentes a servicios sobre datos geoespaciales (GeoSpatialService.class).
- Elementos correspondientes al controlador
    - Paquete correspondiente al controlador.
        - Fichero referente al controlador de peticiones (controller.class)
- Elementos correspondientes a las vistas.
    - Paquete correspondiente a la vista.
        - Ficheros JSP que representan vistas (index.JSP, index2.JSP).
        - Ficheros de estilos CSS que proporcionan el diseño de las vistas (form.CSS).

A continuación se presenta el diagrama correspondiente a la aplicación Web desarrollada y correspondiente a los métodos, clases y paquetes que conforman su estructura.

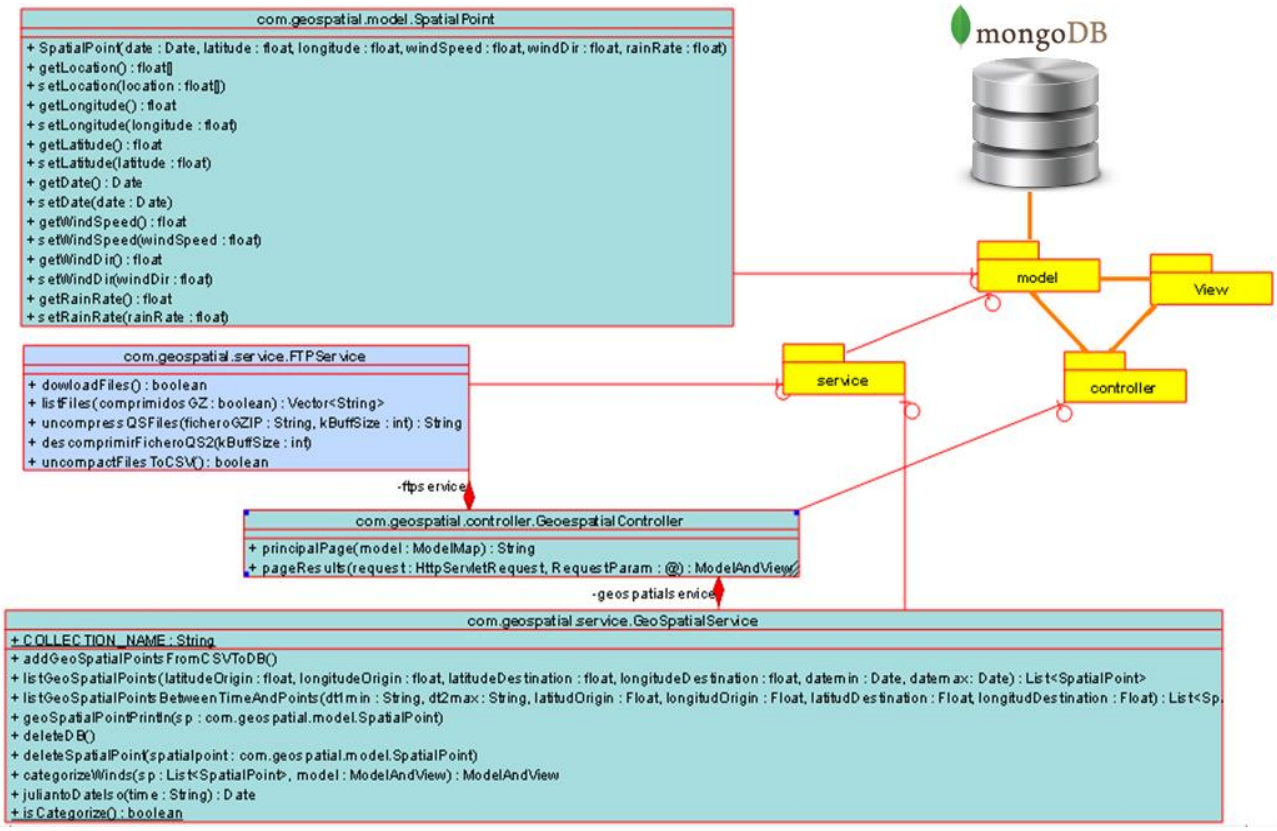


Ilustración 24: Diagrama UML de la aplicación Web desarrollada

La aplicación además cuenta con un fichero ejecutable externo a la aplicación, encargado de procesar de forma adecuada los archivos que almacenan los datos geoespaciales descargados vía FTP del centro de descargas de la NASA procediendo a su interpretación y conversión de formato HDF a ficheros en formato CSV.

Seguidamente, se detalla la misión y funcionalidad de cada paquete del proyecto:

- El Modelo es la representación de la información con la cual la Aplicación Web opera, se encarga gestionar toda la información y los accesos a su almacén, concretamente a la base de datos de MongoDB y a los datos geoespaciales en ella almacenada.

  Es misión del modelo proporcionar a la "vista" aquella parte de la información que en cada momento se le solicita para que sea mostrada a los usuarios, concretamente consultas de datos geoespaciales a través rangos espaciales y temporales. Las peticiones de consulta realizadas desde las vistas al "modelo"

son interceptadas por el controlador que sirve de enlace para implementar las diversas necesidades del desarrollo.

Para realizar una implementación correcta, y no solo funcional, que permita cumplir con las buenas prácticas de desarrollo a fin de proporcionar software fácil de mantener y escalable, se ha simplificado a su vez el modelo en:

- Capa de servicios (services), se tratan de las clases que implementan la lógica de negocio de la aplicación. Se encarga de codificar las reglas de cómo la información es creada, mostrada y modificada. La capa de servicios es la encargada de realizar los tratamiento necesarios con los datos geoespaciales respecto a su consulta, gestión, inserción o borrado.

  Para mayor simplicidad aun, cada clase de la capa de servicios implementa una lógica de negocio concreta, así, la clase FTPService, se encarga del tratamiento de los datos para su recuperación y adaptación, proporcionando el acceso y descarga de los ficheros de los datos geoespaciales desde los servidores FTP de la NASA y su posterior descomprensión, interpretación y descompactación en ficheros en formato CSV. La clase GeoespatialService se encarga de ofrecer funciones orientadas al tratamiento de los datos geoespaciales; lectura, adaptación e inserción y consulta en el almacén de datos de MongoDB; así como su consulta y clasificación.

- Modelo, se trata de la propia representación de los datos. Realiza la codificación de los "objetos" reales de los que hace uso la aplicación Web, en este caso de los datos geoespaciales. La función del modelo es simple, proveer una estructura para albergar un elemento correspondiente a los datos y permitir la inserción y consulta de sus atributos. El modelo en la aplicación Web desarrollada se trata de clase SpatialPoint, la cual cuenta con la estructura necesaria para albergar una medición de los datos geoespaciales, dicha estructura está formada por: fecha, latitud, longitud, velocidad del viento, dirección del viento y grado de precipitaciones.

- El Controlador, se encarga de responder a los eventos realizados por el usuario, para ello invoca peticiones al modelo cuando se hace alguna solicitud sobre la información.
- La Vista, presenta el modelo, es decir, la información procesada a través de la lógica de negocio en un formato adecuado para poder ser interactuado desde la interfaz de usuario. Necesita de la información procesada por el modelo para poder representar la información como salida.

  En las vistas desarrolladas se ha utilizado, entre otras tecnologías, OpenLayers en su versión 3.0, esta biblioteca JavaScript a través de la integración de un mapa terrestre interactivo se encarga gráficamente de capturar las peticiones del usuario y a su vez, de representar los datos geoespaciales consultados, clasificados y proporcionados por el modelo.

El siguiente flujo simplificado, correspondiente a una interacción habitual de un usuario con la Aplicación Web desarrollada, representa la interacción de los distintos componentes que conforman el proyecto:

1. El usuario interactúa con la interfaz gráfica, realizando la acción de consulta de datos geoespaciales en un rango espacial y temporal específico.
2. El controlador recibe, por parte de los objetos de la interfaz-vista que implementa de forma intrínseca Spring MVC, la notificación de la acción solicitada por el usuario. El controlador gestiona el evento que le llega a través del gestor de eventos.
3. El controlador accede al modelo, procesando la consulta transmitida desde la vista de acuerdo a la lógica de negocio implementada, en este caso realizando la consulta al almacén de datos de MongoDB. Retornados los datos geoespaciales correspondientes a la consulta de rango espacial y temporal especificada por el usuario desde el repositorio y procedida su clasificación en función de la intensidad de los vientos, se ponen a disposición del controlador los datos en forma de objeto.
4. El controlador delega los objetos del modelo a la vista, encargándose de desplegar la interfaz de usuario.

5. La vista obtiene sus datos del modelo para generar la interfaz apropiada para el usuario, en este caso la información geoespacial relativa a una consulta por rango. Recalcar que el modelo no tiene en ningún momento interacción directa sobre la vista
6. La interfaz de usuario espera nuevas interacciones del usuario, comenzando el ciclo nuevamente.

6 RESULTADOS OBTENIDOS

6.1 TOMA DE DATOS

Uno de los objetivos fundamentales de este proyecto, además de desarrollar una aplicación Web que facilitase el estudio de los datos geoespaciales, era realizar un estudio exhaustivo sobre el rendimiento de las diferentes técnicas de indexación que permite MongoDB.

Como ya se ha explicado en este documento, especialmente en el punto 3.3.5 "Técnicas de indexación sobre el modelo de datos", MongoDB permite distintas estrategias de indexación a partir de:

- Índices simples
- Índice compuestos
- Índice geoespaciales

Los datos sobre los que tratamos, son índices de tres dimensiones. Como MongoDB no permite ningún índice de tres dimensiones de forma nativa, es necesario simularlo estableciendo índices o conjuntos de índices a fin de agilizar las consultas de documentos por rango de búsquedas de acuerdo al espacio (longitud, latitud) y tiempo.

A continuación se detallan las técnicas de indexación que se proponen para ser objeto del estudio y en especial para la toma de los datos:

- Índice simple temporal + Índice simple longitud + Índice simple latitud.
    - Se trata de la técnica de indexación más simple, un índice por elemento de búsqueda.
- Índice geoespacial (longitud, latitud) + Índice simple temporal.
    - Se trata de la técnica de indexación más lógica, por un lado un índice geoespacial para los datos espaciales (longitud y latitud) y por otro lado un índice simple para el tiempo.
- Índice compuesto (temporal, longitud, latitud).
    - Se trata de una técnica de índice no muy intuitiva, construye una única estructura de indexación (árbol B) mediante los atributos utilizados para la consulta rendimiento.
- Índice geoespacial (longitud, latitud) + Índice compuesto (temporal, longitud, latitud).

- Se trata de la técnica, a priori, menos lógica e intuitiva, sin embargo MongoDB permite la utilización de uno o más índice por búsqueda. El objetivo es utilizar el índice compuesto para la búsqueda de rangos en primer lugar por tiempo y luego utilizar para la búsqueda espacial (longitud y latitud) de manera auxiliar y automática por MongoDB el índice geoespacial (longitud y latitud).

Para llevar a cabo la toma de datos de las siguientes técnicas de indexación, se realizarán consultas por rango espacial y temporal que devuelvan un número de documentos previamente conocido. Se establece para la medición la consulta del siguiente número de documentos: 500, 2800, 10.000, 40.000, 117.000, 220.000, 435.00, 700.000 y 1.380.000. De cada consulta se estudiará sus tiempos de respuesta, estableciendo estadísticas y comparativas.

La consulta que se ha realizado es una consulta estándar que se utiliza en la aplicación Web desarrollada para retornar los datos y ser posteriormente representados gráficamente. La consulta para el retorno del número de documentos utilizado es la misma para todas las técnicas de indexación, de forma que no existan variaciones en función de los documentos retornados en la búsqueda.

La muestra de estudio se realizará sobre los datos geoespaciales recogidos por el satélite QuikSCAT correspondientes a 6 meses del año 2005 de medidas, con un tamaño de 28Gbytes y 128.580.556 elementos.

El equipo utilizado para realizar las mediciones se trata de un Intel Core i5 de 2,4Ghz con 6Gb de RAM y una tarjeta gráfica ATI Radeon 5470.

Ahora se indican las mediciones y resultados para cada técnica de indexación en función del número de elementos:

TIEMPOS DE CONSULTA DE 500 ELEMENTOS

- Índice simple temporal + Índice simple longitud + Índice simple latitud:

  3,884 segundos (máx.)

  3,752 segundos (mín.)

3,883 segundos

3,783 segundos

Omitiendo mejor y peor resultando y realizando la media: **3,833 segundos**

- Índice geoespacial (longitud, latitud) + Índice simple temporal:

  1,985 segundos (máx.)

  1,523 segundos (mín.)

  1,857 segundos

  1,951 segundos

  Omitiendo mejor y peor resultando y realizando la media: **1,904 segundos**

- Índice compuesto (temporal, longitud, latitud):

  0,324 segundos (mín.)

  1,221 segundos (máx.)

  0,442 segundos

  0,450 segundos

  Omitiendo mejor y peor resultando y realizando la media: **0,446 segundos**

- Índice geoespacial (longitud, latitud) + Índice compuesto (temporal, longitud, latitud):

  1,501 segundos

  1,336 segundos (mín.)

  1,642 segundos

  1,850 segundos (máx.)

  Omitiendo mejor y peor resultando y realizando la media: **1,529 segundos**

TIEMPOS DE CONSULTA DE 2.800 ELEMENTOS

- Índice simple temporal + Índice simple longitud + Índice simple latitud:

  4,317 segundos

4,885 segundos (máx.)

4,239 segundos (mín.)

4,339 segundos

Omitiendo mejor y peor resultando y realizando la media: **4,328 segundos**

- Índice geoespacial (longitud, latitud) + Índice simple temporal:

  5,183 segundos (mín.)

  5,378 segundos

  5,696 segundos (máx.)

  5,494 segundos

  Omitiendo mejor y peor resultando y realizando la media: **5,436 segundos**

- Índice compuesto (temporal, longitud, latitud):

  0,530 segundos

  0,585 segundos (máx.)

  0,538 segundos

  0,488 segundos (mín.)

  Omitiendo mejor y peor resultando y realizando la media: **0,534 segundos**

- Índice geoespacial (longitud, latitud) + Índice compuesto (temporal, longitud, latitud):

  5,698 segundos

  5,787 segundos (máx.)

  5,481 segundos

  5,146 segundos (mín.)

  Omitiendo mejor y peor resultando y realizando la media: **5,589 segundos**

TIEMPOS DE CONSULTA DE 10.000 ELEMENTOS

- Índice simple temporal + Índice simple longitud + Índice simple latitud:

4,693 segundos (máx.)

4,381 segundos

4,117 segundos (mín.)

4,310 segundos

Omitiendo mejor y peor resultando y realizando la media: **4,345 segundos**

- Índice geoespacial (longitud, latitud) + Índice simple temporal:

  13,454 segundos

  13,619 segundos (máx.)

  13,202 segundos

  13,089 segundos (mín.)

  Omitiendo mejor y peor resultando y realizando la media: **13,328 segundos**

- Índice compuesto (temporal, longitud, latitud):

  0,747 segundos (mín.).

  0,886 segundos

  0,950 segundos (máx.)

  0,772 segundos

  Omitiendo mejor y peor resultando y realizando la media: **0,829 segundos**

- Índice geoespacial (longitud, latitud) + Índice compuesto (temporal, longitud, latitud):

  17,414 segundos (mín.)

  17,453 segundos

  17,257 segundos

  17,791 segundos (máx.)

  Omitiendo mejor y peor resultando y realizando la media: **17,355 segundos**

TIEMPOS DE CONSULTA DE 40.000 ELEMENTOS

- Índice simple temporal + Índice simple longitud + Índice simple latitud:

  5,808 segundos (máx.)

  5,087 segundos

  5,593 segundos

  4,819 segundos (mín.)

  Omitiendo mejor y peor resultando y realizando la media: **5,34 segundos**

- Índice geoespacial (longitud, latitud) + Índice simple temporal:

  24,427 segundos

  24,958 segundos (máx.)

  21,307 segundos (mín.)

  23,666 segundos

  Omitiendo mejor y peor resultando y realizando la media: **24,046 segundos**

- Índice compuesto (temporal, longitud, latitud):

  2,011 segundos

  2,140 segundos (máx.)

  1,892 segundos

  1,818 segundos (mín.)

  Omitiendo mejor y peor resultando y realizando la media: **1,951 segundos**

- Índice geoespacial (longitud, latitud) + Índice compuesto (temporal, longitud, latitud):

  26,793 segundos

  25,092 segundos

  27,211 segundos (máx.)

  24,323 segundos (mín.)

  Omitiendo mejor y peor resultando y realizando la media: **25,942 segundos**

TIEMPOS DE CONSULTA DE 117.000 ELEMENTOS

- Índice simple temporal + Índice simple longitud + Índice simple latitud:

  7,294 segundos (mín.)

  9,033 segundos (máx.)

  7,663 segundos

  7,285 segundos

  Omitiendo mejor y peor resultando y realizando la media: **7,474 segundos**

- Índice geoespacial (longitud, latitud) + Índice simple temporal:

  44,785 segundos

  46,375 segundos

  42,062 segundos (mín.)

  48,974 segundos (máx.)

  Omitiendo mejor y peor resultando y realizando la media: **45,58 segundos**

- Índice compuesto (temporal, longitud, latitud):

  4,630 segundos

  4,231 segundos (mín.)

  5,093 segundos (máx.)

  4,530 segundos

  Omitiendo mejor y peor resultando y realizando la media: **4,58 segundos**

- Índice geoespacial (longitud, latitud) + Índice compuesto (temporal, longitud, latitud):

  50,705 segundos (mín.)

  53,057 segundos

  51,056 segundos

  53,276 segundos (máx.)

  Omitiendo mejor y peor resultando y realizando la media: **52,056 segundos**

TIEMPOS DE CONSULTA DE 220.000 ELEMENTOS

- Índice simple temporal + Índice simple longitud + Índice simple latitud:

  11,309 segundos

  11,775 segundos (máx.)

  10,784 segundos

  10,484 segundos (mín.)

  Omitiendo mejor y peor resultando y realizando la media: **11,046 segundos**

- Índice geoespacial (longitud, latitud) + Índice simple temporal.

  65,673 segundos

  62,041 segundos (mín.)

  65,872 segundos (máx.)

  64,264 segundos

  Omitiendo mejor y peor resultando y realizando la media: **65,013 segundos**

- Índice compuesto (temporal, longitud, latitud):

  8,044 segundos

  7,735 segundos (mín.)

  8,211 segundos (máx.)

  7,883 segundos

  Omitiendo mejor y peor resultando y realizando la media: **7,963 segundos**

- Índice geoespacial (longitud, latitud) + Índice compuesto (temporal, longitud, latitud):

  86,041 segundos (mín.)

  88,254 segundos

  88,342 segundos (máx.)

  87,264 segundos

Omitiendo mejor y peor resultando y realizando la media: **87,759 segundos**

TIEMPOS DE CONSULTA DE 435.000 ELEMENTOS

- Índice simple temporal + Índice simple longitud + Índice simple latitud:

  18,160 segundos

  18,587 segundos (máx.)

  18,118 segundos

  17,943 segundos (mín.)

  Omitiendo mejor y peor resultando y realizando la media: **18,139 segundos**

- Índice geoespacial (longitud, latitud) + Índice simple temporal:

  85,914 segundos (máx.)

  82,149 segundos (mín.)

  83,482 segundos

  85,869 segundos

  Omitiendo mejor y peor resultando y realizando la media: **84,765 segundos**

- Índice compuesto (temporal, longitud, latitud):

  14,336 segundos (máx.)

  14,008 segundos

  13,290 segundos

  13,013 segundos (mín.)

  Omitiendo mejor y peor resultando y realizando la media: **13,649 segundos**

- Índice geoespacial (longitud, latitud) + Índice compuesto (temporal, longitud, latitud):

  135,914 segundos (máx.)

  132,149 segundos (mín.)

  133,482 segundos

135,869 segundos

Omitiendo mejor y peor resultando y realizando la media: **134,765 segundos**

TIEMPOS DE CONSULTA DE 700.000 ELEMENTOS

- Índice simple temporal + Índice simple longitud + Índice simple latitud:

  26,444 segundos (máx.)

  25,700 segundos

  25,389 segundos (mín.)

  26,141 segundos

  Omitiendo mejor y peor resultando y realizando la media: **25,92 segundos**

- Índice geoespacial (longitud, latitud) + Índice simple temporal:

  115,914 segundos (máx.)

  112,149 segundos (mín.)

  113,482 segundos

  115,869 segundos

  Omitiendo mejor y peor resultando y realizando la media: **114,765 segundos**

- Índice compuesto (temporal, longitud, latitud):

  22,741 segundos (máx.)

  21,425 segundos

  21,070 segundos (mín.)

  21,416 segundos

  Omitiendo mejor y peor resultando y realizando la media: **21,42 segundos**

- Índice geoespacial (longitud, latitud) + Índice compuesto (temporal, longitud, latitud):

  147,654 segundos (máx.)

  147,191 segundos (mín.)

153,321 segundos

150,001 segundos

Omitiendo mejor y peor resultando y realizando la media: **151,661 segundos**

TIEMPOS DE CONSULTA DE 1.380.000 ELEMENTOS

- Índice simple temporal + Índice simple longitud + Índice simple latitud:

  54,70 segundos (máx.)

  52,68 segundos

  50,65 segundos (mín.)

  51,89 segundos

  Omitiendo mejor y peor resultando y realizando la media: **52,291 segundos**

- Índice geoespacial (longitud, latitud) + Índice simple temporal:

  147,23 segundos (mín.)

  151,65 segundos (máx.)

  148,87 segundos

  149,03 segundos

  Omitiendo mejor y peor resultando y realizando la media: **148,95 segundos**

- Índice compuesto (temporal, longitud, latitud):

  44,874 segundos (mín.)

  49,234 segundos (máx.)

  41,382 segundos

  42,822 segundos

  Omitiendo mejor y peor resultando y realizando la media: **42,102 segundos**

- Índice geoespacial (longitud, latitud) + Índice compuesto (temporal, longitud, latitud):

  197,23 segundos (mín.)

190,65 segundos (máx.)

198,87 segundos

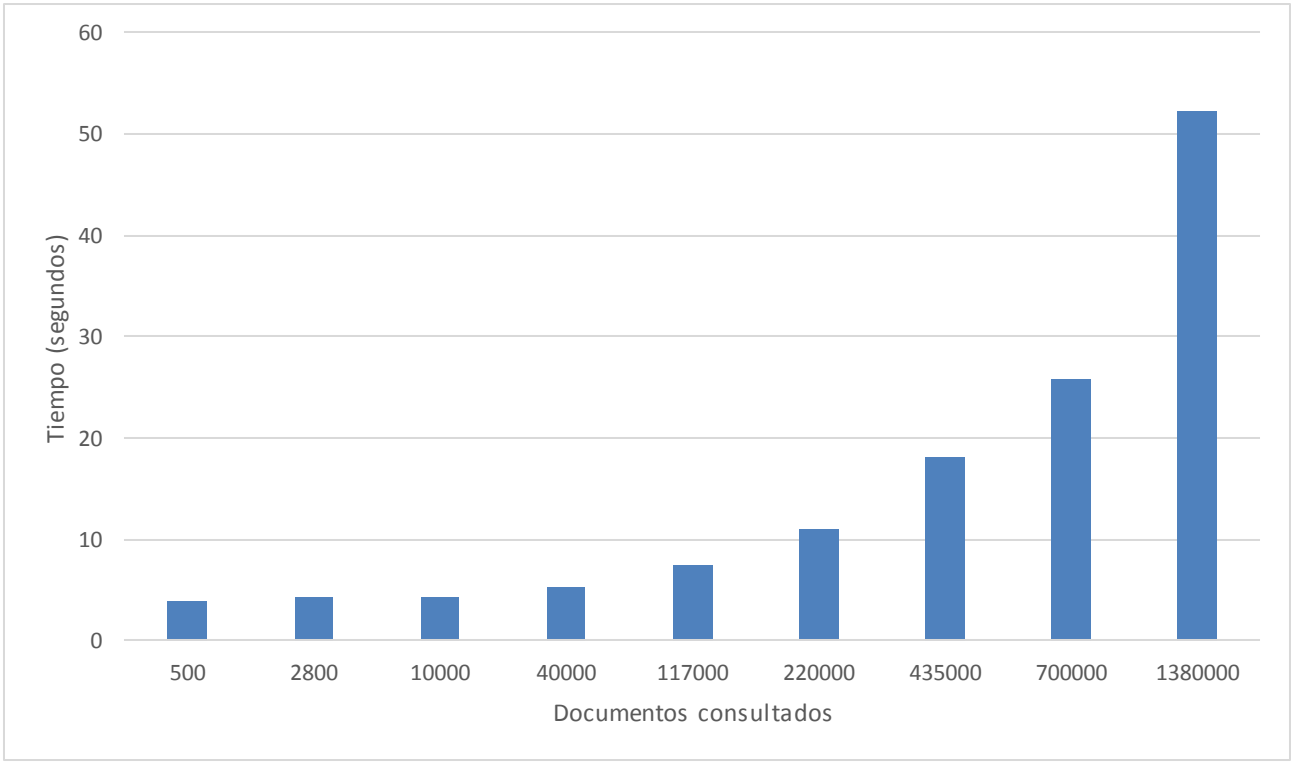
199,03 segundos

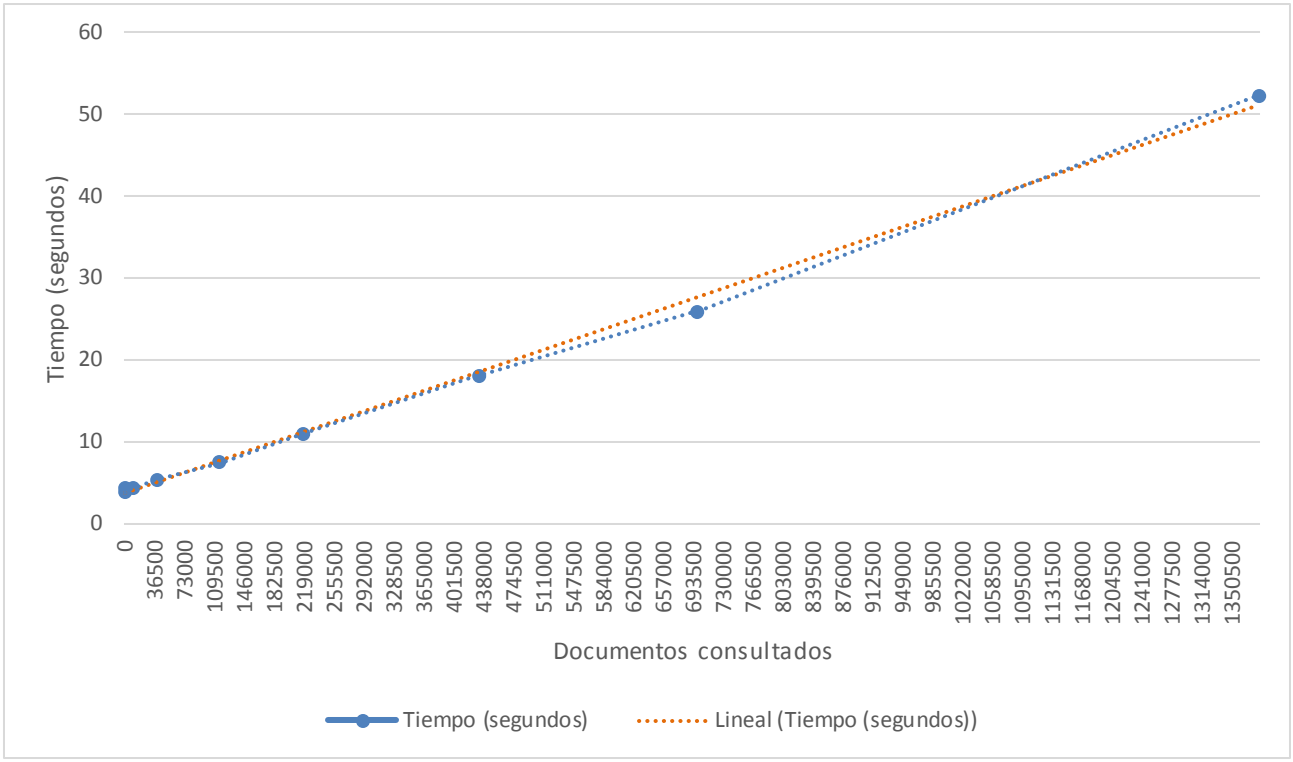
Omitiendo mejor y peor resultando y realizando la media: **198,95 segundos**

6.2 RESULTADOS PROCESADOS

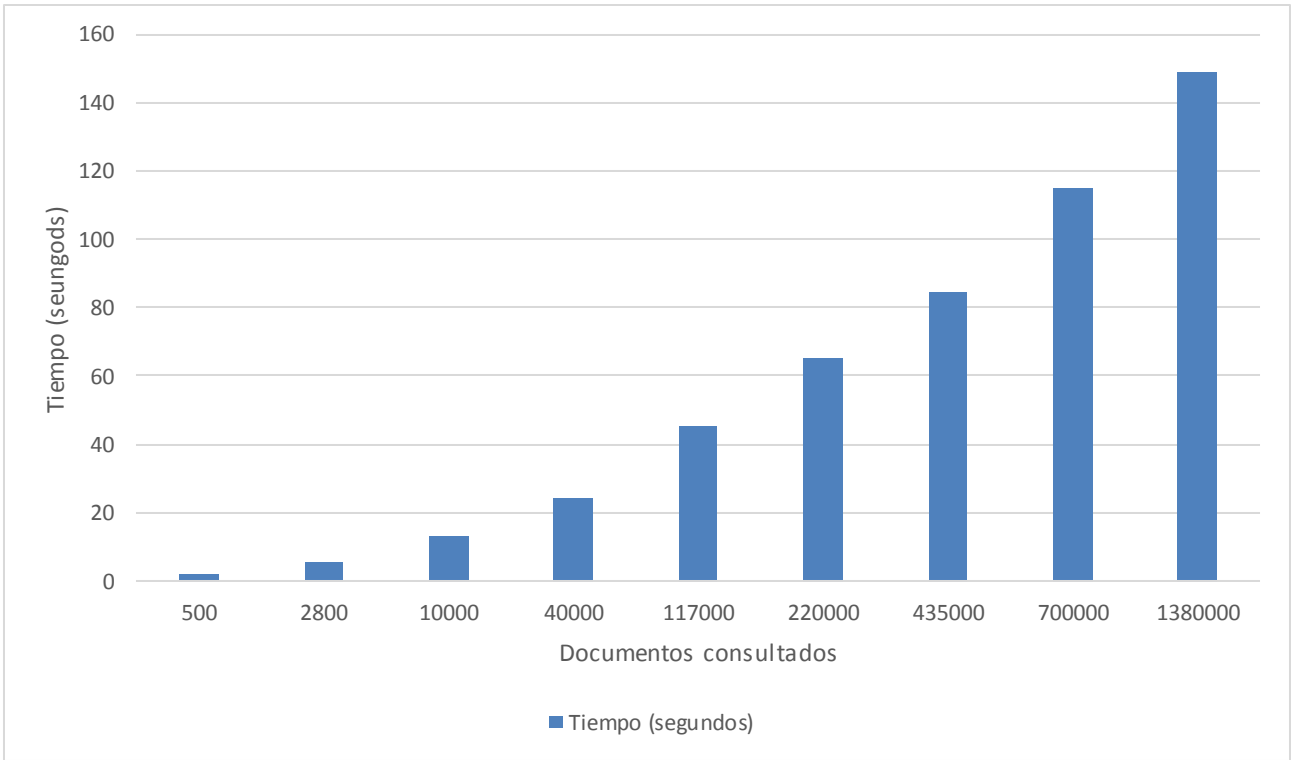
A continuación se muestran los datos procesados de a partir de la toma de datos realizada para las diferentes técnicas de indexación objeto de estudio.

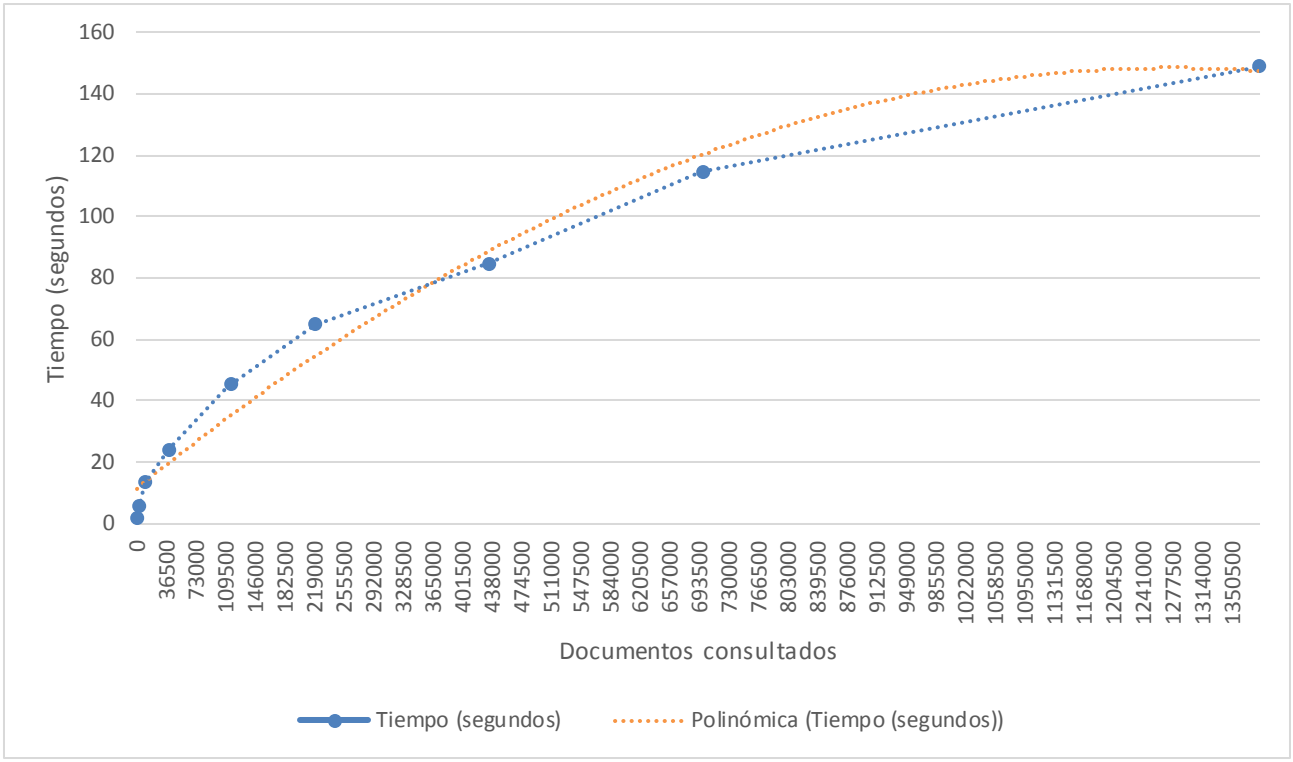
6.2.1 TÉCNICA INDEXACION I: ÍNDICE TEMPORAL SIMPLE + ÍNDICE LONGITUD SIMPLE + ÍNDICE LATITUD SIMPLE



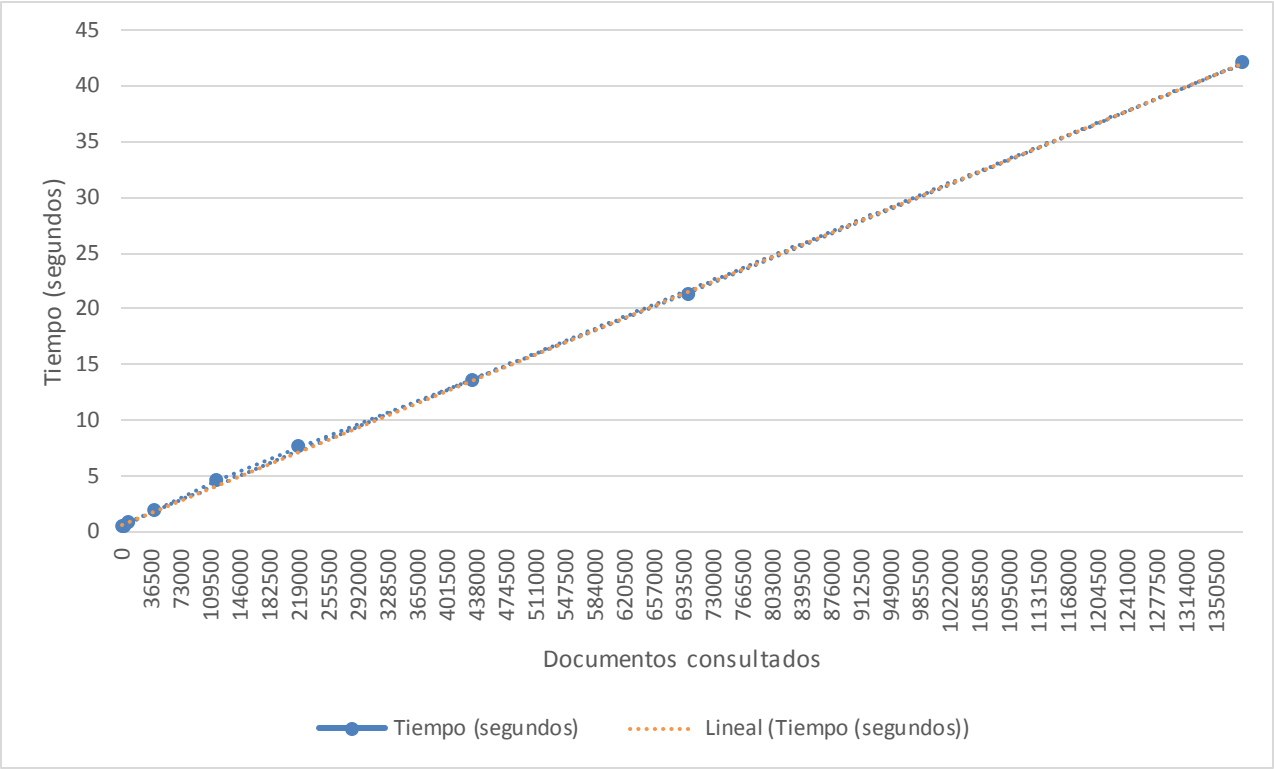


6.2.2 TÉCNICA INDEXACION II: ÍNDICE GEOESPACIAL (LONGITUD, LATITUD) + ÍNDICE SIMPLE TEMPORAL

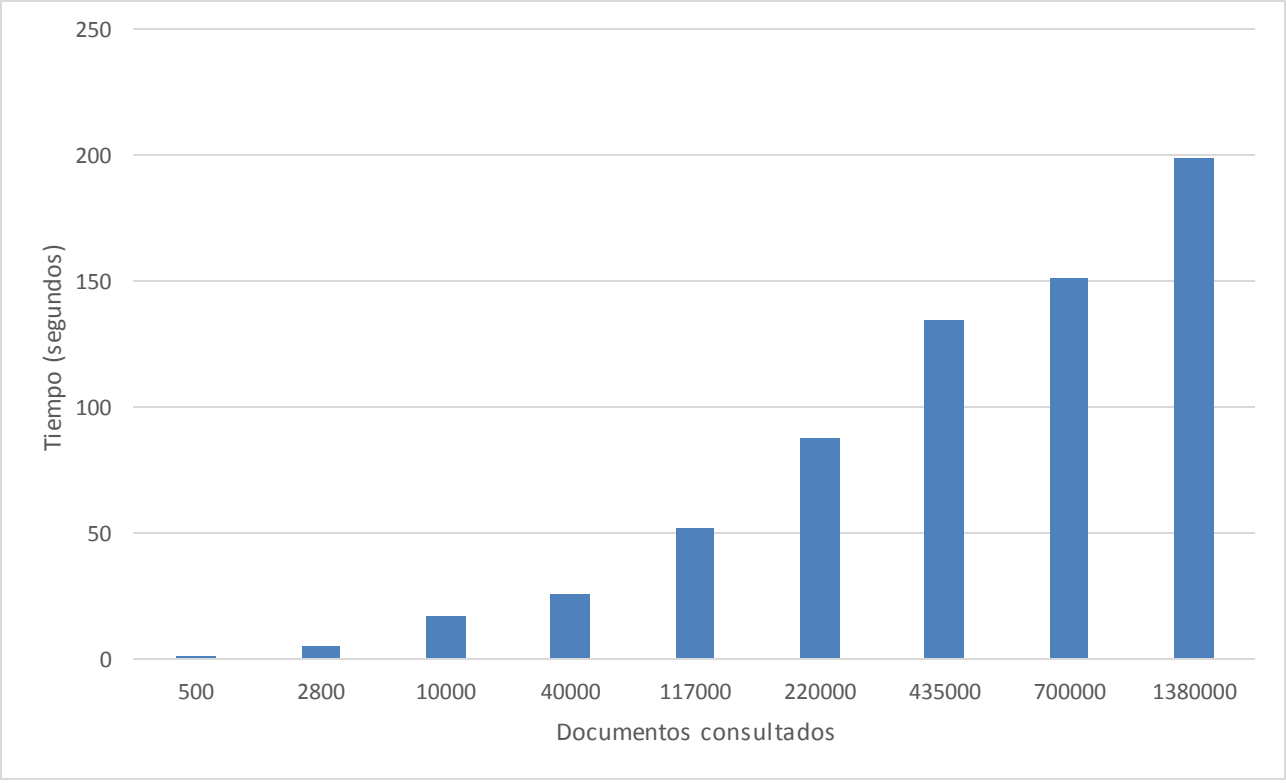


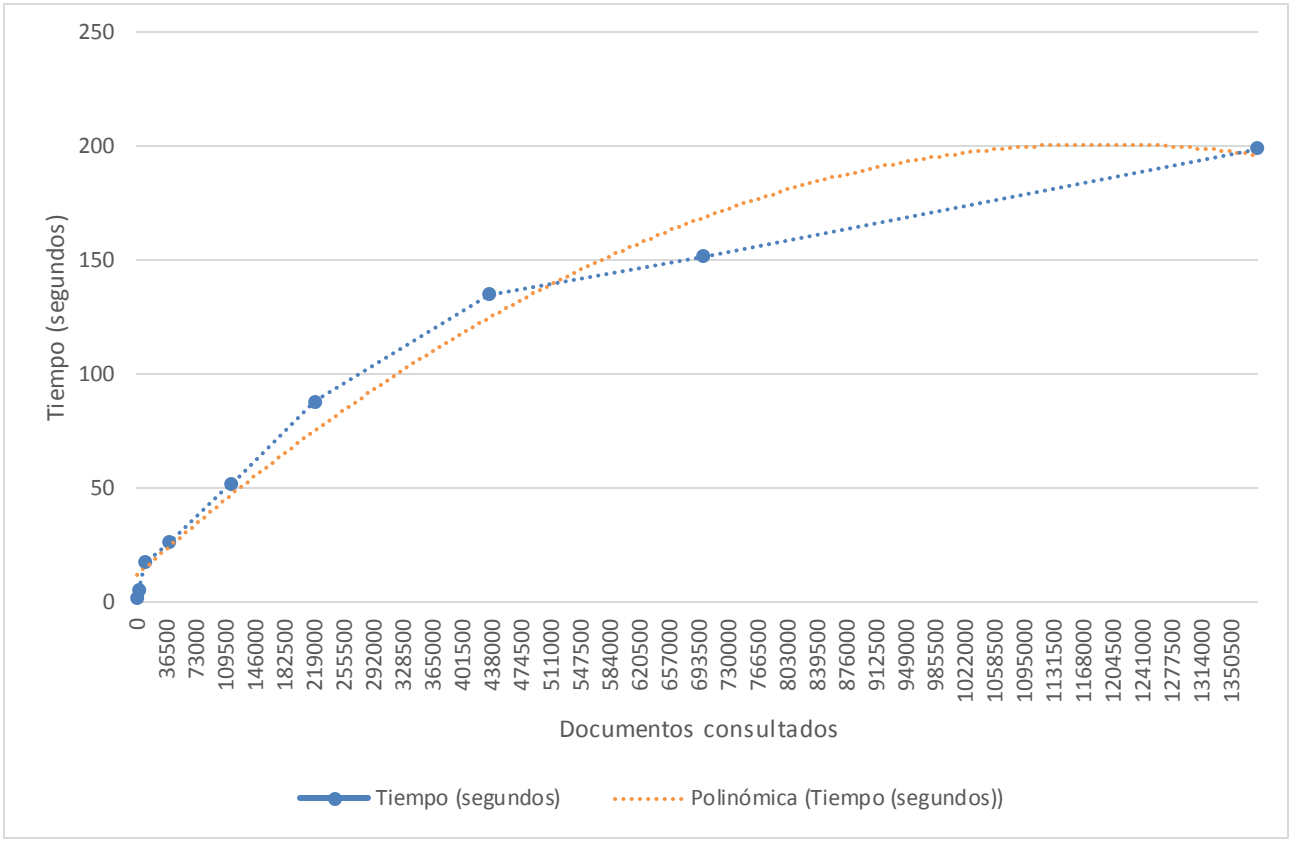


6.2.3 TÉCNICA INDEXACION III: ÍNDICE COMPUESTO (TEMPORAL, LONGITUD, LATITUD)

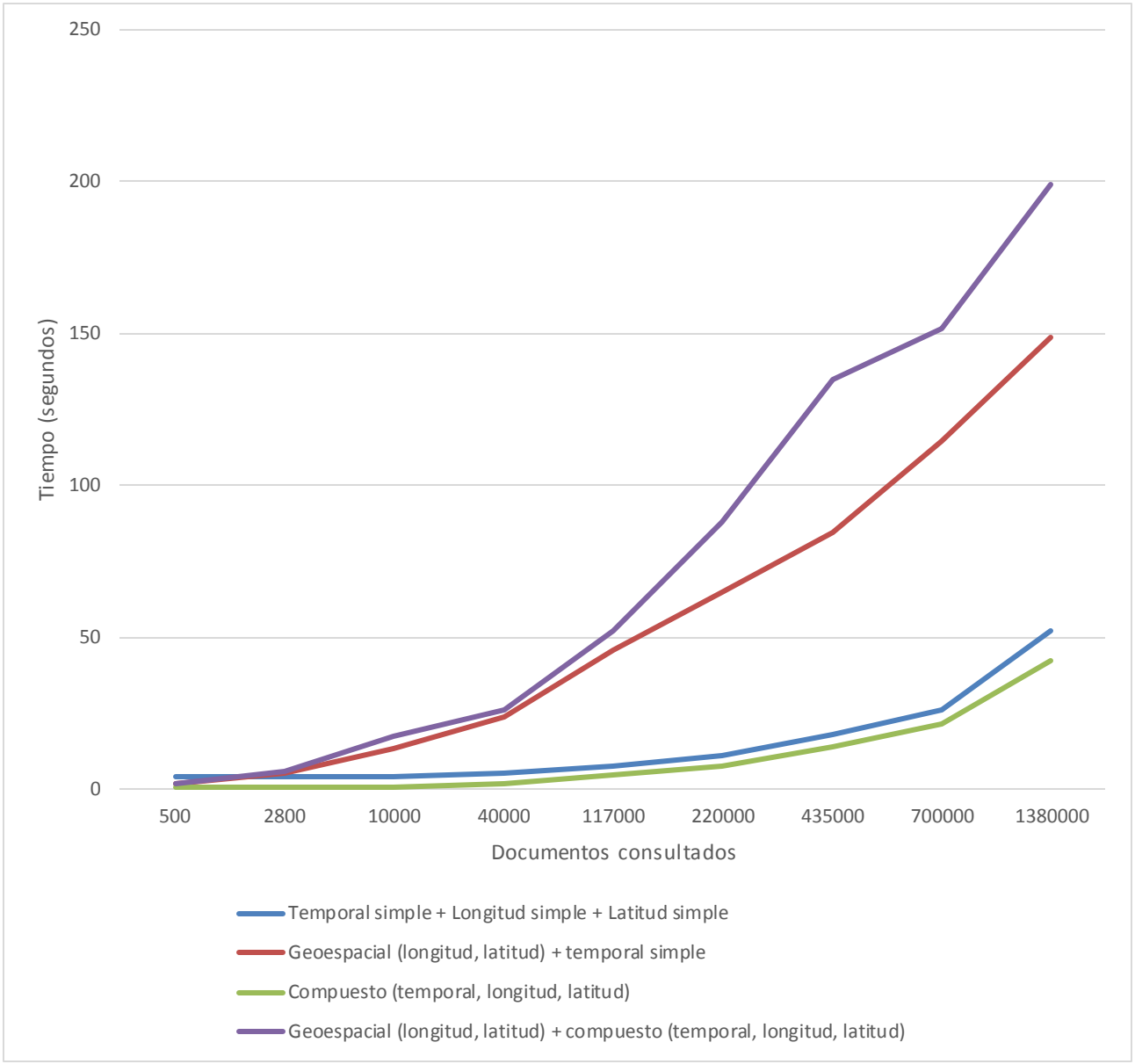


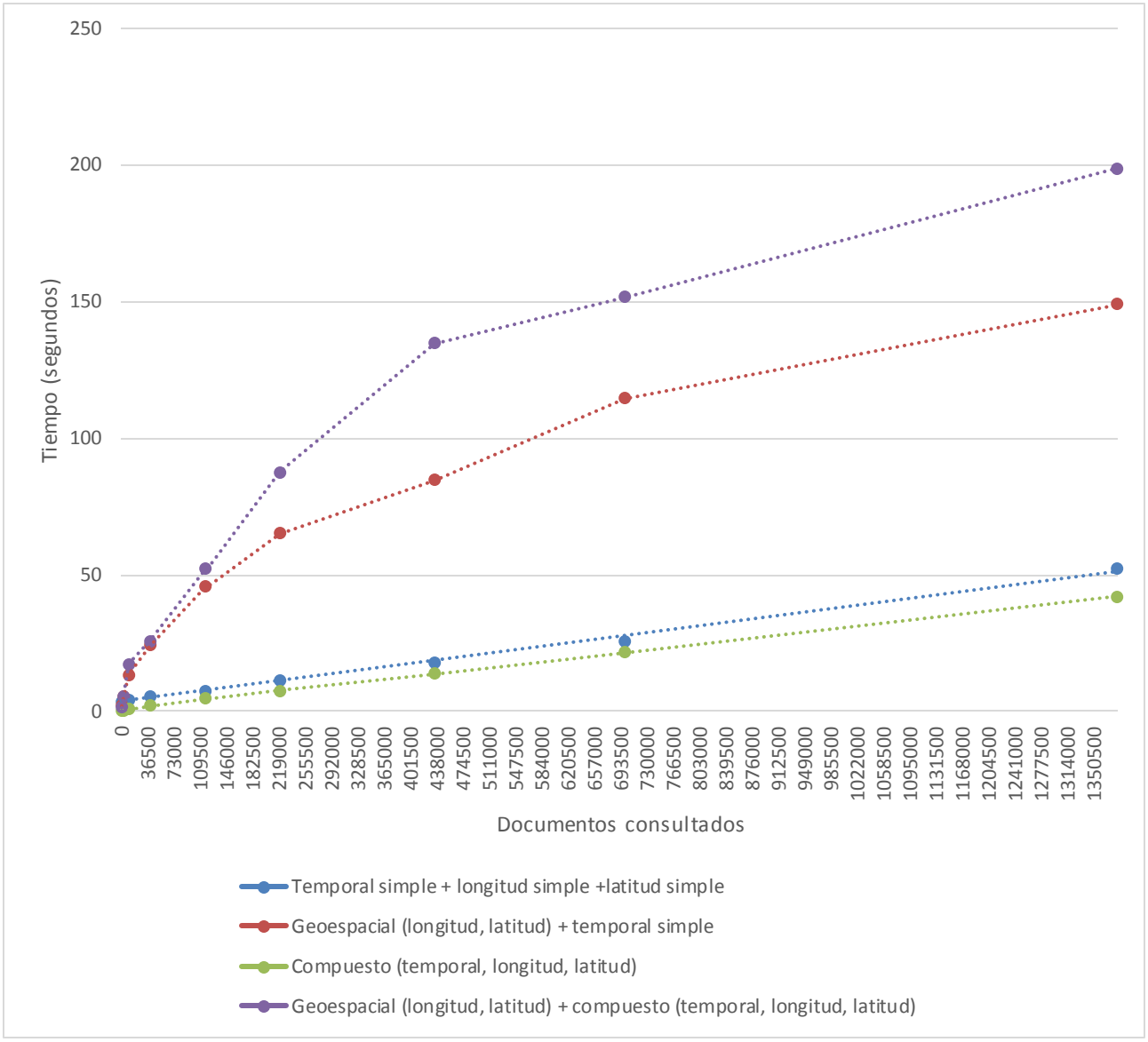
6.2.4 TÉCNICA INDEXACION IV: ÍNDICE GEOESPACIAL (LONGITUD, LATITUD) + ÍNDICE COMPUESTO (TEMPORAL, LONGITUD, LATITUD)




250
200
150
100
50
0
Tiempo (segundos)
0
36500
73000
109500
146000
182500
219000
255500
292000
328500
365000
401500
438000
474500
511000
547500
584000
620500
657000
693500
730000
766500
803000
839500
876000
912500
949000
985500
1022000
1058500
1095000
1131500
1168000
1204500
1241000
1277500
1314000
1350500
Documentos consultados
Tiempo (segundos)
Polinómica (Tiempo (segundos))

6.2.5 TÉCNICAS DE INDEXACIÓN: COMPARATIVA





6.3 DISCUSIÓN DE RESULTADOS

En este apartado se realizará la discusión de la toma de datos descrita en este documento en el apartado 6.1 TOMA DE DATOS y de su posterior procesamiento, mostrado en el apartado 6.2 RESULTADOS PROCESADOS.

Se han estudiado cuatro técnicas de indexación de datos geoespaciales.

- Estrategia 1. Se declara un índice simple por cada elemento que conforman los tres elementos de consultas (tiempo, longitud y latitud).
    - Índice temporal simple + índice longitud simple + índice latitud simple.

- Estrategia 2. Se declara un índice geoespacial 2D para indexar la longitud y latitud y un índice simple para el tiempo.
    - Índice geoespacial (longitud, latitud) + índice temporal simple.
- Estrategia 3. Se declara un índice compuesto para los tres elementos de consultas (tiempo, longitud y latitud).
    - Índice compuesto (tiempo, longitud, latitud).
- Estrategia 4. Se declara un índice geoespacial 2D para indexar la localización y un índice compuesto conformado por el tiempo y la localización (longitud, latitud)
    - Índice geoespacial (longitud, latitud) + índice compuesto (tiempo, longitud, latitud).

A continuación se procede a analizar cada estrategia de indexación individualmente.

La estrategia 1 se trata de una técnica sencilla, la declaración de un índice por cada elemento que conforma los atributos de consulta. Con esta técnica se realiza la construcción de tres estructuras de árbol B, una por cada clave: longitud, latitud y tiempo. Presenta una tendencia claramente lineal, dada por la fórmula:

$$O(n)= 0{,}0172n + 3{,}6243$$

n indica el número de documentos consultados

O es el tiempo estimado de consulta en segundos

A partir de dicha fórmula se extrae un tiempo de consulta por cada documento de 0,0172 segundos aproximadamente.

Como ventaja presenta que es fácil e intuitiva y ofrece unos tiempos de consulta realmente buenos. Como inconveniente indicar su tendencia lineal.

La estrategia 2 se trata de otra de las técnicas más intuitivas, utilizar el índice geoespacial que proporciona MongoDB para indexar el espacio (longitud y latitud) y utilizar un índice simple para indexar el tiempo. Con esta esta técnica se utiliza internamente una estructura de Árbol R para el espacio y una estructura de Árbol B para el tiempo. A priori, parece la mejor opción como técnica de indexación, sin embargo, no ofrece buenos

tiempos de cómputo, ofrece una tendencia difícil de predecir, pero con tendencia exponencial-cuadráticas dada por la fórmula:

$$O(n)=2^{0.5n^2} + 0,1072n + 11,378$$

n indica el número de documentos consultados

O es el tiempo estimado de consulta en segundos

Se trata de una técnica de indexación bastante pésima, ofrece un tiempo de cómputo muy elevado y aunque ofrece una moderación de su tendencia en consultas de documentos de volúmenes elevados no resulta muy ventajosa.

La estrategia 3 se trata de una técnica no tan intuitiva como las anteriores, consiste en aprovechar una técnica de indexación propia de MongoDB. Se basa en la indexación de los tres elementos de búsqueda, tiempo, longitud y latitud a través de un índice compuesto. Esta técnica construye una única estructura de Árbol B para las tres claves. En función del orden de declaración de las claves en el índice se puede conformar una estructura muy diferente de Árbol B (tiempo>>longitud>>latitud o latitud>>longitud>>tiempo) proporcionando tiempos de consulta mejores o peores. En este caso se ha utilizado el orden: tiempo, longitud y latitud; ya que declarando aquellas claves más restrictivas primero, se consiguen mejores tiempos de consulta. En este caso el tiempo es más restrictivo que el espacio y la longitud, a su vez, más restrictiva que la latitud. La tendencia de esta técnica es claramente lineal, dada por la fórmula:

$$O(n)=0,0149x + 0,6764$$

n indica el número de documentos consultados

O es el tiempo estimado de consulta en segundos

Se trata sin duda alguna de la mejor técnica que proporciona MongoDB. A partir de dicha fórmula se extrae un tiempo de consulta por cada documento de 0,0149 segundos aproximadamente. Como ventaja sin duda alguna destaca su eficiencia, como

inconvenientes su carácter lineal y la naturaleza del propio índice, que dota de mayor relevancia a unas claves frente a otras.

La estrategia 4 se trata de la estrategia más audaz, pretende por un lado aprovechar las ventajas del índice geoespacial para la indexación de la localización (longitud y latitud) y por otro lado, utilizar un índice compuesto integrado por la localización de forma que se sirva del índice geoespacial como auxiliar. Internamente construye un Árbol R para la localización y un Árbol B compuesto cuya estructura depende parcialmente del Árbol R utilizado para la localización. Se basa en la idea de que MongoDB permite la integración de múltiples índices. Basándonos en los resultados, la técnica ofrece una tendencia exponencial

$$O(n)=3\text{^}05n^2 + 0,1588n + 11,63$$

n indica el número de documentos consultados

O es el tiempo estimado de consulta en segundos

Es sin duda alguna la peor técnica, ya que en principio MongoDB no parece ser capaz de aunar las estructuras de datos: Ofreciendo el peor rendimiento de todas las técnicas.

Sin duda alguna, la estrategia 3 se trata de la mejor técnica de indexación. La utilización de índices compuestos, debidamente diseñados, ofrece comparativamente mejores rendimientos que técnicas en principio adaptadas a un uso geoespacial como los índices 2D.

A su vez, técnicas de indexación sencillas como la estrategia 2, que utiliza índices simples, ofrece un rendimiento excelentes muy cercanos al uso de índice compuestos, como la estrategia 3.

Queda de manifiesto que la utilización de índices geoespaciales nativos de MongoDB no ofrecen el rendimiento esperado en datos geoespaciales de tres dimensiones (longitud, latitud y tiempo), ni si quiera en conjunto con otros índices: ni simples, ni compuestos.

El motivo parece deberse a la imposibilidad actual de MongoDB de indexar el tiempo en una estructura única con el espacio.

7 CONCLUSIONES

7.1 CONCLUSIONES DEL TRABAJO DE FIN DE GRADO

Las conclusiones extraídas de este proyecto elaborado como trabajo de fin de grado son diversas, en primer lugar, partiendo de una idea se ha conseguido elaborar una aplicación Web funcional, esto no tendría mayor importancia si no se hubiera conseguido aunar en un solo proyecto tecnologías tan relevantes.

Un aspecto importante a recalcar durante la elaboración de este proyecto, ha sido dotar de la importancia que se merece a la fase de análisis, muchas veces olvidadas en los proyectos. Para este proyecto ha sido primordial investigar sobre las tecnologías, comparar distintos productos de una misma tecnología, ver el alcance que pueden tener y las limitaciones, en definitiva realizar un estudio de viabilidad.

Realizando un estudio de viabilidad exhaustivo se ha conseguido obtener una aplicación Web totalmente operativa, con una alta escalabilidad y con capacidad de incorporar otras tecnologías fácilmente, debido en gran parte a la utilización del *framework* Spring.

El gran poder computacional de Spring ha proporcionado el desarrollo de una aplicación Web flexible y escalable. La existencia de módulos de Spring que amplían sus funcionalidades se trata de un elemento clave. Spring además ha permitido implementar un diseño en la aplicación Web claro, separando la lógica de negocio de las vistas debido a la implementación del patrón Modelo Vista Controlador.

Sin duda alguna Spring posee una ventaja competitiva frente a otros entornos de desarrollos Web y es que permite una complementación muy buena con otras tecnologías, razón por la cual se ha optado para la elaboración de este proyecto, desechando otros *frameworks* de características similares en la fase de viabilidad como Struts.

Sin duda alguna, Spring se trata de un referente en la construcción de aplicaciones Web y la tendencia es claramente ascendente en su uso. Actualmente existe un auge de tecnológicas como AngularJS, Django, NodeJS, etc. que están pegando fuerte en el mercado y que en su mayoría usan JavaScript como lenguaje de programación. Sin embargo pienso y mediante la elaboración de este proyecto he certificado, que para la realización de aplicación Web robustas, y con cierta complejidad en su lógica de negocio, es necesario utilizar un *framework* como Spring que te permita desarrollar lógicas de

negocios complejas a través de un lenguaje de programación utilizado a nivel mundial como Java.

Por otro lado existe también un auge creciente en la utilización de bases de datos NoSQL. Este tipo de bases de datos y la gran cantidad de tipos existentes, permiten elegir la base de datos a utilizar para coincidir con el modelo de datos con el cual queremos trabajar, de forma contraria hasta como sucedía hasta ahora, que se debía adaptar el modelo de datos a la base de datos tradicional SQL.

Utilizar una base de datos como MongoDB, dejando a un lado el carácter didáctico, aporta una gran personalidad a la aplicación Web desarrollada como proyecto. Ofreciendo las ventajas generales de contar con una base de datos NoSQL, entre las que destacan principalmente la eficiencia y la alta escalabilidad y por otro lado las ventajas de MongoDB, la posibilidad de almacenar datos como documentos semiestructurados y utilizar índices a fin de agilizar las consultas de los datos.

Sin duda alguna de MongoDB extraería que se trata de una base de datos NoSQL de carácter on-line, muy útil para realizar escrituras y especialmente lecturas en tiempo real. Ofrece un alto nivel de respuesta ante altos niveles de concurrencia y un bajo nivel transaccional. Su utilización no es muy apropiada para data *warehousing*, a pesar de la posibilidad de almacenar datos semiestructurados, ni para el almacenamiento de datos cargados en lotes de forma offline. El uso más apropiado es el almacenamiento de datos operacionales de la infraestructura de un sitio Web. Muy apropiada para datos dinámicos y cambiantes y para realizar lecturas rápidas gracias a sus técnicas de indexación.

Respecto a las técnicas de indexación utilizadas y analizadas en este proyecto, ha quedado patente que en ocasiones la utilización de técnicas generales no ofrecen los mejores rendimientos en todas las plataformas. La utilización de técnicas específicas correspondientes al sistema que se utiliza y en concreto en este caso a MongoDB ha proporcionado los mejores resultados. Es siempre importante entender el sistema que se utiliza y todos los recursos que ofrece, en ocasiones un estudio exhaustivo puede arrojar información que de otra manera seria imposible de conocer.

7.2 CONCLUSIONES PERSONALES

A nivel personal ha sido toda una satisfacción el ver cómo, día a día, lo que al principio era una idea audaz se iba transformando mediante el trabajo continuado en algo "tangible" y real. Y más aún cuando el proyecto se trataba únicamente de una idea, y por qué no decirlo, de cierta complejidad, que pretendía integrar diversidad de tecnologías, patrones y modelos haciendo uso de datos provenientes de un campo científico que no se suele dominar, como son los datos provenientes de los satélites meteorológicos.

A medida que avanzaba el proyecto, como si de una estructura Lego se tratara, se añadían más piezas al proyecto: una conexión a una base de datos NoSQL, la inclusión de una nueva librería, la integración de un mapa gráfico mediante OpenLayers... Conformando así, un proyecto cada vez más robusto, con mayores funcionalidades, integrando tecnologías muy diversas sobre el marco Web y con un alto poder computacional,

Resultaba indispensable comprender cada elemento nuevo que formaba parte del proyecto, en muchos casos tecnologías no muy utilizadas, de las cuales apenas existe documentación: Spring Data e índices geoespaciales, técnicas de indexación para datos geoespaciales y OpenLayers 3, entre otras.

Para comprender cada elemento que conforma este proyecto hacía falta empaparse de documentaciones bibliográficas, la lectura de manuales y en muchos casos, la mera investigación.

El carácter investigador es sin duda una de las piezas claves de este proyecto. Aunque el estudio se ha realizado en MongoDB, creo que sin duda puede ser extrapolables a otras bases de datos. Creo además, que a través de esta investigación se vislumbra una nueva forma de entender los datos y su indexación, entendiendo nuevos tipos de índices como los índices compuestos.

Sin duda alguna la dificultad de este trabajo era aunar en un único proyecto tanta diversidad de tecnologías, en muchos casos no muy utilizadas, y haber obtenido el máximo provecho mediante su integración.

El contexto del proyecto resulta idóneo, actualmente con el avance de Internet y dispositivos que no dejan de incrementar su poder computacional; cada vez son más los

portales Web que ofrecen contenidos interactivos fácilmente ejecutables por cualquier dispositivo y navegador. Haber podido aprender y ampliar mis conocimientos en desarrollo de Aplicaciones Web pienso que ha sido de enorme beneficio. Además, haber podido utilizar *frameworks*, patrones y modelos de trabajo tan diversos ha aumentado enormemente mis horizontes en desarrollos Web, especialmente a través de Spring Framework.

Del mismo modo, haber podido estudiar una base de datos NoSQL como MongoDB lo considero muy beneficioso. Actualmente El movimiento NoSQL está causando mucho impacto en los medios y la industria, y muchos responsables de sistemas están comenzando a interesarse por este tipo de alternativas NoSQL, que se presentan como un concepto distinto al que se manejaba hasta ahora en las bases de datos relacionales. Creo que sin duda en los próximos años, bases de datos como MongoDB se dispararán en el mercado sustituyendo a multitud de bases de datos tradicionales.

Para finalizar, diré que se trata de un proyecto con un gran potencial, con innumerables posibilidades de ampliación. Pero sin duda alguna, se trata de una realidad funcional y operativa, la cual simplemente basta ser desplegada para ser utilizada.

7.3 TRABAJOS FUTUROS

A continuación se plantean una serie de funcionalidades para ser añadidas:

- Incorporación de una página de administración, para inserción de nuevos datos geoespaciales y administración de los actuales.
- Permitir en la página de administración, mediante Spring Security, la existencia de un usuario administrador único y con capacidad para llevar cabo las gestiones de administración.
- Capacidad de selección entre datos geoespaciales del satélite QuikSCAT y RapidSCAT mediante la integración de sus datos.
- Optimización de las técnicas de representación gráfica de los datos geoespaciales.

8 BIBLIOGRAFÍA Y REFERENCIAS

Seguidamente vienen detalladas las bibliográficas y referencias utilizadas para la elaboración de este proyecto:

1. Ruiz F, Rodríguez F (director). *Indexación espacio-temporal de datos del satélite QuikSCAT mediante Grid-File*. Proyecto Fin de Carrera. Universidad de Extremadura, 2010.

2. Rodríguez FR, Barrena M. "*Spatio-temporal indexing of the QuikSCAT wind data*". En: IGARSS'09, Proceedings of the 2009 IEEE International Geoscience and Remote Sensing Symposium. IEEE Computer Society. Ciudad del Cabo, Sudáfrica. 2009. Vol II: 503-506

3. Rodríguez F. *Algoritmos de indexación y carga masiva de datos geoespaciales*. Tesis doctoral. Director: Barrena M. Trabajo de fin de Máster. Escuela Politécnica, Universidad de Extremadura, 2011: 277

4. Ruiz F, Rodríguez F, Plaza A. *Improving the processing of data obtained from meteorological satellites using graphics processing units (GPUs)*. Proceedings of SPIE, Satellite Data Compression, Communication and Processing VIII, SPIE Symposium on Optical Engineering and Applications. San Diego, CA. 2012.

5. Ruiz F, Rodríguez F (director). *Almacenamiento, Recuperación y Carga Masiva de Datos Topográficos Terrestres SRTM en Grid-File*. Proyecto Fin de Carrera. Universidad de Extremadura, 2007.

6. Winds Measuring Ocean Wind from Space (Winds). "SeaWinds on QuikSCAT". 1999. Visitado en 2015. Disponible en: http://winds.jpl.nasa.gov/index.cfm

7. Sun Microsystems. JAVA™ Software Development Kit. Visitado en 2015. Disponible en: http://java.sun.com

8. Eclipse IDE, Plataforma de software de código abierto. Visitado en enero, 2015. https://eclipse.org/

9. OpensLayers 3, paquetes de librerías para el desarrollo de mapas de alto rendimiento. Visitado enero 2015. Disponible en: http://openlayers.org/

10. Spring Tool Suite, Plataforma de software ready-to-use para la integración de frameworks y soporte tecnológico. Visitado en 2015. Disponible en: https://spring.io/tools

11. MongoDB, Base de datos de código libre NoSQL orientada a documentos. Visitado en 2015. Disponible en: https://www.mongodb.org/

12. Physical Oceanography DAAC (PO.DAAC)."SeaWinds on QuikSCAT". Visitado en 2015. Disponible en: http://podaac.jpl.nasa.gov/OceanWind/QuikSCAT

13. Hoffman RN, Leidner SM. *An Introduction to the Near-Real-Time QuikSCAT Data*. Weather and Forecasting Journal, 2005. 20(4): 476-493.

14. Muñoz J, Felicísimo AM, Cabezas F, et al. *Wind as a Long-Distance Dispersal Vehicle in the Southern Hemisphere*. Science, 2004. 304(5674):1144-1147

15. Felicísimo AM, Muñoz J, González-Solís J. *Ocean Surface Winds Drive Dynamics of Transoceanic Aerial Movements*. Plos ONE, 2008. 3(8): 2928

16. "Quick Scatterometer". "Missions – SeaWinds on QuikSCAT". Visitado en 2015. Disponible en: http://winds.jpl.nasa.gov/missions/quikscat/index.cfm

17. "NASA Sees Rapid Changes in Arctic Sea Ice" (Winds). Visitado en 2015. Disponible en: http://winds.jpl.nasa.gov/publications/arcticSeaIce.cfm

18. National Aeronautics and Space Administration (NASA). "Ocean Wind Power Maps Reveal Possible Wind Energy Sources". Visitado en 2015. Disponible en: http://www.nasa.gov/topics/earth/features/quikscat-20080709.html

19. "NASA Satellite Monitors Post-Hurricane Gulf Coast Flood Potential" (Winds). Visitado en 2015. Disponible en: http://winds.jpl.nasa.gov/publications/PIA03029.cfm

20. "Hierarchical Data Format". Visitado en 2015. Disponible en: http://www.hdfgroup.org/products/hdf4/

21. Gaede V, Gunther O. *Multidimensional Access Methods*. ACM Computing Surveys, 1998. 30(2): 170-231.

22. Rubiales M. *HTML5, CSS3 y JavaScript*. 2013.

23. Gauchat, JD. *El gran libro de HTML5, CSS3 y Javascript*. Barcelona, 2012. Disponible en: https://adegiusti.files.wordpress.com/2013/09/el-gran-libro-de-html5-css3-y-javascript.pdf

24. Johnson R, Hoeller J, Arendsen A et al. *Professional Java Development with the Spring Framework*. USA, 2005. 23-539. Disponible en: https://books.google.es/books?id=lt_nh9uJEGgC&printsec=frontcover&dq=spring+framework&hl=es&sa=X&ei=Ie-bVbCcDcblUYzhgLgL&ved=0CDEQ6AEwAA#v=onepage&q=spring%20framework&f=false

25. Van de Velde T, Snyder B, Dupuis C et al. *Beginning Spring Framework 2*. Wiley, 2008

26. Bohm C, Berchtold S, Keim DA. *Searching in High-Dimensional Spaces: Index Structures for Improving the Performance of Multimedia Databases*. ACM Computing Surveys, 2001. 33(3): 322-373.

27. Mokbel MF, Ghanem TM, Aref WG. *Spatio-Temporal Access Methods*. IEEE Data Engineering Bulletin, 2003. 26(2): 40-49.

28. Nguyen-Dinh LV, Aref WG, Mokbel MF. "*Spatio-temporal Access Methods: Part2 (2003 {2010)*". IEEE Data Engineering Bulletin, 2010. 33(2): 46-55.

29. NASA PO.DAAC, 2009. Phisical Oceanography. Distributed Active Archive Center. Sea-Winds on QuikSCAT. Visitado en 2015. Disponible en: ftp://podaac-opendap.jpl.nasa.gov/opendap/allData/quikscat/

30. NASA PO.DAAC, 2010. Phisical Oceanography. Distributed Active Archive Center. Ocean Winds. Visitado en 2015. Disponible en: http://podaac.jpl.nasa.gov/

31. Chodorow K. *MongoDB: The Definitive Guide: Powerful and Scalable Data Storage*. Reilly O. 2013

32. Naderi F, Freilich MH, Long DG. *Spaceborne Radar Measurement of Wind Velocity Over the Ocean. An Overview of the NSCAT Scatterometer System*. Proceedings of the IEEE. 1991. 79(6).

33. *Artist's concept of QuikScat* (Winds). Visitado en 2015. Disponible en: http://winds.jpl.nasa.gov/missions/quikscat/seawinds_img.cfm

34. *Hierarchical Data Format*. Visitado en enero 2015. Disponible en: http://www.hdfgroup.org/

35. Fan Z, Qiu F, Kaufman A, Yoakum-Stover S. *GPU Cluster for High Performance Computing*. Center For Visual Computing and Department of Computer Science (Stony Brook University). 2004.

36. Leutenegger S, Nicol D. *Efficient Bulk-Loading of GridFiles*. Institute for Computer Applications in Science and Engineering. 1994. 9: 410-420. Disponible en: http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19950007476.pdf

37. *Java bindings for OpenCL*. Versión 0.1.7. Visitado en 2015. Disponible en: http://www.jocl.org/

38. Microsoft Inc. MS Windows 7 Professional Edition, Microsoft Inc. Visitado en 2015. Disponible en: http://www.microsoft.com

39. Wikipedia. Enciclopedia libre. Artículo "Eclipse". Visitado en 2015. Disponible en: http://es.wikipedia.org/wiki/Eclipse

40. Wikipedia. Enciclopedia libre. Artículo "MongoDB". Visitado en 2015. Disponible en: http://es.wikipedia.org/wiki/MongoDB

41. Muthukrishnan S, Poosala V, Suel T. *On rectangular partitionings in two dimensions: Algorithms, complexity, and applications*. ICDT'99, Proceedings of the 7th International Conference on Database Theory. Jerusalem, Israel. 1999. 1540: 236-256.

42. Muthukrishnan S, Suel T. *Approximation Algorithms for Array Partitioning Problems*. Journal of Algorithms. 2004. 54(1):85–104.

43. Berman P, DasGupta B, Muthukrishnan S. *Approximation Algorithms for MAX-MIN Tiling*. Journal of Algorithms. 2003. 47(2):122-134.

44. Shell D. *A High-Speed Sorting Procedure*. General Electric, Cincinnati, Ohio. 1959.

45. Guide to IDL Programming, Coyote S. Visitado en mayo 2015. Disponible en: http://www.idlcoyote.com/map_tips/lonconvert.html

46. Sedgewick R. *A New Upper Bound for Shellsort*. Journal of Algorithms, 1986. 159-173.

47. Star, Center for Satellite Application and Research. Visitado en 2015. Disponible en: http://manati.orbit.nesdis.noaa.gov/

9 ANEXO 1. MANUAL DE USUARIO

El uso de la aplicación es muy intuitivo y descriptivo, desarrollando para tal fin una gran cantidad de recursos gráficos para mejorar la usabilidad. Para hacer uso de la aplicación necesitamos tener instalado en nuestro equipo cualquier tipo de navegador Web actual. A continuación se declaran algunos navegadores disponibles sobre los que se confirma su uso y compatibilidad:

Ilustración 25: Navegadores más comunes

Sobre los navegadores descritos, estando previamente instalados, procedemos a la ejecución. En mi caso, para elaborar este manual utilizaré el referente entre los navegadores: Google Crome. Procedemos a la ejecución de una ventana o pestaña nueva:

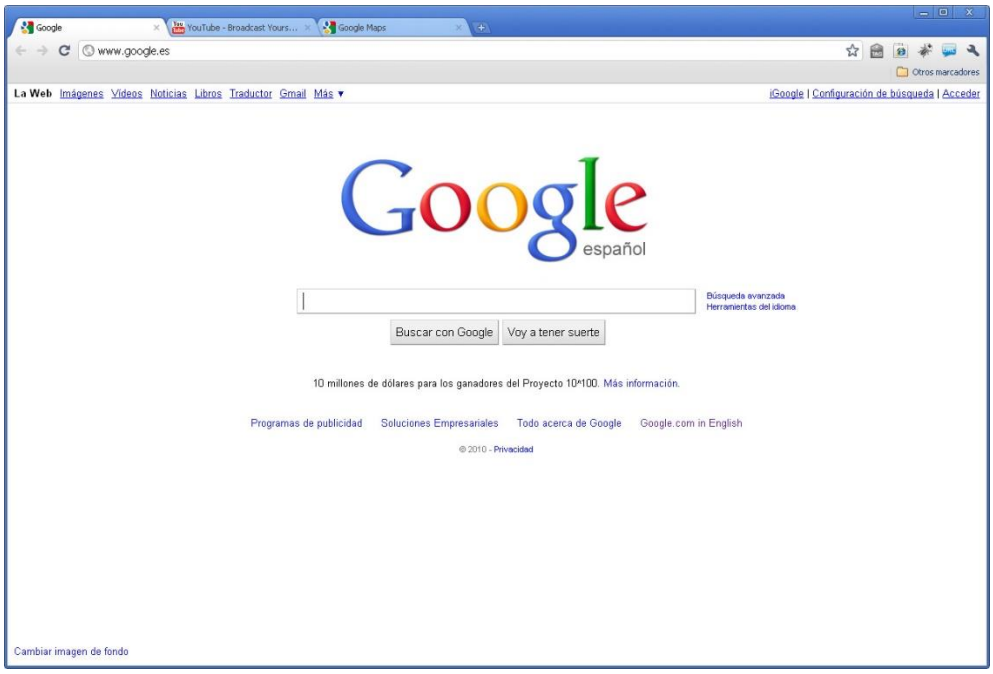


Ilustración 26: Ventana principal de navegación de Google Chrome

En primer lugar debemos acceder a la página principal de la aplicación. Para ello y como acceso para cualquier otra página o portal Web, declaramos la dirección URL en la barra de direcciones y escribiremos donde se encuentra alojada la aplicación.

Ilustración 27. Barra de direcciones

Al estar la aplicación alojada en nuestro dispositivo local de manera provisional y lanzada a través de Apache Tomcat nos remitimos a "http://localhost:8080/HelloSpringWithMongoDB/".

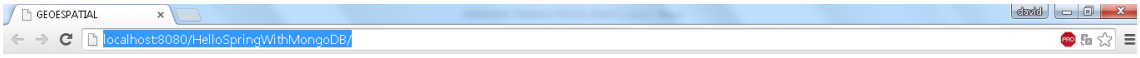


Ilustración 28: Barra de direcciones con declaración de ruta

Finalmente se procederá a mostrar la página principal de la aplicación web desarrollada para este proyecto:

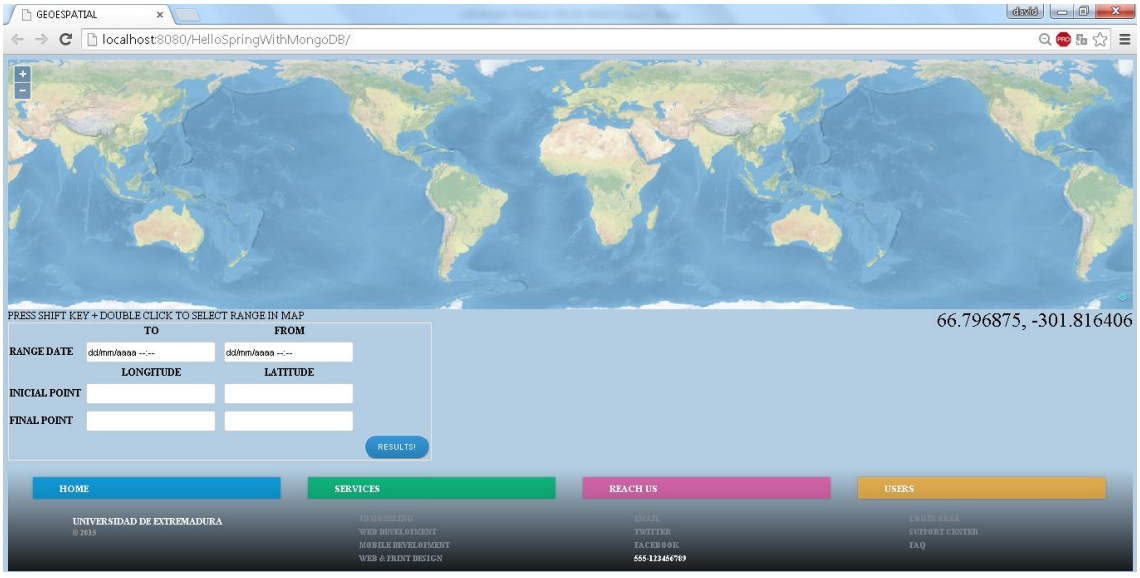


Ilustración 29: Página principal de la aplicación web del proyecto visualizada desde Google Chrome

A partir de dicha ventana podemos interactuar con la aplicación, procediendo a realizar búsquedas de datos geoespaciales por rango marítimo y fecha. Para establecer un rango marítimo se puede realizar de dos formas:

De manera textual, a través del cuadro de mandos desarrollado para tal fin en la ventana principal, donde se puede declarar la longitud y latitud del punto inicial y la longitud y latitud del punto final de manera que dichos puntos formen un área:

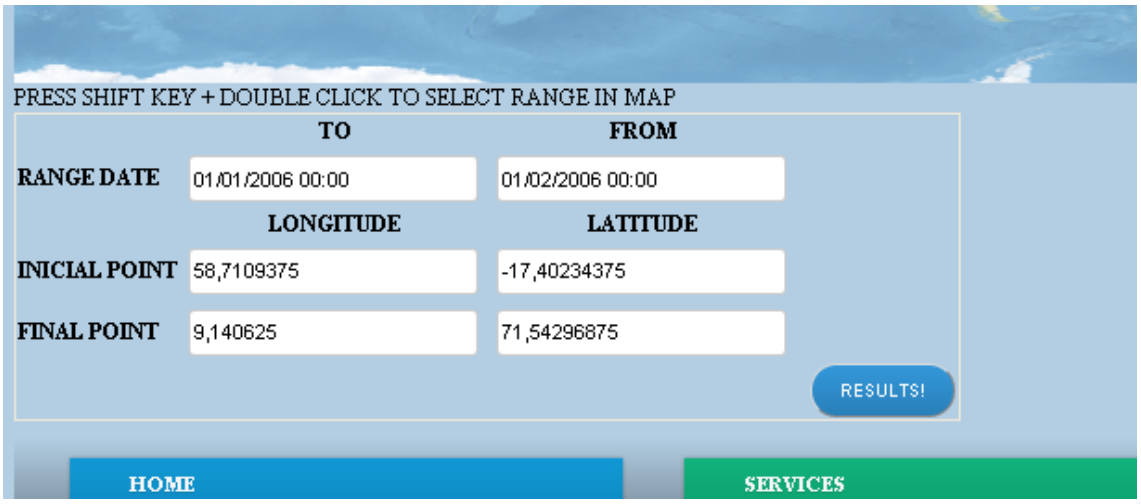


Ilustración 30: Cuadro de mando de la página principal de la aplicación web

O bien, de manera gráfica, a través del mapa implementado con OpenLayers3, para ello pulsando la tecla SHIFT de nuestro teclado y haciendo doble clic en el mapa podemos seleccionar una región del mapa. De manera auxiliar, se establecen las coordenadas de latitud y longitud (respectivamente) sobre el punto en el que se encuentra situado el ratón en la esquina inferior derecha del mapa. Una vez que se ha seleccionado la región deseada, basta con soltar el ratón y/o teclado y el rango seleccionado se convertirá coordenadas textuales en el cuadro de mandos.

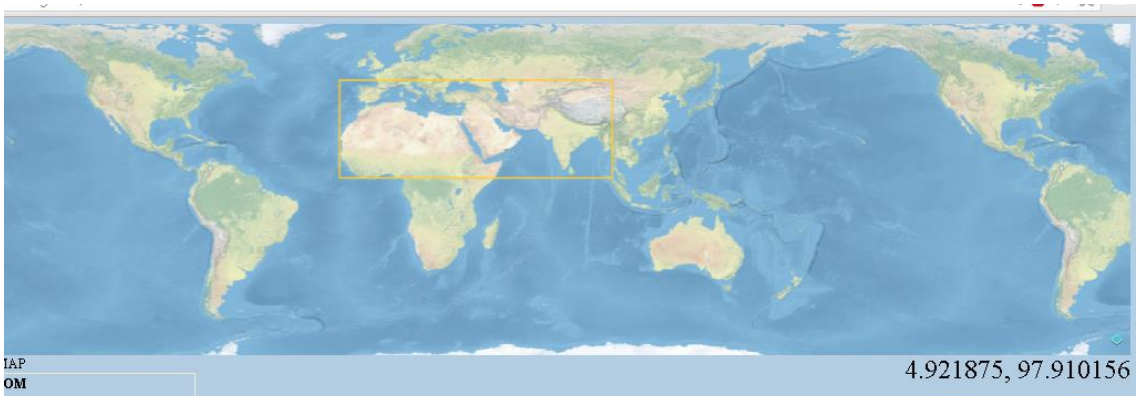


Ilustración 31: Selección sobre el mapa gráfico de la página principal de la aplicación Web

A partir de la selección del rango marítimo y el rango de fecha, solo es necesario hacer clic en el botón "Results!", para tener reflejado de manera gráfica los datos por pantalla:

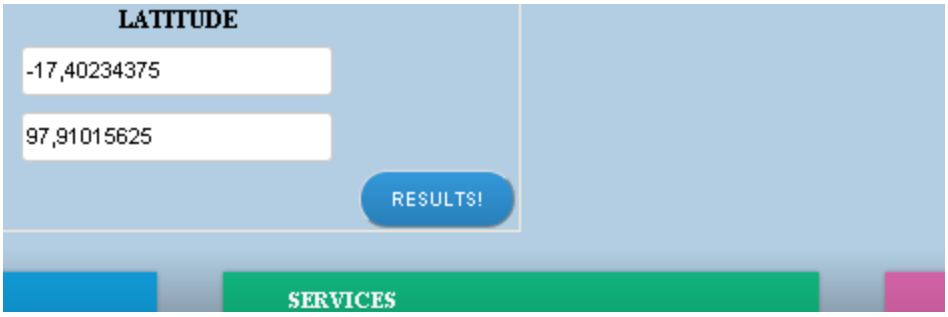


Ilustración 32: Botón del cuadro de manos de la pagina principal para obtener los resultados

Dependiendo del volumen de los datos peticionados, el tiempo de carga variará. El tiempo de carga será variable en función del equipo desde el que se ejecute y la conexión de datos en caso de realizar la petición a un servidor externo. El tiempo de carga no deberá exceder de unos pocos segundos.

A continuación se muestra un ejemplo de la pantalla en la que se exponen los datos geoespaciales para una amplia región comprendida entre los puntos (54, -19) y (14, 133).

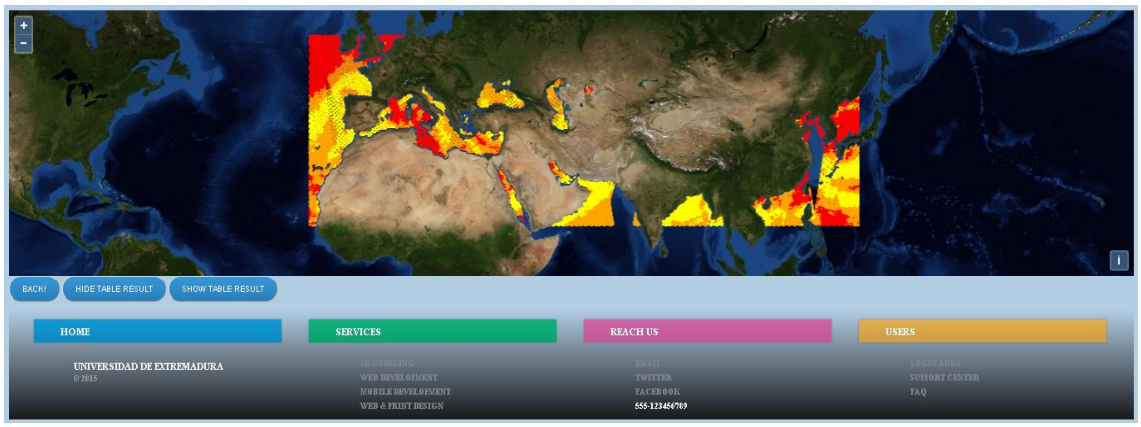
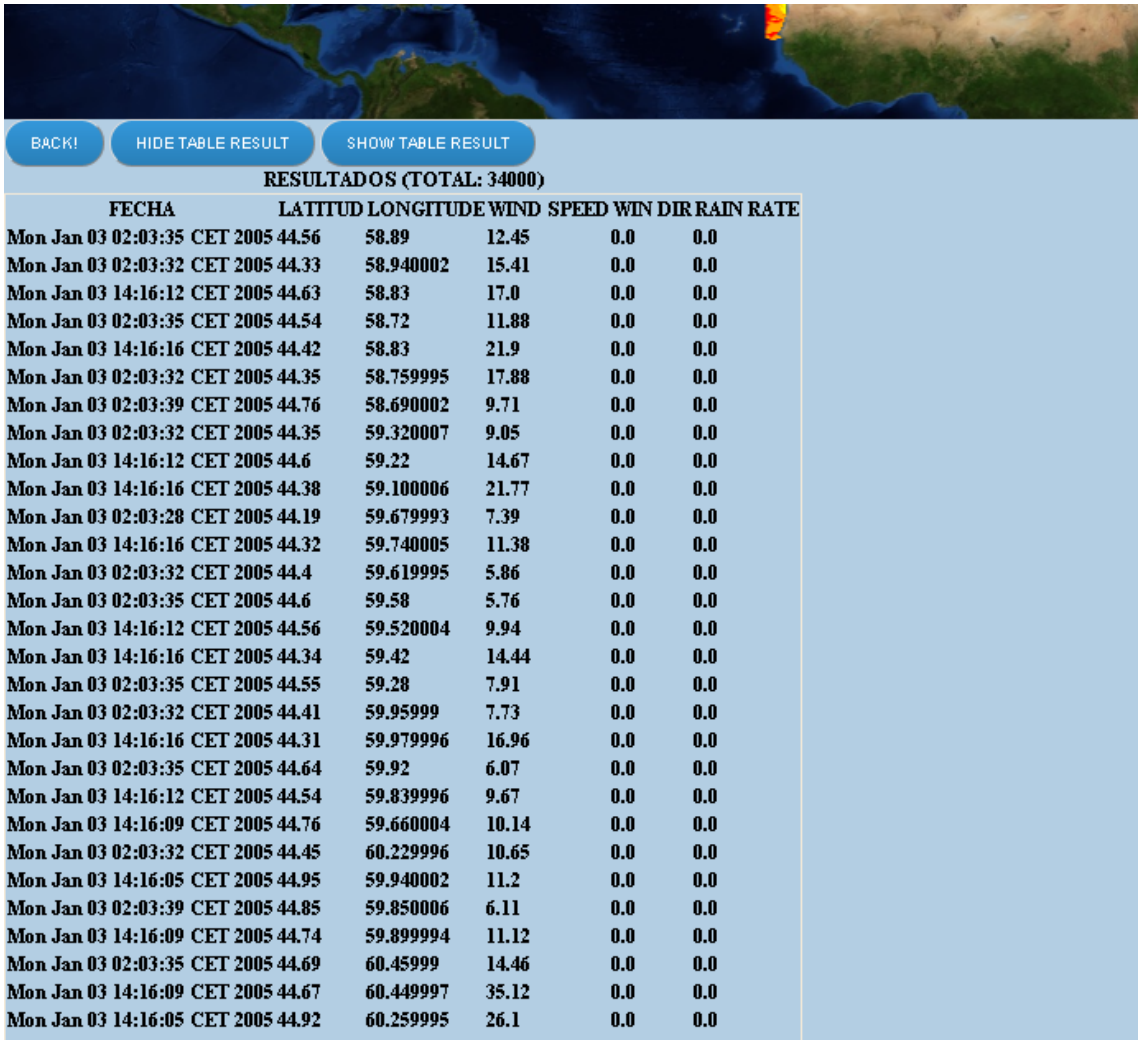


Ilustración 33: Demostración grafica de los datos geoespaciales obtenidos

A través del botón "SHOW TABLE RESULTS" se muestran los resultados en formato textual en forma de tabla.



BACK! HIDE TABLE RESULT SHOW TABLE RESULT

RESULTADOS (TOTAL: 34000)

FECHA	LATITUD	LONGITUDE	WIND SPEED	WIN DIR	RAIN RATE
Mon Jan 03 02:03:35 CET 2005	44.56	58.89	12.45	0.0	0.0
Mon Jan 03 02:03:32 CET 2005	44.33	58.940002	15.41	0.0	0.0
Mon Jan 03 14:16:12 CET 2005	44.63	58.83	17.0	0.0	0.0
Mon Jan 03 02:03:35 CET 2005	44.54	58.72	11.88	0.0	0.0
Mon Jan 03 14:16:16 CET 2005	44.42	58.83	21.9	0.0	0.0
Mon Jan 03 02:03:32 CET 2005	44.35	58.759995	17.88	0.0	0.0
Mon Jan 03 02:03:39 CET 2005	44.76	58.690002	9.71	0.0	0.0
Mon Jan 03 02:03:32 CET 2005	44.35	59.320007	9.05	0.0	0.0
Mon Jan 03 14:16:12 CET 2005	44.6	59.22	14.67	0.0	0.0
Mon Jan 03 14:16:16 CET 2005	44.38	59.100006	21.77	0.0	0.0
Mon Jan 03 02:03:28 CET 2005	44.19	59.679993	7.39	0.0	0.0
Mon Jan 03 14:16:16 CET 2005	44.32	59.740005	11.38	0.0	0.0
Mon Jan 03 02:03:32 CET 2005	44.4	59.619995	5.86	0.0	0.0
Mon Jan 03 02:03:35 CET 2005	44.6	59.58	5.76	0.0	0.0
Mon Jan 03 14:16:12 CET 2005	44.56	59.520004	9.94	0.0	0.0
Mon Jan 03 14:16:16 CET 2005	44.34	59.42	14.44	0.0	0.0
Mon Jan 03 02:03:35 CET 2005	44.55	59.28	7.91	0.0	0.0
Mon Jan 03 02:03:32 CET 2005	44.41	59.95999	7.73	0.0	0.0
Mon Jan 03 14:16:16 CET 2005	44.31	59.979996	16.96	0.0	0.0
Mon Jan 03 02:03:35 CET 2005	44.64	59.92	6.07	0.0	0.0
Mon Jan 03 14:16:12 CET 2005	44.54	59.839996	9.67	0.0	0.0
Mon Jan 03 14:16:09 CET 2005	44.76	59.660004	10.14	0.0	0.0
Mon Jan 03 02:03:32 CET 2005	44.45	60.229996	10.65	0.0	0.0
Mon Jan 03 14:16:05 CET 2005	44.95	59.940002	11.2	0.0	0.0
Mon Jan 03 02:03:39 CET 2005	44.85	59.850006	6.11	0.0	0.0
Mon Jan 03 14:16:09 CET 2005	44.74	59.899994	11.12	0.0	0.0
Mon Jan 03 02:03:35 CET 2005	44.69	60.45999	14.46	0.0	0.0
Mon Jan 03 14:16:09 CET 2005	44.67	60.449997	35.12	0.0	0.0
Mon Jan 03 14:16:05 CET 2005	44.92	60.259995	26.1	0.0	0.0

Ilustración 34: Demostración textual de los datos geoespaciales obtenidos

Pudiendo volver a ocultar la tabla a través del botón “HIDE TABLE RESULT”.

Haciendo clic en el botón “Back!” retrocedemos a la página principal desde donde podemos volver a realizar una petición de datos geoespaciales.

10 ANEXO 2. MANUAL DEL PROGRAMADOR

Como se describe en este documento, el entorno tecnológico que integra la aplicación Web desarrollada lo conforman un número de tecnologías relevante. El proyecto está estructurada y desarrollada de acuerdo al patrón Modelo, Vista y Controlador, se trata de una arquitectura software que separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones. El objetivo de esta arquitectura de software se basa en las ideas de reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento

Los código fuentes que conforman el proyecto se encuentran estructurados en paquetes, clases y ficheros, de acuerdo al estándar de programación estructurada en Java™. Para mantener una estructura que ayude a la organización del proyecto, se han estructurado los elementos de acuerdo al patrón Modelo Vista Controlador previamente explicado y utilizado en este proyecto.

Además en el proyecto se incluyen una serie de ficheros de formato XML para la configuración de los elementos que conforman la estructura del proyecto y sus interacciones.

Para el desarrollo del proyecto se han seguido las pautas de buena programación de Java, tales como: desarrollo del código en inglés; utilización de nombres descriptivos y con una sintaxis adecuada, minúsculas para atributos, lowerCamelCase para métodos y UpperCamelCase para atributos; principio KISS (Keep It Simple, Stupid!), sistemas simples funcionan mejor que si se realizan complejos; etc.

A continuación se muestra la organización de los ficheros del proyecto.

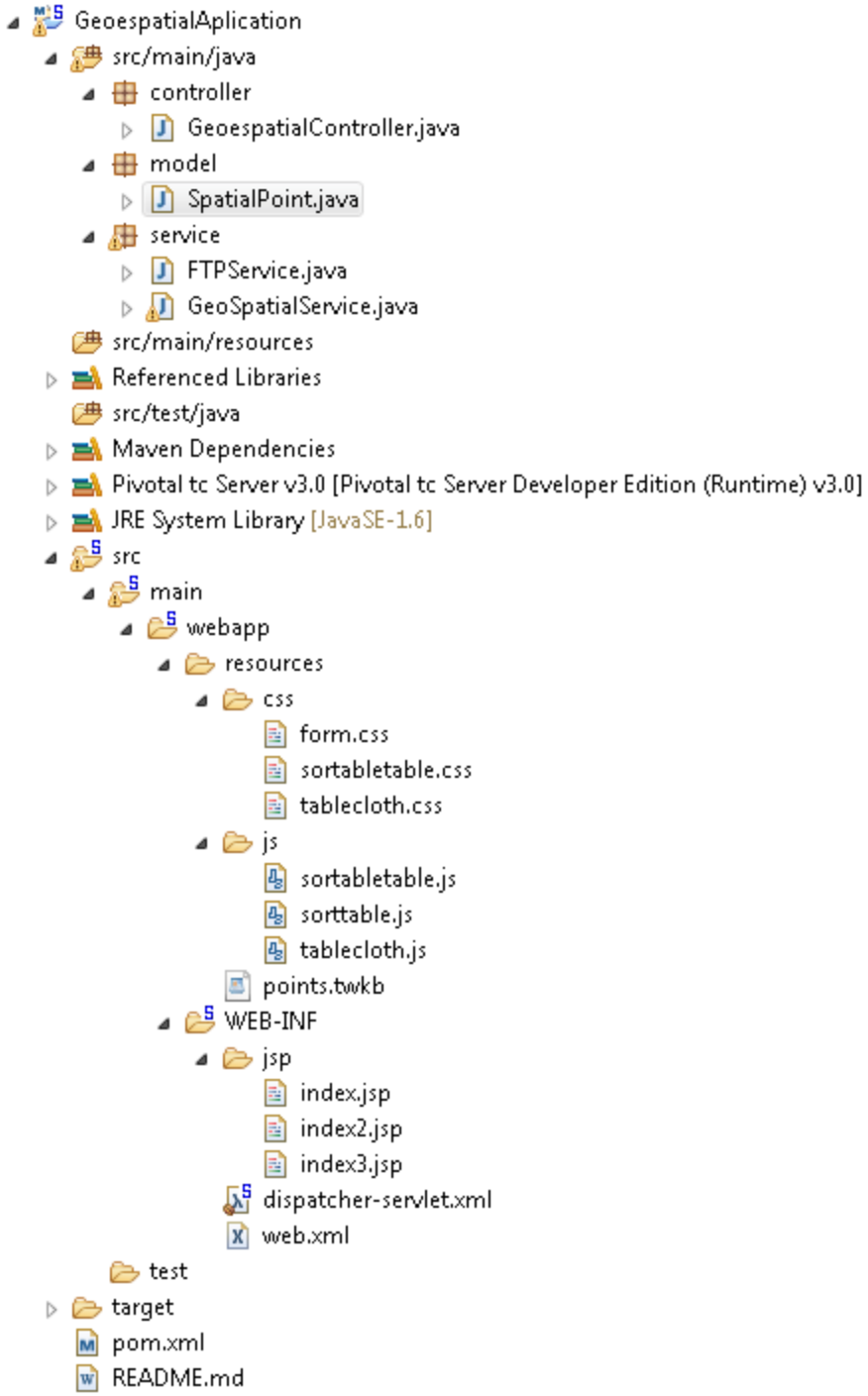


Ilustración 35: Estructura organizativa de la implementación de la Aplicación Web

Seguidamente se detalla la implementación realizada de acuerdo a los elementos que conforman la estructura del proyecto y sus elementos de configuración.

10.1 MODELO

Se trata del elemento de mayor complejidad que conforma el proyecto, se encarga de representar la información con la cual el sistema opera y de gestionar todos los accesos a dicha información desde el repositorio proporcionado, en este caso MongoDB

Se encarga también de enviar a la "vista" aquella parte de la información que en cada momento se le solicita para que sea mostrada al usuario. Las peticiones sobre la información o su procesamiento llegan al modelo a través del controlador.

Básicamente está formado por el modelo con el cual se opera y las operaciones que suceden sobre el modelo. El modelo en este proyecto se trata de los datos geoespaciales representados por la clase "SpatialPoint". Se trata de una clase denominada en programación como POJO cuyas iniciales provienen de "Plain Old Java Object", que puede definirse como "Un objeto Java Plano y a la Antigua". Se trata de una instancia de una clase que no extiende ni implementa nada en especial, pero conforma la estructura de un dato, en este caso de una medición meteorológica. La implementación de la clase es la siguiente:

```
@Document(collection="geoespatial")
@CompoundIndex(collection="geoespatial", name="compundIndex", def = "{'date': 1, 'longitude': 1, 'latitude': 1}")
public class SpatialPoint {


	private Date date; // [fmt="yyyydddHHmmss"]
	@GeoSpatialIndexed
	private float[] location = new float[2]; // location[0]=longitude, location[1]=latitude, [unit="deg"]
	private float windSpeed; //[unit="m/s"]
	private float windDir; //[unit="deg"]
	private float rainRate; //[unit="mm/hr"]
	private float longitude;
	private float latitude;


	public SpatialPoint(Date date, float latitude, float longitude, float windSpeed, float windDir, float rainRate) {
		this.date = date;
		this.latitude=latitude;
		this.longitude=longitude;
```

```
            this.windSpeed = windSpeed;
            this.windDir = windDir;
            this.rainRate = rainRate;

            this.location[0] = longitude;
            this.location[1] = latitude;
      }

      public float[] getLocation() {
            return location;
      }

      public void setLocation(float[] location) {
            this.location = location;
      }

      public float getLongitude() {
            return longitude;
      }

      public void setLongitude(float longitude) {
            this.longitude = longitude;
      }

      public float getLatitude() {
            return latitude;
      }

      public void setLatitude(float latitude) {
            this.latitude = latitude;
      }

      public Date getDate() {
            return date;
      }

      public void setDate(Date date) {
            this.date = date;
      }
```

```
	public float getWindSpeed() {
		return windSpeed;
	}

	public void setWindSpeed(float windSpeed) {
		this.windSpeed = windSpeed;
	}

	public float getWindDir() {
		return windDir;
	}

	public void setWindDir(float windDir) {
		this.windDir = windDir;
	}

	public float getRainRate() {
		return rainRate;
	}

	public void setRainRate(float rainRate) {
		this.rainRate = rainRate;
	}

}
```

La clase POJO "SpatialPoint" cuenta con una serie de atributos y métodos. Los atributos son:

- Date, almacena la fecha de la recogida del dato geoespacial en formato ISO Date Java.
- Location, se trata de un array de tipo float de tamaño dos, conformado por dos elementos: el primero, se trata de la longitud; el segundo, se trata de la latitud. La importancia de declarar la longitud y la latitud como array es para permitir la compatibilidad con el framework Spring-Data-MongoDB utilizado, encargado de transformar la estructura declarada en formato Java, como un documento de la base de datos de MongoDB.
- WindSpeed, se trata de una variable float que almacena la velocidad del viento.

- WindDir, se trata de una variable float que almacena la dirección del viento.
- RainRate , se trata de una variable float que almacena el grado de precipitaciones.
- Se establecen dos atributos redundates. La existencia de estos atributos es permitir el posterior estudio de rendimiento de los diferentes tipos de índices:
    - Longitude, se trata de una variable float que almacena la longitud del punto. Mismo elemento que el primer valor del array location.
    - Latitude, se trata de una variable float que almacena la latitud del punto. Mismo elemento que el segundo valor del array location.

Sobre la clase existen una serie de métodos, de manera general se definen:

- Métodos de tipo “Get”, encargados de retornar los elementos de la estructura de datos
- Métodos de tipo “Set”, encargados de conformar los elementos de la estructura de datos.

Para definir aspectos correspondientes al frameworkd de Spring sobre la clase SpatiaPointSe se declaran una serie de anotaciones, se tratan de anotaciones de tipo Java Beans y son proporcionadas por Spring framework son:

- @Document, permite definir el nombre de la colección de MongoDB asociado al documento que se creara tras su conversión a documento en MongoDB. Es obligatorio la inclusión de un atributo indicando el nombre de la colección asociado, en mi caso la colección recibe el nombre de “geoespatial” (collection = “geoespatial”).
- @CompundIndex, permite definir un índice compuesto automáticamente sobre el documento creado y posteriormente insertado en la colección de MongoDB. Para que la anotación sea funcional debe proporcionarse como atributos: el nombre de la colección; el nombre del índice; y las claves del documento sobre los que se construirá el índice compuesto (dos minimo).
- @GeoSpatialIndexed, permite definir un índice compuesto automáticamente sobre el documento creado y posteriormente insertado en la colección de MongoDB. Se incluye la clave (location) sobre la cual se construirá el índice geoespacial.

La clase modelo, como ya se ha descrito, aparte de representar la información implementa la lógica de negocio. Para la implementación ordenada y estructurada de la lógica de negocio, se han creado una serie de clases que implementen métodos con funcionalidades concretas. Estas clases conforman una capa "service" o capa de servicios, encargada de realizar los tratamiento necesarios con los datos geoespaciales respecto a su consulta, gestión, inserción o borrado.

Para mayor simplicidad aun, cada clase de la capa de servicios implementa una lógica de negocio concreta, así, la clase FTPService, se encarga del tratamiento de los datos para su recuperación y adaptación, proporcionando el acceso y descarga de los ficheros de los datos geoespaciales desde los servidores FTP de la NASA y su posterior descomprensión, interpretación y descompactación en ficheros en formato CSV. La clase GeoespatialService se encarga de ofrecer funciones orientadas al tratamiento de los datos geoespaciales; lectura, adaptación e inserción y consulta en el almacén de datos de MongoDB; así como su consulta y clasificación.

En primer lugar se describirá la clase GeoSpatialService que como ya se ha mencionado se encargada de proveer servicios sobre datos geoespaciales: conexión con base de datos, inserciones y consultas, clasificación, así como métodos auxiliares para dichas funciones. Seguidamente se indica su implementación:

```
@Service
public class GeoSpatialService {


	@Autowired
	private MongoTemplate mongoTemplate;


	public static final String COLLECTION_NAME = "geoespatial";
	private static final boolean categorize = true;


	public void addGeoSpatialPointsFromCSVToDB() throws IOException {
		if (!mongoTemplate.collectionExists(SpatialPoint.class)) {
			mongoTemplate.createCollection(SpatialPoint.class);
		}
try {
			File folder = new File("G:\\QS2005-");
			File[] listOfFiles = folder.listFiles();
```

```
                    for (File file : listOfFiles) {

                                CSVReader datosespaciales = new CSVReader(new
FileReader("G:\\QS2005-\\" + file.getName()));

                                String [] nextLine;

datosespaciales.readNext();datosespaciales.readNext();
                                while ((nextLine = datosespaciales.readNext())
!= null) {
                                   SpatialPoint sp = new
SpatialPoint(juliantoDateIso(nextLine[0]), Float.parseFloat(nextLine[1]),
(((Float.parseFloat(nextLine[2]) + 180) % 360) - 180),
Float.parseFloat(nextLine[3]), Float.parseFloat(nextLine[4]),
Float.parseFloat(nextLine[05]));
                                   System.out.println("INSERTADO" +
file.getName() + "-" + juliantoDateIso(nextLine[0]) +" [" +
(((Float.parseFloat(nextLine[2]) + 180) % 360) - 180) + ", " + nextLine[1] +
"] " + nextLine[3] + " " +nextLine[4] + " " +nextLine[5]);
                                   mongoTemplate.insert(sp, COLLECTION_NAME);
                                }

                                   datosespaciales.close();
                    }
             }
             catch (FileNotFoundException e) { e.printStackTrace();}
             catch (NumberFormatException e) { e.printStackTrace();}


      }


      public List<SpatialPoint> listGeoSpatialPoints(float latitudeOrigin,
float longitudeOrigin, float latitudeDestination, float longitudeDestination,
Date datemin, Date datemax) {
             SimpleDateFormat sdf1 = new SimpleDateFormat("yyyy-MM-
dd'T'HH:mm:ss.SSS'Z'");


             Calendar cal = Calendar.getInstance();
             cal.setTime(datemax);
             cal.add(Calendar.MINUTE, 1);
             datemax = cal.getTime();


             BasicQuery query = new BasicQuery("{ location :{ $geoWithin : {
$box : [ [ "+String.valueOf(longitudeOrigin)+" ,
"+String.valueOf(latitudeOrigin)+" ] , [
"+String.valueOf(longitudeDestination)+" ,
"+String.valueOf(latitudeDestination)+" ] ] } }, date: {$gte: {$date : \""+
sdf1.format(datemin) +"\"}, $lte: {$date: \""+ sdf1.format(datemax) +"\"}}}");



             return mongoTemplate.find(query, SpatialPoint.class,
COLLECTION_NAME);
      }
```

```
	public List<SpatialPoint> listGeoSpatialPointsBetweenTimeAndPoints(String dt1min, String dt2max, Float latitudOrigin, Float longitudOrigin, Float latitudDestination, Float longitudDestination) {


		Date gtDate = null;

		Date lteDate = null;


		try {

			gtDate = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm").parse(dt1min);

			lteDate = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm").parse(dt2max);

		} catch (ParseException e) { e.printStackTrace();}


		System.out.println("Leyendo de la base de datos");


		long TInicio, TFin;

		TInicio = System.currentTimeMillis();

		List<SpatialPoint> sp = listGeoSpatialPoints(latitudOrigin, longitudOrigin, latitudDestination, longitudDestination, gtDate, lteDate);

		TFin = System.currentTimeMillis();

		System.out.println("Tiempo de ejecución en segundos: " + String.valueOf(((TFin-TInicio)/1000)));


		return sp;

	}


	public void geoSpatialPointPrintln(SpatialPoint sp) {

		System.out.println("MOSTRANDO:" + sp.getDate() + "|" + sp.getLatitude() + "|" + sp.getLongitude() + "|" + sp.getRainRate() + "|" + sp.getWindDir() + "|" + sp.getWindSpeed());

	}



	public void deleteDB(){

		mongoTemplate.dropCollection(COLLECTION_NAME);

	}


	public void deleteSpatialPoint(SpatialPoint spatialpoint) {

		mongoTemplate.remove(spatialpoint, COLLECTION_NAME);
```

```
	}

	public ModelAndView categorizeWinds(List<SpatialPoint> sp, ModelAndView model) {
		long TInicio, TFin;
		TInicio = System.currentTimeMillis();

		List<SpatialPoint> yellow = new ArrayList<SpatialPoint>();
		List<SpatialPoint> orange = new ArrayList<SpatialPoint>();
		List<SpatialPoint> red = new ArrayList<SpatialPoint>();

		for (SpatialPoint selectedpoint : sp) {
			if(selectedpoint.getWindSpeed() < 5)
				yellow.add(selectedpoint);
			else if (selectedpoint.getWindSpeed() > 9)
				red.add(selectedpoint);
			else
				orange.add(selectedpoint);
		}


		model.addObject("yellow", yellow);
		model.addObject("orange", orange);
		model.addObject("red", red);

		TFin = System.currentTimeMillis();
		System.out.println("Tiempo de categorizacion en segundos: " + String.valueOf((TFin-TInicio)));

		return model;
	}

	public Date juliantoDateIso (String time) {
		Date date = null;

		try {
			date = new SimpleDateFormat("yyyydddHHmmss").parse(time);

		} catch (ParseException e) { e.printStackTrace();}

		return date;
```

```
	}

	public static boolean isCategorize() {
		return categorize;
	}
}
```

Para establecer el carácter "service" de la clase, se establece la anotación Java Bean proporcionada por Spring "@Service" al comienzo. Se conforman una serie de atributos los cuales se describen:

- mongoTemplate, se trata de una variable que proporciona métodos sobre la base de datos NoSQL de MongoDB. Spring automaticamente, al arranque de la aplicación Web inicializa la variable. La variable proporciona métodos de la base de datos de MongoDB sobre documentos y colecciones, los cuales son redefinidos en la subclase heredera desde al cual se declara (GeoSpatialService).
- COLLECTION_NAME, define el nombre de la colección en MongoDB, sobre la cual se operara, y se realizarán las consultas e inserciones.
- Categorize, se trata de una variable de tipo booleana que eligiendo su opción verdadera permite categorizar los datos geoespaciales de acuerdo a la intensidad de sus vientos para ser ser mostrados en las vistas

Asimismo, se definen sobre la clase una serie de métodos:

- Método "addGeoSpatialPointsFromCSVToDB", se trata de un método que lee los ficheros en formato CSV, formato en el cual se encuentran almacenado los datos geoespaciales, los convierte en una clase Java que los representa (SpatialPoint) y los inserta en una colección de MongoDB proporcionada como atributo. Para leer dichos ficheros CSV hace uso de la librería "CSVReader" que proporciona mecanismo para su lectura de forma más sencilla. Para ello, la librería implementa básicamente una función "readNext" que se encarga de leer la siguiente fila del documento CSV de manera iterativa desde el inicio

hasta el final y facilitar los elementos en un array. Para acceder a los elementos de dicho array basta con indicar la posición del elemento que ocupa en la fila

Para la inserción de las mediciones, además de su lectura, es necesario realizar un tratamiento. El tratamiento consiste en crear una clase referente al modelo SpatialPoint, el cual será proporcionado a mongoTemplate a través de su sentencia insert para la inserción en la base de datos de MongoDB. Al crear un objeto referente la clase SpatialPoint es necesario convertir las longitudes de los datos, para ello se hace uso de la función:

```
((Float.parseFloat(nextLine[2]) + 180) % 360) - 180
```

La función convierte el rango de longitud de 0 a 360 grados a un rango más utilizado de -180 a 180. Esta función esta diseñada por Craig Markwardt para solventar el problema descrito. En caso de no realizar esta operación o realizarla los datos aparecerán invertidos respecto al eje de la longitud.

Por otro lado también es necesario transformar las fechas en un formato legible y entendible por la base de datos de MongoDB para ello se realiza la siguiente función:

```
juliantoDateIso(nextLine[0])ç
```

Se encarga de transformar las fechas de **formato juliano** a ISO Date Java.

Nota: Este método "addGeoSpatialPointsFromCSVToDB" debe modificarse manualmente para proveer la fuente de los ficheros a leer.

- listGeoSpatialPointsBetweenTimeAndPoints, se trata de un método encargado de devolver en un Arraylist los datos geoespaciales comprendidos en una región terrestre y temporal. No realiza la consulta directamente a la base de datos. Se encarga, por un lado de transformar los formatos de las fechas retornados por el controlador (y este a su vez de las vista) a un formato adaptado. Por otro lado, se encarga de cuantificar el tiempo que tarda en realizarse la consulta a la base de datos de MongoDB. Realiza la llamada al

método listGeoSpatialPoints el cual si realiza la consulta de los documentos en la base de datos de MongoDB.

- listGeoSpatialPoints, realizar la consulta de los documentos comprendidos en una localización geográfica y temporal. En primer lugar adapta nuevamente las fechas en un formato apropiado para para poder incrementar el tiempo, de forma que ante un rango de tiempo, por ejemplo de 25 a 30, compute también los datos del minuto 30 añadiendo un minuto más (31).

  Realiza la consulta de los documentos correspondientes a los datos geoespaciales requeridos en la consulta a la base de datos de MongoDB utilizando la sintaxis JavaScript propia de MongoDB.

- juliantoDateIso, se encarga de transformar las fechas de formato juliano (yyyydddHHmmss), donde: los cuatro primeros dígitos forman el año; los tres siguientes el día a partir del comienzo del año; los dos siguientes la hora; los dos siguientes el minuto; y los dos siguientes y últimos los segundos. A un formato más utilizable por java, el formato Date ISO Java, encontrado en el paquete java.util, que encapsula como valor Long (numérico de gran tamaño) un momento específico en el tiempo.
- deleteDB() y deleteSpatialPoint. Son métodos auxiliares para realizar el borrados de todos los elementos existente en una colección y/o la propia colección. No se utilizan en ninguna operación.
- categorizeWinds. Se encarga dado un ArrayList de datos geoespaciales de categorizar los mismos de acuerdo a la velocidad de los vientos, estableciendo tres categorías (fuertes, medios y leves) de acuerdo a tres listas. Cada lista será insertada en el Modelo que proporciona Spring framework para ser visualizado a través de las vistas.

La clase FTPService, se encarga de proveer métodos y funciones para la conexión y descarga del servidor FTP de la NASA donde se encuentran alojados los datos, así como la posterior descomprensión de los ficheros y descompactación, mediante su interpretación como fichero HDF obteniendo como resultado fichero en formato CSV.

Se detalla la implementación de la clase FTPService:

```
@Service
public class FTPService{



      //DIRECCIÓN SERVIDOR= ftp://podaac.jpl.nasa.gov/allData/rapidscat/
      private String clienthostname="podaac.jpl.nasa.gov";
      private String clienthostusername="Anonymous";
      private String clienthostpassword="uni@uni.com";
      private String clienthostpath="/allData/quikscat/L2B/v2/2009/001";
      private String pathtodowload="C:\\archivosnasa";

       public boolean dowloadFiles(){
              try {
                  FTPClient ftp = new FTPClient();
                  ftp.connect(clienthostname);

                  if(!ftp.login(clienthostusername, clienthostpassword)){
                      ftp.logout();
                      return false;
                  }

                  int reply = ftp.getReplyCode();

                  if (!FTPReply.isPositiveCompletion(reply)){
                      ftp.disconnect();
                      return false;
                  }

                  ftp.enterLocalPassiveMode();
                  ftp.changeWorkingDirectory(clienthostpath);

                  System.out.println("Current    directory    is    "    +
ftp.printWorkingDirectory());

                  FTPFile[] ftpFiles = ftp.listFiles();

                  if (ftpFiles != null && ftpFiles.length > 0) {
                      for (FTPFile file : ftpFiles) {
                          if (!file.isFile()) {
                              continue;
                          }
```

```
                        System.out.println("File is " + file.getName());

                        OutputStream output;
                        output = new FileOutputStream(pathtodowload + "/" +
file.getName());
                        ftp.retrieveFile(file.getName(), output);
                        output.close();
                    }
                }

                ftp.logout();
                ftp.disconnect();
            }
            catch (Exception ex){
                 ex.printStackTrace();
                return false;
            }
            return true;
        }

          public Vector<String> listFiles (boolean comprimidosGZ) {
                Vector<String> fichList = new Vector<String>();
                File fichero = new File(pathtodowload);
                String [] listaQSfichs;
                boolean listar;

                System.out.println("* Seleccionando en " + pathtodowload +
" los ficheros descomprimidos");

                if (!fichero.exists() || !fichero.canRead())
                      System.out.println("No   puedo   acceder   a   "   +
pathtodowload);
                else {
                      if (fichero.isDirectory()) {
                            listaQSfichs = fichero.list();
                            for (int i = 0; i < listaQSfichs.length; i++){
                                  if (listaQSfichs[i].startsWith("QS")){
                                        listar                          =
(listaQSfichs[i].endsWith(".gz")   ||   listaQSfichs[i].endsWith(".GZ"))   ?
comprimidosGZ : (!comprimidosGZ);
                                        if (listar)
```

```
      fichList.add(listaQSfichs[i]);
                                    }
                                }
                            }
                            else System.out.println(pathtodowload + " no es un
directorio.");
                    }
                    return fichList;
            }

            public String uncompressQSFiles (String ficheroGZIP, int
kBuffSize){
                    File file = new File(pathtodowload+ "\\" + ficheroGZIP);
                    String namefileuncompress = new
String(ficheroGZIP.substring(0,ficheroGZIP.length()-3));
                    File uncompressfile = new File(pathtodowload+ "\\" +
namefileuncompress);

                    System.out.println("* Descomprimiendo " + ficheroGZIP + "
en " + uncompressfile.getName());

                    try {
                            GZIPInputStream in = new GZIPInputStream(new
FileInputStream(file));
                            OutputStream out = new
FileOutputStream(uncompressfile);
                            byte[] buf = new byte[1024*kBuffSize];
                        int len;
                        while ((len = in.read(buf)) > 0)
                            out.write(buf, 0, len);
                        in.close();
                        out.close();
                    }
                    catch(IOException e) {
                            e.printStackTrace();
                            return null;
                    }
                    return uncompressfile.getName();
            }


            public boolean uncompactFilesToCSV (){
```

```
                    System.out.println("Interpretacion de observaciones HDF-4 de QuikSCAT");
                    String CdirFiles = "archivosnasa"; //File of directory of C:// where is the executable and files


                    try {
                          System.out.println("COmenzando proceso");



                          Process p = Runtime.getRuntime().exec(new String[]{" cd ..", " cd ..", "cd " + CdirFiles , "C:\\archivosnasa\\QSreadLFx64.exe C:\\archivosnasa\\ficheroIntQSlf 1 0"});


                          System.out.println("Finalizando proceso");


                          BufferedReader buffer = new BufferedReader(new InputStreamReader(p.getInputStream()));
                          System.out.println("Buffer y tal");
                          String t;
                          while ((t = buffer.readLine()) != null) {
                                System.out.println(t);
                          }
                    }
                    catch (Exception e){
                          //System.out.println("En lanzamiento del proceso con parametros: " + ficheroIntQSlf);
                    }
                    return true; // proceso ejecutado
              }
}
```

Mediante la anotación Java Bean proporcionada por Spring framework "@Service" se declara que se trata de una clase que provee servicios al modelo.

Se definen una serie de atributos sobre la clases:

- Clienthostname, dirección del servidor FTP sobre el que se realiza la conexión para descargar los ficheros.
- Clienthostusername, nombre de usuario con el que se realizará la conexión con el servidor FTP.

- Clienthostpassword, contraseña referente al nombre de usuario con el que se realizará la conexión con el servidor FTP.
- Clienthostpath, dirección relativa referente a un directorio particular del servidor FTP del cual se descargaran los datos geoespaciales.
- Pathtodowload, directorio del equipo local donde se almacenarán los ficheros descargados.

Se definen una serie de métodos sobre la clase FTPService:

- dowloadFiles, realiza la conexión con el servidor FTP declarado en el atributo clienthostname. El servidor FTP actual se trata del servidor FTP PODAAC (Physical Oceanography Distributed Archive Center) donde se encuentran alojados todos los datos generados por el satélite QuikSCAT y RapidScat.
- listFiles, lista los ficheros de un directorio, en caso de proporcionar una variable true al método retorna aquellos comprimidos exclusivamente y en caso contrario, devuelve todos los ficheros.
- uncompressQSFiles, descomprime los ficheros proporcionados como atributo.
- uncompactFilesToCSV, des compacta los ficheros mediante su interpretación como ficheros HDF a través de un ejecutable llamado como rutina y al cual se le otorgan como parámetro la ruta de los fichero en formato CSV a interpretar..

10.2 CONTROLADOR

Se encarga de reaccionar a las peticiones de los usuarios, ejecutando las acciones adecuadas y creando el modelo pertinente. Responder a los eventos realizados por el usuario invocando peticiones al modelo cuando se hace alguna solicitud sobre la información.

La implementación de la clase controller es:

```
@Controller
public class GeoespatialController {


	@Autowired
```

```
	private FTPService ftpservice;
	@Autowired
	private GeoSpatialService geospatialservice;


    @RequestMapping(value = "/", method = RequestMethod.GET)
	public String principalPage(ModelMap model) throws IOException {


	//ftpservice.lanzarP();
	//geospatialservice.addGeoSpatialPointsFromCSVToDB();
	return "index";
    }


    @RequestMapping(value = "/paginaResultado", method = RequestMethod.POST)
	public ModelAndView pageResults(HttpServletRequest request,
@RequestParam(value="date1", required=false, defaultValue = "") String date1,
		@RequestParam(value="date2", required=false, defaultValue
= "") String date2, @RequestParam(value="longitudeorigin", required=true)
Float longitudeorigin,
		@RequestParam(value="latitudeorigin", required=true) Float
latitudeorigin, @RequestParam(value="longitudedestination", required=true)
Float longitudedestination,
		@RequestParam(value="latitudedestination") Float
latitudedestination) throws IOException {


	System.out.println("Fecha inicial : " + date1);
	System.out.println("Fecha final : " + date2);


	System.out.println("Punto origen: [latitud:" + latitudeorigin + "|
lontitud: " + longitudeorigin + "]");
	System.out.println("Punto destino: [latitud: " + latitudedestination +
"| longitud: " + longitudedestination + "]");


	List<SpatialPoint> sp =
geospatialservice.listGeoSpatialPointsBetweenTimeAndPoints(date1, date2,
latitudeorigin, longitudeorigin, latitudedestination, longitudedestination);


	System.out.println("Numero de elementos:" + sp.size());
	ModelAndView model = new ModelAndView("index2");


	if(GeoSpatialService.isCategorize()) {
		model= geospatialservice.categorizeWinds(sp, model);
	}


	model.addObject("listspatialpointfilter", sp);
```

```
        return model;
    }
}
```

La clase controladora de Spring se estructura usando un patrón de comandos que encapsulan las acciones y simplifican su extensión. Para identificar una clase como controladora del patrón se declara con la anotación Java Bean proporcionada por Spring framework como "@Controller" indicando que la clase se encarga de las funciones de controlador.

Se definen una serie de atributos sobre la clase, estos atributos reciben la anotación @Autowired y se encargan de "auto-inyectar" las interacciones del controlador con el modelo. Los atributos son:

- Ftpservice, se trata de una variable correspondiente a la clase FTPService y se encarga de proporcionar los métodos y funciones para la conexión y descarga del servidor FTP donde se encuentran alojados los datos, así como su interpretación.
- Geospatialservice, se trata de una variable correspondiente a la clase GeoSpatialService, encargada de proveer servicios sobre los datos geoespaciales: conexión con base de datos, inserciones y consultas, así como métodos auxiliares para dichas funciones.

Se definen de manera particular dos controladores, estos dos controladores ofrecen las dos vistas diferentes de acuerdo a las interfaces graficas elaboradas para la aplicación Web. Los dos controladores se definen a través de anotaciones java Beans de Spring framework y se definen como "@RequestMapping" (peticiones de mapeo), sobre la anotación se declara un valor, que será la URL de la aplicación sobre la que actuara dicho controlador y el carácter del método del controlador, es decir si se trata de tipo GET o de tipo POST, es decir obtiene o devuelve datos. Los dos controladores declarados son:

- paginaPrincipal, se trata del controlador más básico, simplemente muestra la vista correspondiente a la página principal de la aplicación.
- paginaResultados, por un lado se encarga de obtener los parámetros, proporcionados por la vista de la página principal para realizar la consulta sobre

los datos geoespaciales a ser mostrador posteriormente en la siguiente vista. Ofrece la vista sobre la que se muestran esos datos. Comprueba además si existe la variable correspondiente al modelo "isCategorize" y si está activa, en ese caso, categoriza los vientos en tres tipos (fuertes, medios y débiles).

10.3 VISTA

Se trata básicamente de lo que el usuario ve, la interfaz gráfica de la aplicación Web. Las vistas se han desarrollado a través de páginas de formato JSP (Java Server Pages), que no contienen ninguna lógica de negocio, ni flujos, únicamente sólo tags encargados de representar la información. Utiliza el modelo del patrón generado para obtener la información y representarla.

En las páginas de formato JSP el código es compilado como cualquier otra clase Java. La máquina virtual compilará dinámicamente a código de máquina las partes de la aplicación que lo requieran. Esto hace que JSP tenga un buen desempeño y sea más eficiente que otras tecnologías Web que ejecutan el código de una manera puramente interpretada.

La principal ventaja de las paginas JSP es que utilizan Java como lenguaje de propósito general y que es apto para crear clases que manejen lógica de negocio y acceso a datos de una manera prolija. Esto permite separar en niveles las aplicaciones Web, dejando la parte encargada de generar el documento HTML en el archivo JSP.

Otra ventaja es que JSP hereda la portabilidad de Java, y es posible ejecutar las aplicaciones en múltiples plataformas sin cambios. Es común incluso que los desarrolladores trabajen en una plataforma y que la aplicación termine siendo ejecutada en otra.

Las páginas de formato JSP se tratan de un método más de los existentes para la creación de páginas Web dinámicas en servidor usando el lenguaje Java. En ese sentido son similares a otros métodos o lenguajes tales como el PHP, ASP o los CGIs, programas que generan páginas Web en el servidor. Sin embargo se han desarrollado páginas en formato JSP por las siguientes ventajas:

- Las paginas JSP se ejecutan en una máquina virtual Java, lo cual permite que, en principio, se puedan usar en cualquier tipo de ordenador, siempre que exista una máquina virtual Java para él.
- Cada página JSP ejecuta en su propio hilo, es decir, en su propio contexto; pero no se comienza a ejecutar cada vez que recibe una petición, sino que persiste de una petición a la siguiente, de forma que no se pierde tiempo en invocarlo.
- Por su persistencia le permite también hacer una serie de cosas de forma más eficiente: conexión a bases de datos y manejo de sesiones, por ejemplo.

Las JSPs utilizan una forma alternativa de crear servlets ya que el código JSP se traduce a código de servlet Java la primera vez que se le invoca y en adelante es el código del nuevo servlet el que se ejecuta produciendo como salida el código HTML que compone la página Web de respuesta.

Para el desarrollo de las vistas se ha utilizado en su composición, además de la sintaxis propia de las páginas JSP, se ha utilizado:

- HTML, se trata del estándar de referencia para la elaboración de páginas, permite define una estructura básica y un código (denominado código HTML) para la definición de contenido de una página Web, como texto, imágenes, videos, entre otros. Es un estándar a cargo de la W3C, organización dedicada a la estandarización de casi todas las tecnologías ligadas a la Web, sobre todo en lo referente a su escritura e interpretación. Proporciona la estructura de la página Web.
- CSS, este lenguaje se ha usado para definir y crear la presentación de las distintas, permitiendo su uso en documentos escritos en HTML, como en este caso, o en XML2 (y por extensión en XHTML). El World Wide Web Consortium (W3C) es el encargado de formular la especificación de las hojas de estilo que servirán de estándar para los agentes de usuario o navegadores.

  CSS permite separar la estructura de un documento de su presentación. Como en el proyecto desarrollado, la información del estilo puede ser definida en un documento separado, a través de ficheros de extensión CSS, o también puede definirse en el mismo documento HTML, esta forma se despreció para separar

las vistas en estructura y listo de acuerdo a los estándares de programación Web actuales.

- JavaScripst se trata de un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos basado en prototipos, imperativo, débilmente tipado y dinámico. JavaScript se utiliza en el lado del cliente, permitiendo mejoras en la interfaz de usuario y páginas Web dinámicas.

  JavaScript se ha utilizado en el proyecto para implementar todas las funciones referentes al mapa gráfico a través de OpenLayers 3

Se han desarrollado dos vistas principalmente: una se trata de la página principal, la que el usuario ve en primer lugar al acceder a la aplicación, la otra, se trata de la página para mostrar los resultados referentes a la petición realizada desde la página principal. A continuación se detallan.

10.3.1 PÁGINA PRINCIPAL

La página principal es la primera vista que el usuario se encuentra al acceder. Se trata del *home* de la aplicación. La página principal se encarga de capturar los datos que ofrecerá al modelo para la gestión y posterior mostrado de los resultados en la página de resultados.

A continuación se detalla la vista principal en cuanto a su implementación.

En primer lugar, se declaran las librerías y fuentes externas que se utilizan

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>


<link type="text/css" rel="StyleSheet" href="<c:url
value="/resources/css/form.css"/>"/>
<link rel="stylesheet" href="http://openlayers.org/en/v3.5.0/css/ol.css"
type="text/css">
<script src="http://openlayers.org/en/v3.5.0/build/ol.js"
type="text/javascript"></script>
```

Las librerías y fuentes externas se tratan de:

- La librería JSTL, se trata de un conjunto de librerías de etiquetas simples y estándares que encapsulan la funcionalidad principal para escribir páginas JSP, se encarga de proporcionar los tags para hacer uso de funciones como “c:url”.
- Las fuentes del estilo CSS desarrolladas para el proyecto y en concreto para esta vista, la cual se trata de “form.css”.
- Las librerías correspondientes al mapa grafico implementado mediante OpenLayers, las cuales se tratan en primer lugar de un documento de tipo CSS y un documento JavaScript.

Se declaran mediante HTML un formulario con un formato de tabla para introducir los parámetros por el usuario. Se declara en dicho formulario la especiación “form action”, que indica que al pulsar el botón “Results!”, se transfiéralos y procese los valores de los campos introducidos por el usuario al modelo encargado de la lógica de negocio, cuyo controlador estará asociada a la URL "/HelloSpringWithMongoDB/paginaResultado" y cambie la vista a la página de resultados.

```
<body>
   <form action="/HelloSpringWithMongoDB/paginaResultado" method="POST">
      <table>
            <tr>
                  <td></td>
                  <td class="ui-helper-center">TO</td>
                  <td class="ui-helper-center">FROM</td>
            </tr>
            <tr>
                  <td>RANGE DATE</td>
                  <td><input name="date1" type="datetime-local" name="Fecha
inicio" value="" /> </td>
                  <td><input name="date2" type="datetime-local" name="Fecha
final" value="" /> </td>
            </tr>
            <tr>
                  <td></td>
                  <td class="ui-helper-center">LONGITUDE</td>
                  <td class="ui-helper-center">LATITUDE</td>
            </tr>
```

```
            <tr>
                  <td>INICIAL POINT</td>
                  <td><input    name="latitudeorigin"    id="latitudeorigin"
type="number" value="" step="any"/> </td>
                  <td><input   name="longitudeorigin"   id="longitudeorigin"
type="number" value="" step="any"/> </td>
            </tr>
            <tr>
                  <td>FINAL POINT</td>
                  <td><input                        name="latitudedestination"
id="latitudedestination" type="number" value="" step="any"/> </td>
                  <td><input                        name="longitudedestination"
id="longitudedestination" type="number" value="" step="any"/> </td>
            </tr>
            <tr>
                  <td> <td>
                  <td> <td>
                  <td> <td>
                  <button                                          class="btn"
type="submit"><b>RESULTS!</b></button></td>
            </tr>
      </table>
     </form>
  </body>
```

En dicho formulario, se define un punto inicial y un punto final, cada uno de ellos formado por un campo longitud y latitud de tipo “input” (entrada de datos). De dicho formulario se extraerá los valores del rango temporal y espacial de la consulta. El rango puede indicarse de manera textual sobre los campos declarados o de manera gráfica mediante el mapa grafico implementado mediante OpenLayers. El rango de la fecha de estudio es siempre obligatorio.

Se definen otros elementos HTML en la vista, los cuales se detallan a continuación.

Una cabecera en la vista, indicando el título de la aplicación Web para el navegador y un elemento de tipo “map”, situado en el cuerpo de la vista. Dicho elemento “map” se describirá más adelante en detalle.

```
<head>
    <title>GEOESPATIAL</title>
```

```
</head>
<body>
      <div id="map" class="map"></div>
```

Un pie de página, a modo decorativo y con posibles futuras funcionalidades. Se implementa con estructura de listas HTML.

```
<footer>
  <ul>
    <li>
     <p class="home">Home</p>
     <a class="logo" href="#">UNIVERSIDAD DE EXTREMADURA<i>&copy;2015</i></a>
    </li>
    <li>
     <p class="services">Services</p>
       <ul>
           <li><a href="#">3D modeling</a></li>
           <li><a href="#">Web development</a></li>
           <li><a href="#">Mobile development</a></li>
           <li><a href="#">Web &amp; Print Design</a></li>
        </ul>
    </li>
    <li>
      <p class="reachus">Reach us</p>
       <ul>
           <li><a href="#">Email</a></li>
           <li><a href="#">Twitter</a></li>
           <li><a href="#">Facebook</a></li>
           <li>555-123456789</li>
       </ul>
    </li>
    <li>
      <p class="clients">Users</p>
       <ul>
           <li><a href="#">Login Area</a></li>
           <li><a href="#">Support Center</a></li>
           <li><a href="#">FAQ</a></li>
       </ul>
   </li>
```

```
  </ul>
</footer>
```

Sin duda alguna el elemento principal de la vista es el mapa gráfico desarrollado mediante OpenLayers y JavaScript. OpenLayers se trata de una biblioteca JavaScript de código abierto bajo una derivación de la licencia BSD para mostrar mapas interactivos en los navegadores Web. OpenLayers ofrece un API para acceder a diferentes fuentes de información cartográfica en la red. Su implementación no es sencilla, a continuación se detalla el código construido.

En primer lugar, se realiza la inserción de un mapa básico. Este mapa básico se tratará del mapa sobre el que se irán añadiendo funcionalidades y métodos, incrementando su complejidad y permitiendo obtener un mapa grafico e interactivo que satisfaga nuestros requerimientos.

```
var map = new ol.Map({
    target: 'map',
    renderer: 'canvas',
    layers: layers, //CAMBIAR MAPA
    controls: ol.control.defaults({
         attributionOptions:
         collapsible: false
         })
     }).extend([mousePositionControl]),
     view: new ol.View({
       projection: 'EPSG:4326',
           center: [0, 0],
           zoom: 2
      }),
      interactions: ol.interaction.defaults().extend([box])
});
```

Para implementar este mapa básico es necesario declarar algunos atributos como:

- Target, se trata del “objetivo”. Se declara “map”, para implementar el mapa.

- Renderer, se trata del estándar que renderizará el mapa. Por sencillez y rendimiento se utiliza "canvas", un elemento HTML incorporado en HTML5 que permite la generación de gráficos dinámicamente por medio del scripting. Permite generar gráficos estáticos y animaciones. También se puede utilizar DOM o WebGL.
- Layers, se trata del mapa que se va a utilizar. Es necesario definirlo previamente.
- Controls, se trata de controles adicionales que se implementan sobre el mapa:
    - Se define para esta vista, "mousePositionControl".
- View, se trata de la proyección que se utilizará, el zoom con el que se mostrará el mapa y el punto centrado para la proyección. La proyección se trata del tipo de mapa, en este caso hemos elegido el mapa "EPSG:4326", que la imagen este centrada en el punto de longitud 0 y latitud 0 y un zoom de grado 2.
- Interactions, se tratan de interacciones que se realicen para el mapa.
    - Se define para esta vista, "box".

Se definen dos tipos de mapas gráficos para el mapa previamente implementado, los cuales modificando el atributo "layers por "layers" o "raster" se procede a su modificación. El primero de los mapas implementa un mapa geofísico y el segundo un mapa satélite.

```
    var layers = [ // MAPA GEOFISICO
        new ol.layer.Tile({
          source: new ol.source.TileWMS({
                url: 'http://demo.boundlessgeo.com/geoserver/wms',
                params: {
                      'LAYERS': 'ne:NE1_HR_LC_SR_W_DR'
                }
           })
        })
    ];

  var raster = new ol.layer.Tile({ // MAPA SATELITE
      source: new ol.source.MapQuest({
          layer: 'sat'
```

```
        })
    });
```

Asimismo se añade al mapa, la interacción "dragBox", funcionalidad que permite que al pulsar SHIFT + doble clic del ratón, se puede extender un cuadrado de selección para establecer un rango en el mapa y luego posteriormente transfiera el punto inicial y final a los campos de longitud y latitud respectivos.

```
var box = new ol.interaction.DragBox({
    condition: ol.events.condition.shiftKeyOnly,
    style: new ol.style.Style({
        stroke: new ol.style.Stroke({
            color: '#ffcc33',
            width: 2
        })
    })
})

box.on('boxend', function() {
    var split = box.getGeometry().getCoordinates().toString().split(',');
       //   1               4
      // (X, Y)          (X, Y)
      x1=split[0]; x4=split[6];
      y1=split[1]; y4=split[7];
      //   2                3
      // (X, Y)          (X, Y)
      x2=split[2]; x3=split[4];
      y2=split[3]; y3=split[5];

      document.getElementById("longitudeorigin").value=x1;
      document.getElementById("latitudeorigin").value=y1;
      document.getElementById("longitudedestination").value=x3;
      document.getElementById("latitudedestination").value=y3;
});
```

Sobre el mapa se implementa otra funcionalidad mas para que al colocar el ratón sobre el mapa nos muestre los valores textuales de latitud y longitud (respectivamente) sobre la zona en la cual está situado el ratón.

```
var myFormat = function(dgts){
    return (
      function(coord1) {
          var coord2 = [coord1[1], coord1[0]];
          return ol.coordinate.toStringXY(coord2,dgts);
     });
}

 var mousePositionControl = new ol.control.MousePosition({
     coordinateFormat: myFormat(6), //DECIMALES
     projection: 'EPSG:4326',
     className: 'custom-mouse-position',
     target: document.getElementById('mouse-position'),
     undefinedHTML: ' '
 });
```

A continuación se presenta la vista en su conjunto:

```
<!doctype html>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>


<html lang="en">


<link type="text/css" rel="StyleSheet" href="<c:url value="/resources/css/form.css"/>"
/>
<link        rel="stylesheet"        href="http://openlayers.org/en/v3.5.0/css/ol.css"
type="text/css">
<script                              src="http://openlayers.org/en/v3.5.0/build/ol.js"
type="text/javascript"></script>




<head>
    <title>GEOESPATIAL</title>
</head>
<body>
       <div id="map" class="map"></div>


       <div class="ui-helper-right" id="mouse-position"></div>
    PRESS SHIFT KEY + DOUBLE CLICK TO SELECT RANGE IN MAP
    <form action="/HelloSpringWithMongoDB/paginaResultado" method="POST">
              <table>
```

```
                    <tr>
                            <td></td>
                            <td class="ui-helper-center">TO</td>
                            <td class="ui-helper-center">FROM</td>
                    </tr>
                    <tr>
                            <td>RANGE DATE</td>
                            <td><input name="date1" type="datetime-local" name="Fecha
inicio" value="" /> </td>
                            <td><input name="date2" type="datetime-local" name="Fecha
final" value="" /> </td>
                    </tr>
                    <tr>
                            <td></td>
                            <td class="ui-helper-center">LONGITUDE</td>
                            <td class="ui-helper-center">LATITUDE</td>
                    </tr>
                    <tr>
                            <td>INICIAL POINT</td>
                            <td><input    name="latitudeorigin"    id="latitudeorigin"
type="number" value="" step="any"/> </td>
                            <td><input   name="longitudeorigin"   id="longitudeorigin"
type="number" value="" step="any"/> </td>
                    </tr>
                    <tr>
                            <td>FINAL POINT</td>
                            <td><input                    name="latitudedestination"
id="latitudedestination" type="number" value="" step="any"/> </td>
                            <td><input                   name="longitudedestination"
id="longitudedestination" type="number" value="" step="any"/> </td>
                    </tr>
                    <tr>
                            <td> <td>
                            <td> <td>
                            <td> <td>
                            <button                                        class="btn"
type="submit"><b>RESULTS!</b></button></td>
                    </tr>
      </table>
      </form>
  </body>
   <footer>
    <ul>
        <li>
            <p class="home">Home</p>
            <a class="logo" href="#">UNIVERSIDAD DE EXTREMADURA<i>&copy; 2015</i></a>
```

```
        </li>
        <li>
            <p class="services">Services</p>

            <ul>
                <li><a href="#">3D modeling</a></li>
                <li><a href="#">Web development</a></li>
                <li><a href="#">Mobile development</a></li>
                <li><a href="#">Web &amp;amp; Print Design</a></li>
            </ul>
        </li>
        <li>
            <p class="reachus">Reach us</p>

            <ul>
                <li><a href="#">Email</a></li>
                <li><a href="#">Twitter</a></li>
                <li><a href="#">Facebook</a></li>
                <li>555-123456789</li>
            </ul>
        </li>
        <li>
            <p class="clients">Users</p>

            <ul>
                <li><a href="#">Login Area</a></li>
                <li><a href="#">Support Center</a></li>
                <li><a href="#">FAQ</a></li>
            </ul>
        </li>
    </ul>
</footer>
    <script type="text/javascript">
    //----------------------SELECCIÓN TIPO MAPA-------------------------//
        var layers = [ // MAPA GEOFISICO
            new ol.layer.Tile({
               source: new ol.source.TileWMS({
                      url: 'http://demo.boundlessgeo.com/geoserver/wms',
                      params: {
                             'LAYERS': 'ne:NE1_HR_LC_SR_W_DR'
                    }
                })
            })
```

```
    ];

var raster = new ol.layer.Tile({ // MAPA SATELITE
    source: new ol.source.MapQuest({
        layer: 'sat'
    })
});
//---------------------FIN SELECCIÓN TIPO MAPA-----------------------//

//------------------MOSTRAR LONGITUDES Y LATITUDES------------------//
var myFormat = function(dgts){
  return (
    function(coord1) {
        var coord2 = [coord1[1], coord1[0]];
      return ol.coordinate.toStringXY(coord2,dgts);
  });
}

var mousePositionControl = new ol.control.MousePosition({
    coordinateFormat: myFormat(6), //DECIMALES
    //ol.coordinate.createStringXY(2),
    projection: 'EPSG:4326',
    className: 'custom-mouse-position',
    target: document.getElementById('mouse-position'),
    undefinedHTML: ' '
});
//------------------ FIN MOSTRAR LONGITUDES Y LATITUDES----------------//

//--------------------DISEÑO SELECCION CUADRADO-----------------------//
   var box = new ol.interaction.DragBox({
       condition: ol.events.condition.shiftKeyOnly,
       style: new ol.style.Style({
           stroke: new ol.style.Stroke({
               color: '#ffcc33',
               width: 2
           })
       })
   })

box.on('boxend', function() {
          var split = box.getGeometry().getCoordinates().toString().split(',');

          //   1                  4
          // (X, Y)          (X, Y)
```

```
            x1=split[0];   x4=split[6];
            y1=split[1];   y4=split[7];
            //   2                  3
            // (X, Y)          (X, Y)
            x2=split[2];   x3=split[4];
            y2=split[3];   y3=split[5];

            document.getElementById("longitudeorigin").value=x1;
            document.getElementById("latitudeorigin").value=y1;
            document.getElementById("longitudedestination").value=x3;
            document.getElementById("latitudedestination").value=y3;

            //alert(box.getGeometry().getCoordiantes().getArea());
        });
        //----------------------FIN SELECCION CUADRADO ------------------//

        //-----------------------CARGA DEL MAPA--------------------------//
    var map = new ol.Map({
        target: 'map',
        renderer: 'canvas',
        layers: layers, //CAMBIAR MAPA
        controls: ol.control.defaults({
            attributionOptions:
                   collapsible: false
            })
        }).extend([mousePositionControl]),
        view: new ol.View({
              projection: 'EPSG:4326',
            center: [0, 0],
            zoom: 2
        }),
        interactions: ol.interaction.defaults().extend([box])
    });
    //----------------------FIN CARGA DEL MAPA-------------------------//
    </script>
</html>
```

10.3.2 PÁGINA DE RESULTADOS

La segunda de las vistas se trata de la página de resultados, encargada de mostrar los resultados que se han procesado en el modelo en función de los datos recogidos en la vista de la página principal.

Se declaran en primer lugar las librerías y fuentes externas que se utilizan:

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>


<link          type="text/css"          rel="StyleSheet"          href="<c:url
value="/resources/css/form.css"/>"/>
<link    rel="stylesheet"    href="http://openlayers.org/en/v3.5.0/css/ol.css"
type="text/css">
<script                      src="http://openlayers.org/en/v3.5.0/build/ol.js"
type="text/javascript"></script>
```

Las librerías y fuentes externas se tratan de:

- La librería JSTL, se trata de un conjunto de librerías de etiquetas simples y estándares que encapsulan la funcionalidad principal para escribir páginas JSP, se encarga de proporcionar los tags para hacer uso de funciones como “c:url”.
- Las fuentes del estilo CSS desarrolladas para el proyecto y en concreto para esta vista, la cual se trata de “form.css”.
- Las librerías correspondientes al mapa grafico implementado mediante OpenLayers, las cuales se tratan en primer lugar de un documento de tipo CSS y un documento JavaScript.

Se declara mediante HTML una tabla para mostrar los datos. La tabla cuenta con las filas correspondientes a los atributos de las mediciones de los datos geoespaciales, y de tantas columnas como elementos se hayan procesados.

```
      <table id="testrow" style="display:none">
      <caption>RESULTADOS (TOTAL: ${listspatialpointfilter.size()})</caption>
        <thead>
                    <tr>
                       <th scope="col">FECHA</th>
                       <th scope="col">LATITUD</th>
```

```
                    <th scope="col">LONGITUDE</th>
                    <th scope="col">WIND SPEED</th>
                    <th scope="col">WIN DIR</th>
                    <th scope="col">RAIN RATE</th>
                  </tr>
      </thead>
      <tbody>
          <c:forEach var="lspf" items="${listspatialpointfilter}">
               <tr>
                    <td>${lspf.date}</td>
                    <td>${lspf.latitude}</td>
                    <td>${lspf.longitude}</td>
                    <td>${lspf.windSpeed}</td>
                    <td>${lspf.windDir}</td>
                    <td>${lspf.rainRate}</td>
               </tr><
           /c:forEach>
      </tbody>
      </table>
```

La tabla de datos se puede mostrar u ocultar en función de su uso para una visualización más estética, la función JavaScript que permite dicha función es la siguiente:

```
function hideShowHTMLItem(itemID, iState) {
      htmlItem = document.getElementById(itemID);
      if (htmlItem) {
            if (iState == 0) {  // hide
                  htmlItem.style.display = 'none';
             }
             else {  // show
                  htmlItem.style.display = '';
             }
      }
}
```

Se definen también una serie de elementos mediante HTML, como son una cabecera, que indica el título de la vista en el navegador Web.

```
<head>
    <title>GEOESPATIAL</title>
</head>
```

Se define también en el cuerpo de la estructura, un elemento "map", el cual será explicado más adelante en detalle.

```
<body>
      <div id="map" class="map"></div>
      <div class="ui-helper-right" id="mouse-position"></div>
```

Un pie de página, a modo decorativo y con posibles futuras funcionalidades que se implementa con estructura de listas HTML.

```
<footer>
  <ul>
    <li>
     <p class="home">Home</p>
     <a class="logo" href="#">UNIVERSIDAD DE EXTREMADURA<i>&copy;2015</i></a>
    </li>
    <li>
     <p class="services">Services</p>
       <ul>
           <li><a href="#">3D modeling</a></li>
           <li><a href="#">Web development</a></li>
           <li><a href="#">Mobile development</a></li>
           <li><a href="#">Web &amp; Print Design</a></li>
        </ul>
    </li>
    <li>
      <p class="reachus">Reach us</p>
       <ul>
           <li><a href="#">Email</a></li>
           <li><a href="#">Twitter</a></li>
           <li><a href="#">Facebook</a></li>
```

```
            <li> +34 123 456 789</li>
        </ul>
    </li>
    <li>
      <p class="clients">Users</p>
       <ul>
            <li><a href="#">Login Area</a></li>
            <li><a href="#">Support Center</a></li>
            <li><a href="#">FAQ</a></li>
      </ul>
   </li>
  </ul>
</footer>
```

Y sin duda, el elemento principal, el mapa. Al igual que en la vista principal, el mapa se desarrolla mediante OpenLayers y JavaScript para implementar su parte gráfica. OpenLayers se trata de una biblioteca de JavaScript de código abierto bajo una derivación de la licencia BSD para mostrar mapas interactivos en los navegadores Web. OpenLayers ofrece un API para acceder a diferentes fuentes de información cartográfica en la red. Su implementación no se sencilla, a continuación se detallan los procesos realizados.

En primer lugar, se realiza la inserción de un mapa básico. Este mapa básico se tratará del mapa sobre el que se irán añadiendo funcionalidades y atributos, añadiendo complejidad y permitiendo obtener un mapa grafico e interactivo que satisfaga nuestros requerimientos. La diferencia de este mapa respecto al desarrollado para la página principal es la sencillez con la ausencia de algunos atributos.

```
var map = new ol.Map({
    target: 'map',
    layers:[
          raster,
          featureLayerRed,
          featureLayerOrange,
          featureLayerYellow
   ],
```

```
   view: view,
   controls: ol.control.defaults({ }).extend([mousePositionControl]),
});
```

Para implementar este mapa básico se han declarado algunos atributos como:

- Target, se trata del "objetivo". Se declara "map", para implementar el mapa.
- Layers, se trata del mapa que se va a utilizar. Es necesario definirlos previamente. En este caso el mapa que se utiliza es "raster" y además se proporcionan una serie de elementos (vectores) encargados de dibujar los puntos sobre el mapa: featureLayerRed, featureLayerOrange y featureLayerYellow. Cada vector representa puntos rojos, naranjas o amarillos.
- Controls, se trata de controles adicionales que se implementan sobre el mapa:
  - Se define para esta vista, "mousePositionControl".
- View, se trata de la proyección que se utilizará, en este caso se ha aislado del mapa básico para una separación de las capas de los distintos elementos mejor.

A continuación se recogen las funcionalidades añadidas al mapa.

Se implementa en el mapa controles adicionales, como la función "MousePosition" que permite que al colocar el ratón sobre el mapa nos muestre los valores textuales de latitud y longitud (respectivamente) sobre la zona en la cual está situado el ratón en el mapa.

En dicho control adicional, además se indica la proyección del mapa, la cual se trata de "EPSG:4326".

```
var myFormat = function(dgts){
      return (
          function(coord1) {
               var coord2 = [coord1[1], coord1[0]];
               return ol.coordinate.toStringXY(coord2,dgts);
        });
}

var mousePositionControl = new ol.control.MousePosition(
      coordinateFormat: myFormat(6), //DECIMALES
```

```
      projection: 'EPSG:4326',
      className: 'custom-mouse-position',
      target: document.getElementById('mouse-position'),
      undefinedHTML: ' '
});
```

La vista del mapa se representa por separado como ya se ha indicado. La vista indica la proyección que se utilizará, el zoom con el que se mostrará el mapa y el punto centrado para la proyección. La imagen correspondiente al mapa estará centrada en el punto de longitud 0 y latitud 0 y un zoom de grado 3.

```
var view = new ol.View({
      center: [0, 0],
      zoom: 3
})
```

También se representa por otro lado el tipo de mapa, en este caso recibe el nombre de "raster" y se trata del mapa que se va a utilizar. En este caso se ha utilizado un mapa de origen satélite.

```
var raster = new ol.layer.Tile({ // MAPA SATELITE
        source: new ol.source.MapQuest({
            layer: 'sat'
        })
});
```

Para representar las distintas intensidades de los vientos de los datos geoespaciales consultados según sus valores respecto a la intensidad de la velocidad del viento, se declaran tres tipos de estilos:

- Puntos rojos, para las intensidades de vientos más fuertes
- Puntos naranjas, para las intensidades de vientos medias.
- Puntos amarillos, para las intensidades de vientos bajas.

```
        var styleRed = new ol.style.Style({
           image: new ol.style.Circle({
              radius: 1.4,
              fill: new ol.style.Fill({color: 'red'}),
           })
        });

        var styleOrange = new ol.style.Style({
           image: new ol.style.Circle({
              radius: 1.4,
              fill: new ol.style.Fill({color: 'orange'}),
           })
         });

        var styleYellow = new ol.style.Style({
            image: new ol.style.Circle({
              radius: 1.4,
              fill: new ol.style.Fill({color: 'yellow'}),
            })
        });
```

Y tres tipos de vectores correspondientes a los distintos estilos.

```
var featureLayerRed = new ol.layer.Vector(
      source: new ol.source.Vector(),
       style: styleRed
});

var featureLayerOrange = new ol.layer.Vector({
      source: new ol.source.Vector(),
       style: styleOrange
});

var featureLayerYellow = new ol.layer.Vector({
      source: new ol.source.Vector(),
       style: styleYellow
});
```

La carga de los vectores, se realiza utilizando las variables proporcionando desde el modelo, para ello se lista la información contenida en las variables haciendo uso de la sentencia “c:forEach”.

```
featureLayerRed.getSource().addFeatures([
	new ol.Feature( new ol.geom.MultiPoint([
		<c:forEach var="redIterator" items="${red}">[${redIterator.longitude},${redIterator.latitude}],
		</c:forEach>
	]).transform('EPSG:4326', 'EPSG:3857'))
]);

featureLayerOrange.getSource().addFeatures([
	new ol.Feature( new ol.geom.MultiPoint([
		<c:forEach var="orangeIterator" items="${orange}">[${orangeIterator.longitude},${orangeIterator.latitude}],</c:forEach>
	]).transform('EPSG:4326', 'EPSG:3857'))
]);

featureLayerYellow.getSource().addFeatures([
	new ol.Feature(new ol.geom.MultiPoint([
		<c:forEach var="yellowIterator" items="${yellow}">[${yellowIterator.longitude},${yellowIterator.latitude}],</c:forEach>
	]).transform('EPSG:4326', 'EPSG:3857'))
]);
```

A continuación se presenta la vista en su conjunto:

```
<!doctype html>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html lang="en">

<link type="text/css" rel="StyleSheet" href="<c:url value="/resources/css/form.css"/>"/>
<link rel="stylesheet" href="http://openlayers.org/en/v3.5.0/css/ol.css" type="text/css">
<script src="http://openlayers.org/en/v3.5.0/build/ol.js" type="text/javascript"></script>
```

```
<head>
    <title>GEOESPATIAL</title>
</head>
<body>
       <div id="map" class="map"></div>
       <div class="ui-helper-right" id="mouse-position"></div>

       <button                          class="btn"                          type="submit"
onclick="javascript:history.back()"><b>BACK!</b></button>
       <button    class="btn"    type="button"    onClick="hideShowHTMLItem('testrow',
0);">HIDE TABLE RESULT</button>
       <button    class="btn"    type="button"    onClick="hideShowHTMLItem('testrow',
1);">SHOW TABLE RESULT</button>

       <table id="testrow" style="display:none">
       <caption>RESULTADOS (TOTAL: ${listspatialpointfilter.size()})</caption>
        <thead>
                     <tr>
                        <th scope="col">FECHA</th>
                        <th scope="col">LATITUD</th>
                        <th scope="col">LONGITUDE</th>
                        <th scope="col">WIND SPEED</th>
                        <th scope="col">WIN DIR</th>
                        <th scope="col">RAIN RATE</th>
                      </tr>
              </thead>
              <tbody>
       <c:forEach var="lspf" items="${listspatialpointfilter}"><tr>
                    <td>${lspf.date}</td>
                    <td>${lspf.latitude}</td>
                    <td>${lspf.longitude}</td>
                    <td>${lspf.windSpeed}</td>
                    <td>${lspf.windDir}</td>
                    <td>${lspf.rainRate}</td>
               </tr></c:forEach>
       </tbody>
       </table>
 </body>
 <footer>
    <ul>
        <li>
            <p class="home">Home</p>
            <a class="logo" href="#">UNIVERSIDAD DE EXTREMADURA<i>&copy; 2015</i></a>
        </li>
```

```
        <li>
            <p class="services">Services</p>

            <ul>
                <li><a href="#">3D modeling</a></li>
                <li><a href="#">Web development</a></li>
                <li><a href="#">Mobile development</a></li>
                <li><a href="#">Web &amp;amp; Print Design</a></li>
            </ul>
        </li>
        <li>
            <p class="reachus">Reach us</p>

            <ul>
                <li><a href="#">Email</a></li>
                <li><a href="#">Twitter</a></li>
                <li><a href="#">Facebook</a></li>
                <li>555-123456789</li>
            </ul>
        </li>
        <li>
            <p class="clients">Users</p>

            <ul>
                <li><a href="#">Login Area</a></li>
                <li><a href="#">Support Center</a></li>
                <li><a href="#">FAQ</a></li>
            </ul>
        </li>
    </ul>

</footer>


    <script type="text/javascript">
          //-----------MOSTRAR LONGITUDES Y LATITUDES----------------//
          var myFormat = function(dgts){
            return (
              function(coord1) {
                  var coord2 = [coord1[1], coord1[0]];
                return ol.coordinate.toStringXY(coord2,dgts);
            });
          }
```

```
 var mousePositionControl = new ol.control.MousePosition({
     coordinateFormat: myFormat(6), //DECIMALES
     projection: 'EPSG:4326',
     className: 'custom-mouse-position',
     target: document.getElementById('mouse-position'),
     undefinedHTML: ' '
 });
 //---------- FIN MOSTRAR LONGITUDES Y LATITUDES---------------//

 //-------- CARGA DE VECTORES: ROJO, NARANJA Y AMARILLO --------//
var styleRed = new ol.style.Style({
   image: new ol.style.Circle({
      radius: 1.4,
      fill: new ol.style.Fill({color: 'red'}),
   })
});

var styleOrange = new ol.style.Style({
   image: new ol.style.Circle({
      radius: 1.4,
      fill: new ol.style.Fill({color: 'orange'}),
   })
});

var styleYellow = new ol.style.Style({
    image: new ol.style.Circle({
      radius: 1.4,
      fill: new ol.style.Fill({color: 'yellow'}),
    })
});

    var view = new ol.View({
          center: [0, 0],
          zoom: 3
     })

    var featureLayerRed = new ol.layer.Vector({
        source: new ol.source.Vector(),
           style: styleRed
    });

    var featureLayerOrange = new ol.layer.Vector({
```

```
                source: new ol.source.Vector(),
                    style: styleOrange
            });

            var featureLayerYellow = new ol.layer.Vector({
                source: new ol.source.Vector(),
                    style: styleYellow
            });

        var raster = new ol.layer.Tile({ // MAPA SATELITE
            source: new ol.source.MapQuest({
                layer: 'sat'
            })
        });
        //-----FIN CARGA DE VECTORES: ROJO, NARANJA Y AMARILLO------//

        //-------------------CARGA DEL MAPA-----------------------//
            var map = new ol.Map({
                target: 'map',
                layers:     [raster,     featureLayerRed,     featureLayerOrange,
featureLayerYellow],
                view: view,
            controls: ol.control.defaults({ }).extend([mousePositionControl]),
                });
            //------------------FIN CARGA DEL MAPA---------------------//

            //-----------------CARGA DE VECTORES-----------------------//
            featureLayerRed.getSource().addFeatures([
      new     ol.Feature(new     ol.geom.MultiPoint([<c:forEach     var="redIterator"
items="${red}">[${redIterator.longitude},${redIterator.latitude}],</c:forEach>]).tran
sform('EPSG:4326', 'EPSG:3857')) ]);

            featureLayerOrange.getSource().addFeatures([new          ol.Feature(new
ol.geom.MultiPoint([<c:forEach                                    var="orangeIterator"
items="${orange}">[${orangeIterator.longitude},${orangeIterator.latitude}],</c:forEac
h>]).transform('EPSG:4326', 'EPSG:3857')) ]);

            featureLayerYellow.getSource().addFeatures([new          ol.Feature(new
ol.geom.MultiPoint([<c:forEach                                    var="yellowIterator"
items="${yellow}">[${yellowIterator.longitude},${yellowIterator.latitude}],</c:forEac
h>]).transform('EPSG:4326', 'EPSG:3857')) ]);
            //---------------FIN CARGA DE VECTORES----------------------//

            //---------------MOSTRAR/OCULTAR TABLA--------------------//
            function hideShowHTMLItem(itemID, iState) {
                htmlItem = document.getElementById(itemID);
                if (htmlItem) {
```

```
                    if (iState == 0) {  // hide
                      htmlItem.style.display = 'none';
                    }
                    else {  // show
                      htmlItem.style.display = '';
                    }
                  }
                }
              //----------------FIN MOSTRAR/OCULTAR TABLA-----------------//
              </script>
</html>
```

10.4 FICHEROS DE CONFIGURACIÓN

Spring proporciona una serie de ficheros de configuración en formato XML para la gestión del framework y de las tecnologías que conforman el proyecto, así como para la interacción entre los elementos que conforman la estructura del proyecto, en este caso una estructura Modelo Vista Controlador. Estos ficheros son principalmente tres.

El fichero POM.XML (Project Object Model), se trata de un fichero gestionado por el software Maven utilizado para la adminsitraci y construcción de proyectos Java. En él fichero se declaran las tecnologías y bibliotecas que se desean integrar en el proyecto y Maven se encarga de gestionar su descarga, su integración y de las depedencias que estas puedan tener. A continuación se acompaña el fichero POM.XML utilizado en este proyecto:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
       http://maven.apache.org/maven-v4_0_0.xsd">
       <modelVersion>4.0.0</modelVersion>
       <groupId>DavidSanjuanLopez</groupId>
       <artifactId>GeoespatialAplication</artifactId>
       <packaging>war</packaging>
       <version>1.0-SNAPSHOT</version>
       <name>Geoespatial MongoDB Maven Webapp</name>
       <url>http://maven.apache.org</url>
       <dependencies>
              <dependency>
                     <groupId>org.springframework</groupId>
                     <artifactId>spring-orm</artifactId>
                     <version>4.1.5.RELEASE</version>
              </dependency>
              <dependency>
                     <groupId>org.springframework</groupId>
                     <artifactId>spring-webmvc</artifactId>
                     <version>4.1.5.RELEASE</version>
              </dependency>
              <dependency>
```

```
                <groupId>org.springframework.data</groupId>
            <artifactId>spring-data-mongodb</artifactId>
            <version>1.7.0.M1</version>
      </dependency>
            <dependency>
                  <groupId>org.mongodb</groupId>
                  <artifactId>mongo-java-driver</artifactId>
                  <version>3.0.0-SNAPSHOT</version>
            </dependency>
            <dependency>
                  <groupId>jstl</groupId>
                  <artifactId>jstl</artifactId>
                  <version>1.2</version>
            </dependency>
            <dependency>
                  <groupId>junit</groupId>
                  <artifactId>junit</artifactId>
                  <version>3.8.1</version>
                  <scope>test</scope>
            </dependency>
          <dependency>
              <groupId>org.springframework.integration</groupId>
              <artifactId>spring-integration-core</artifactId>
              <version>4.1.2.RELEASE</version>
          </dependency>
          <dependency>
                  <groupId>net.sf.opencsv</groupId>
                  <artifactId>opencsv</artifactId>
                  <version>2.0</version>
            </dependency>
          <dependency>
                      <groupId>commons-net</groupId>
                      <artifactId>commons-net</artifactId>
                      <version>2.0</version>
              </dependency>
              <dependency>
                      <groupId>commons-io</groupId>
                      <artifactId>commons-io</artifactId>
                      <version>2.4</version>
              </dependency>
      </dependencies>
      <repositories>
        <repository>
            <id>sonatype-snapshot</id>

<url>https://oss.sonatype.org/content/repositories/snapshots/</url>
        </repository>
        <repository>
            <id>spring-milestones</id>
            <name>Spring Milestones</name>
            <url>http://repo.spring.io/milestone</url>
            <snapshots>
            <enabled>false</enabled>
            </snapshots>
      </repository>
    </repositories>
      <build>
            <plugins>
                  <plugin>
                        <artifactId>maven-compiler-plugin</artifactId>
                        <version>2.3.2</version>
```

```
                        <configuration>
                              <source>1.6</source>
                              <target>1.6</target>
                        </configuration>
                  </plugin>
            </plugins>
            <finalName>GeoespatialAplication</finalName>
      </build>
</project>
```

Los otros dos ficheros de configuración restantes se tratan de web.XML y dispatcher-servlet.XML. El primero de ellos se encarga de indicar que servlet, clase encargada de ofrecer contenido web, escuchará las peticiones Http del usuario y su configuración es la siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" version="2.5">
  <display-name>Geoespatial Web Application with Spring and MongoDB</display-
name>
  <servlet>
    <servlet-name>dispatcher</servlet-name>
    <servlet-
class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>dispatcher</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>
  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/dispatcher-servlet.xml</param-value>
  </context-param>
  <listener>
    <listener-
class>org.springframework.web.context.ContextLoaderListener</listener-class>
  </listener>
</web-app>
```

El último de los ficheros, dispatcher-servlet.XML, indica desde que controlador se realiza la atención de las peticiones que realice el usuario, de determinar los parámetros del host al que realizar la conexión a la base de datos de MongoDB (en este caso localhost)

y de indicar donde se encuentran en el proyecto las vistas a ser retornadas por el controlador.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:mvc="http://www.springframework.org/schema/mvc"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
	xmlns:p="http://www.springframework.org/schema/p"
	xsi:schemaLocation="
	 http://www.springframework.org/schema/beans
	 http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
	 http://www.springframework.org/schema/mvc
	   http://www.springframework.org/schema/mvc/spring-mvc.xsd
	   http://www.springframework.org/schema/context
	   http://www.springframework.org/schema/context/spring-context-
3.2.xsd">

	<context:component-scan base-package="com.geospatial" />
	<!-- HE AÑADIDO XMLN:MONGO -->
	<!-- Factory bean that creates the Mongo instance -->
	<bean id="mongo"
class="org.springframework.data.mongodb.core.MongoFactoryBean">
		<property name="host" value="localhost" />
	</bean>

	<!-- MongoTemplate for connecting and quering the documents in the
database -->
	<bean id="mongoTemplate"
class="org.springframework.data.mongodb.core.MongoTemplate">
		<constructor-arg name="mongo" ref="mongo" />
		<constructor-arg name="databaseName" value="test" />
	</bean>

	<!-- Use this post processor to translate any MongoExceptions thrown
in
		@Repository annotated classes -->
	<bean

	class="org.springframework.dao.annotation.PersistenceExceptionTranslat
ionPostProcessor" />

		<bean
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
		  <!-- Example: a logical view name of 'showMessage' is mapped
to '/WEB-INF/jsp/showMessage.jsp' -->
		  <property name="prefix" value="/WEB-INF/jsp/"/>
		  <property name="suffix" value=".jsp"/>
	</bean>

	<mvc:annotation-driven/>
	<mvc:resources mapping="/resources/**" location="resources/"/>

</beans>
```

10.5 IMPORTACIÓN EN ENTORNO DE DESARROLLO

Como IDE de desarrollo se ha utilizado Spring Tool Suite, una herramienta de desarrollo basada en Eclipse que integra software y funcionalidades de forma nativa para propiciar y acelerar desarrollos con Spring Framework.

Para importar el proyecto y poder realizar modificaciones o ampliaciones, desde la consola de Spring Tool Suite seleccionando el menú *"File"* en la barra de navegación superior y posteriormente *"Import"*, tras lo cual se desplegará una pantalla para seleccionar el tipo de ficheros fuentes a importar

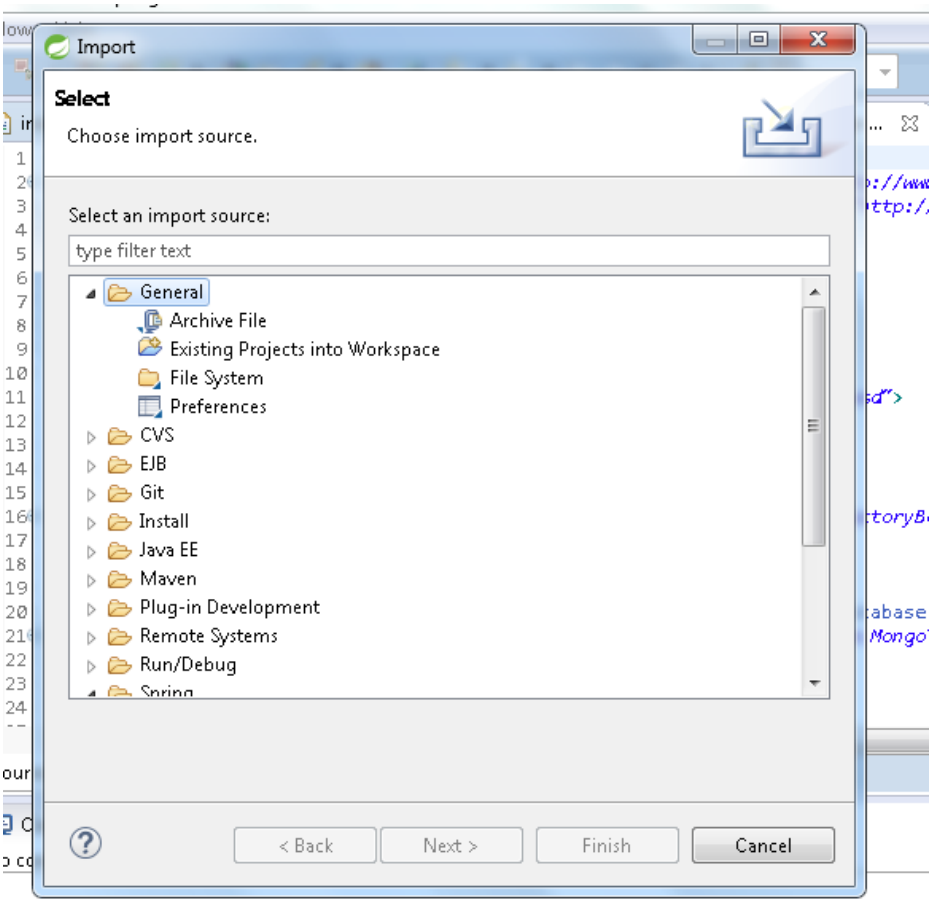


Ilustración 36: Pantalla de importación de proyectos de Spring Tool Suite

Seleccionamos *"General"* y posteriormente *"Existing Projects into Workspace"*, tras lo cual solo nos quedará indicar la ruta en la que se encuentra el directorio con los archivos fuentes y aceptar la importación. La importación se realizará de forma automática.

Si tras su importación, el proyecto contuviera errores se recomienda realizar una actualización de las dependencias, bibliotecas y configuraciones. Para ello se hace clic derecho con el ratón en el proyecto importado, se selecciona "*Maven*" y posteriormente "*Update Project*".

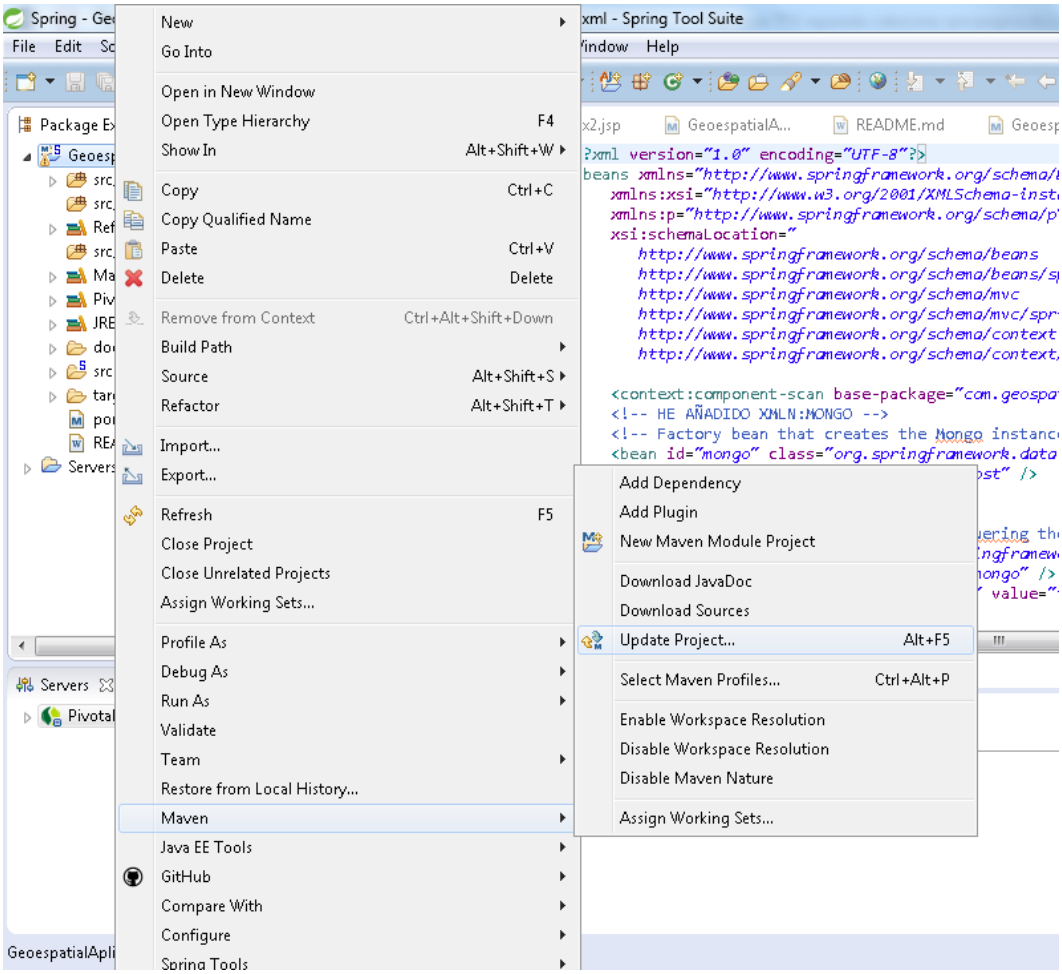
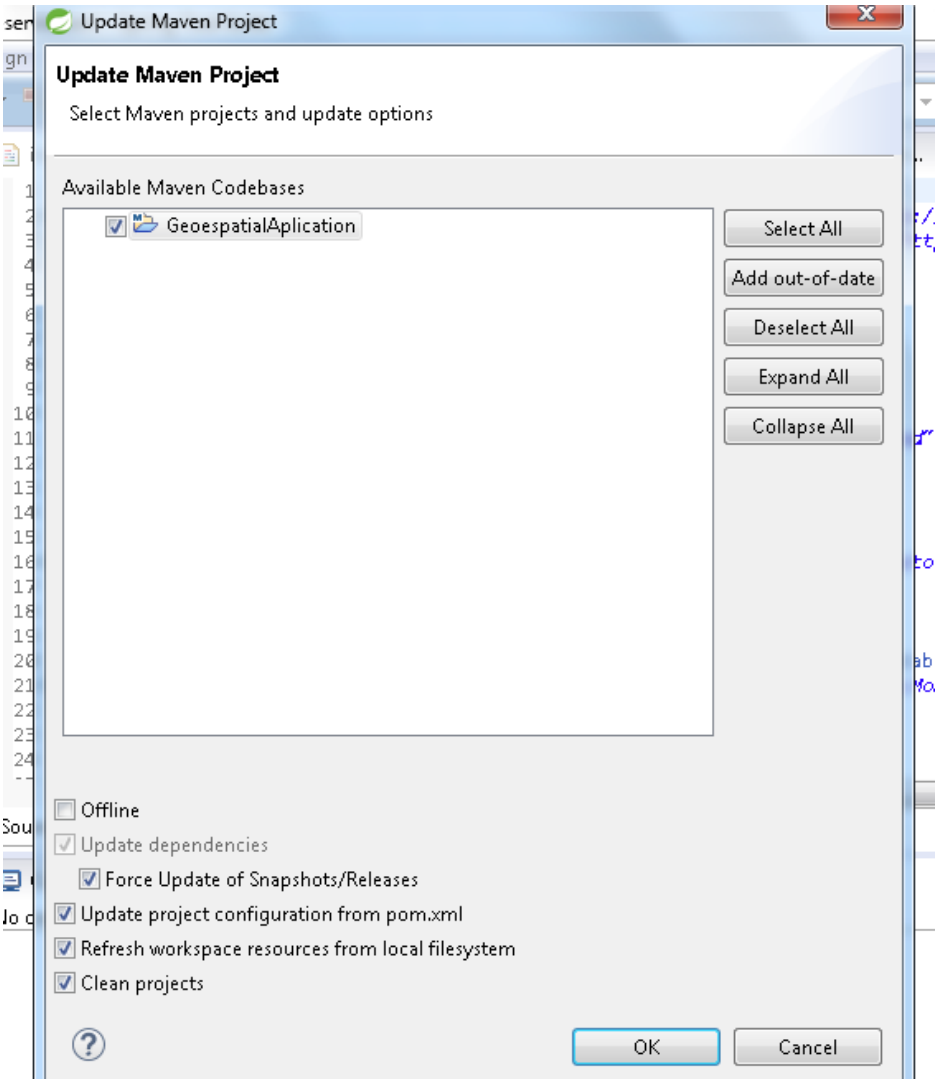


Ilustración 37: Método para actualizar el proyecto

Realizado este paso se nos desplegara una nueva venta, similar a la que aparece en la siguiente figura:



En esta ventana conviene seleccionar la opción “Force Update of Snapshots/Releases” para proporcionar una actualización completa del proyecto. Por defecto aparecerá seleccionado el proyecto con el que estamos trabajando. Para finalizar solo debemos hacer clic sobre “Ok” y se habrá actualizado, solucionando así la mayoría de los errores fruto de dependencias, librerías faltantes o incongruencias en la configuración del proyecto o IDE de desarrollo de Spring Tool Suite.