Tabla de contenido

[1. INTRODUCCIÓN 3](#_Toc488538048)

[1.1. MOTIVACIÓN 3](#_Toc488538049)

[1.2. OBJETIVOS 3](#_Toc488538050)

[1.3. CONTENIDO DE LA MEMORIA 4](#_Toc488538051)

[2. ESTADO DEL ARTE 4](#_Toc488538052)

[3. PROYECTO 4](#_Toc488538053)

[3.1. INTRODUCCIÓN 4](#_Toc488538054)

[3.2. REQUISITOS SOLUCIÓN PLANTEADA 5](#_Toc488538055)

[3.3. PLAN DEL PROYECTO 7](#_Toc488538056)

[4. SOLUCIÓN. 8](#_Toc488538057)

[4.1. TECNOLOGÍAS ESCOGIDAS 8](#_Toc488538058)

[4.1.1. FUENTE DE DATOS: TWITTER 8](#_Toc488538059)

[4.1.2. PROCESADO DE DATOS: PYTHON 10](#_Toc488538060)

[4.1.3. ALMACENAMIENTO DE DATOS: POSTGIS 10](#_Toc488538061)

[4.1.4. GENERACION Y SERVIDOR DE MAPAS: GEOSERVER 11](#_Toc488538062)

[4.1.5. VISUALIZACION DE MAPAS: WEB + OPENLAYERS 11](#_Toc488538063)

[4.2. DISEÑO. 12](#_Toc488538064)

[4.3. IMPLEMENTACIÓN E INSTALACIÓN 13](#_Toc488538065)

[4.3.1. FUENTE DE DATOS 13](#_Toc488538066)

[4.3.2. PROCESADO DE DATOS. 13](#_Toc488538067)

[4.3.3. ALMACENAMIENTO DE DATOS. 15](#_Toc488538068)

[4.3.4. GENERACIÓN DE MAPAS 21](#_Toc488538069)

[4.3.5. VISUALIZACIÓN DE LOS MAPAS. 22](#_Toc488538070)

[4.4. PUESTA EN MARCHA. 23](#_Toc488538071)

[5. CONCLUSIONES 25](#_Toc488538072)

[6. CASOS DE USO 25](#_Toc488538073)

[7. TRABAJO FUTURO 25](#_Toc488538074)

[ANEXO 1. ENLACES DE INTERES 25](#_Toc488538075)

[ANEXO 2. REPOSITORIO DE CÓDIGO 25](#_Toc488538076)

# INTRODUCCIÓN

## MOTIVACIÓN

El presente proyecto tiene como objetivo integrar técnicas y tecnologías disponibles para el tratamiento, almacenamiento, procesado y visualización de datos para explotar el potencial de la información geolocalizada.

El fenómeno de la geolocalización es uno de los más atractivos de los últimos tiempos.

El hecho de que el usuario pueda conectarse a internet desde cualquier sitio, no únicamente desde su casa y que este perfectamente ubicado, ofreciendo al mismo tiempo datos y referencias sobre sus movimientos representa todo un reto.

Con los mapas que se generan con la información geolocalizada se pueden conocer datos de negocio, patrones de comportamiento, cifras, tendencias sociales, que permiten la toma de decisiones de forma más rápida y efectiva.

La información geolocalizada permite a las empresas entender el entorno en relación a sus puntos de venta y consumidores, o permite a un partido político ver en que zonas tiene más apoyos o donde necesita hacer más campaña.

Por lo tanto, el presente proyecto va a analizar tecnologías que permitan de un vistazo rápido saber donde y cuando se está hablando sobre un tema. Es decir, visualizar información temporal y geográficamente localizada.

## OBJETIVOS

Los objetivos del presente proyecto son los siguientes:

* Analizar tecnologías disponibles para el tratamiento, almacenamiento, procesado y visualización de datos geolocalizados.
* Utilizar en la medida de lo posible tecnologías opensource para hacer una solución económicamente aceptable.
* Montar un prototipo que permita ver las bondades de la solución propuesta.
* Analizar los posibles usos y trabajos futuros de la solución propuesta.

## CONTENIDO DE LA MEMORIA

Tras esta pequeña introducción la memoria está dividida en cinco grandes bloques.

Primero analizaremos el estado del arte, posteriormente explicaremos de forma detallada el plan del proyecto para a continuación describir la solución adoptada. En el apartado de la solución se describirá con detalle el diseño y la implementación adoptados, así como las razones para tomar ciertas decisiones de diseño.

En los dos últimos apartados se detallarán los casos de uso y las conclusiones y trabaja futuro.

# ESTADO DEL ARTE

# PROYECTO

## INTRODUCCIÓN

Teniendo en cuenta que el objetivo del proyecto es montar un prototipo que integre tecnologías disponibles que permitan la visualización de información geolocalizada nuestro sistema tendrá cinco bloques claramente diferenciados.

El primer bloque funcional es la fuente de datos y las técnicas y tecnologías para obtener los datos de dicha fuente de datos.

A continuación, tenemos el bloque de procesado de datos que permite la extracción de características de los datos obtenidos de la fuente, así como un procesado posterior para dejar los datos en un estado óptimo para su almacenamiento y uso posterior.

Después tenemos el bloque de almacenamiento de datos que comprende las tecnologías utilizadas para el almacenamiento de datos en bruto, así como las técnicas y procesos para realizar las agregaciones de datos que correspondan para una mejor visualización.

La fase de presentación de los datos comprende todas las tecnologías y técnicas utilizadas para generar mapas a partir de los datos almacenados. Es decir, las tecnologías que permiten generar mapas a partir de datos geolocalizados y aplicar estilos a estos mapas.

Por último, está la parte de visualización que permite la representación de estos datos, así como introducir parametrización para modificar la visualización de los mismos. Por ejemplo, para filtra la fecha y la zona sobre la que se quiere generar el mapa o la granularidad temporal con la que se quieren agregar los datos.

En la figura siguiente se puede ver un diagrama de bloques del sistema.

Figura . DIAGRAMA DE BLOQUES

## REQUISITOS SOLUCIÓN PLANTEADA

Tras un exhaustivo análisis del problema se marca como objetivo del proyecto que la solución planteada cumpla los siguientes requisitos:

* Sea una solución sencilla. Este requisito no lo plantea el problema en si ya que el problema podría resolverse con una solución mucho más compleja, pero nos marcamos como objetivo principal del proyecto que la solución sea lo suficientemente sencilla para que casi cualquiera pueda montarse un sistema para comprobar las bondades de un proyecto que te permita ver información geolocalizada.
* Sea una solución escalable. Aunque la solución inicial planteada sea muy sencilla la elección de tecnologías se hace siempre pensando que el sistema tiene que ser fácilmente escalable para dar soporte a proyectos mucho más complejos, así la solución planteada se puede utilizar como punto de partida y luego se puede escalar a soluciones mucho más complejas.
* Sea una solución económica. Este es también un requisito personal del proyecto y no del problema. El problema se podría resolver con tecnologías de pago, pero como se piensa que las tecnologías opensource disponibles nos pueden servir para solucionar el problema se decide utilizar herramientas opensource en lugar de herramientas de pago. Además, el hecho de utilizar herramientas opensource nos permite contar con toda la ayuda de la comunidad online que tienen detrás.
* Sea en la medida de lo posible multiplataforma. No queremos atarnos a ningún sistema operativo en concreto por lo que la elección de tecnologías se hace también intentando escoger una arquitectura que se pueda utilizar sobre cualquier plataforma y el resultado final, es decir la web de visualización de mapas pueda visualizarse desde cualquier dispositivo.

Además de los requisitos del sistema global cada uno de los bloques del sistema tiene unos requisitos propios.

La fuente de datos tiene que ser escogida teniendo en cuenta que tiene que ofrecer una forma sencilla y rápida de obtener grandes cantidades de datos geolocalizados.

El procesado de datos tiene que extraer las características necesarias de la fuente de datos, así como hacer las transformaciones necesarias para adecuar los datos al almacenamiento y uso posterior de la manera más rápida y eficiente posible.

El bloque de almacenamiento de datos tiene que permitir la ingestión rápida de datos y estar diseñado para que el acceso a los datos para la generación de los mapas sea lo más rápido posible.

El bloque de generación de mapas tiene que ofrecer un servidor de mapas que genere mapas a partir de nuestros datos aplicándole estilos diseñados por nosotros. Además, tiene que presentar una buena integración con el bloque de almacenamiento de datos para obtener rápidamente los datos para generar los mapas. El protocolo ofrecido para acceder a los mapas generados tiene que ser sencillo y estándar para que la comunicación con el bloque de visualización sea sencilla y la elección de la tecnología a utilizar en el bloque de visualización sea independiente del servidor de mapas utilizado en el bloque de generación de mapas.

El bloque de visualización tiene que utilizar alguna tecnología que permita representar de manera conjunta los mapas generados por nuestro bloque de generación de mapas con mapas proporcionados por un servidor de mapas opensource como openstreetmaps o bing que nos den información del terreno para poder identificar claramente donde están situados nuestros datos.

El bloque de visualización también tiene que facilitar controles al usuario para que de una manera sencilla pueda configurar filtros temporales y espaciales para el mapa generado, así como aplicar estilos distintos al mapa.

## PLAN DEL PROYECTO

El proyecto se estructuro en seis fases que serán descritas a continuación:

* FASE 1. Investigación y elección entre tecnologías disponibles.

En esta primera fase se hizo una investigación preliminar de las tecnologías disponibles para cada uno de los bloques del proyecto y se escogió una de entre todas para cada uno de los bloques.

Había que escoger:

* Una fuente de datos geolocalizados
* Un lenguaje programación/tecnología para el procesado de datos
* Un sistema de almacenamiento de datos
* Un servidor de mapas
* Tecnología que permita la visualización de mapas procedentes de nuestro servidor de mapas y un servidor de mapas del terreno como openstreetmaps o bing.
* FASE 2. Investigación de cada una de las tecnologías escogidas.

Una vez tomada la decisión hay que empezar la investigación de la tecnología para ver cómo utilizarla para la solución del problema.

* FASE 3. Instalación, implementación e integración del sistema.

Para cada uno de los bloques instalación de la solución, implementación e integración con los otros bloques para montar el prototipo.

* FASE 4. Test del sistema con pocos datos
* FASE 5. Test del sistema con más datos
* FASE 6. Conclusiones y trabajo futuro.

Una vez el sistema este montado y probado llega el momento de sacar conclusiones e identificar posibles trabajos futuros sobre el tema, así como posibles limitaciones del sistema que requieran una mejora.

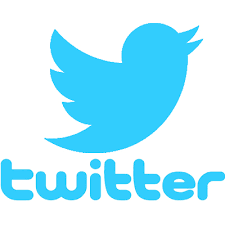
# SOLUCIÓN.

## TECNOLOGÍAS ESCOGIDAS

La elección de las tecnologías de cada bloque se hizo en la medida de lo posible para que aunque se cambie la tecnología de uno de los bloques el resto del sistema fuera valido con unas pequeñas modificaciones. Esto es posible para todos los bloques menos para el de procesado de datos ya que teniendo en cuenta que su principal función es la de extraer características de los datos procedentes de la fuente y transformar estos datos para su posterior almacenamiento y uso está muy ligado a la fuente de datos que se va a utilizar, así como a la tecnología de almacenamiento y al uso posterior de los datos.

Aun así, tal y como está diseñado si se cambia la fuente de datos, adaptando el bloque de procesado de datos todo el sistema seguiría funcionando. El bloque de almacenamiento de datos se puede cambiar por cualquiera con el que se pueda conectar el bloque de generación de mapas. El bloque de generación de mapas se puede cambiar por cualquiera que tenga un protocolo estándar de acceso a los mapas ya que el bloque de visualización se ha hecho utilizando herramientas que utilizan protocolos estándar.

### FUENTE DE DATOS: TWITTER

Twitter es un servicio de microblogging que permite a los usuarios enviar y publicar mensajes de texto de un máximo de 140 caracteres.

En esos 140 caracteres se pueden contar desde qué se está haciendo, interactuar con otros usuarios mediante respuestas y mensajes privados, anunciar cosas, promocionarse, hacer o mantener amistades y redes de amigos, encontrar trabajo, y otros. Estos mensajes que publica el usuario se muestran en su página de perfil y son también enviados de forma inmediata a otros usuarios que han elegido la opción de recibirlas. El usuario que los publica puede restringir el envío de estos mensajes solo a miembros de su círculo de amigos, o permitir su acceso a todos los usuarios de manera pública, que es la opción por defecto.

Los usos y temas tratados es twitter son de lo más diversos. Por ejemplo, fue usada para organizar protestas, a veces referidas como “twitter revolutions”, entre las que se encuentran la revolución egipcia del 2011, la revolución tunecina y las protestas electorales en Iran del 2009. También se utiliza activamente durante los periodos electorales o para temas más banales como comentar péliculas o eventos televisivos en tiempo real.

Por eso la hace la fuente de datos perfecta para montar un sistema para visualizar datos geolocalizados.

La fuente de datos escogida es twitter porque la REST API facilitada por twitter permite obtener tweets de forma relativamente fácil y además permite filtrar los tweets para obtener únicamente los relativos a un tema en concreto.

Cada uno de los tweets obtenidos tiene un formato de JSON y puede tener la siguiente información de geolocalización. Digo puede tener y no tiene porque la presencia o no de esta información depende de si el usuario tiene habilitada o no la geolocalización en su perfil.

El campo coordinates del JSON contiene las coordenadas de la posición del usuario en formato geoJSON (primero la longitud y después la latitud). Esta es la única información que se puede extraer del tweet que tiene la posición concreta donde se encuentra el usuario, pero sólo estará disponible si el usuario tiene habilitada la geolocalización en su perfil.

Además, el tweet puede tener el campo Place que estará relleno si el usuario ha decidido asociar su tweet a un lugar, por ejemplo, un restaurante, un sitio turístico, etc. Este campo está formado por un objeto JSON de tipo places que contiene información del país, la ciudad y las coordenadas del bounding box donde se encuentra el lugar etiquetado. Esta información se podría utilizar en caso de que no estuviera relleno el campo coordinates para determinar la situación del usuario, pero hay que tener en cuenta que el hecho de que este usuario haya decidido etiquetar este lugar no quiere decir que este físicamente allí, por lo tanto, la geolocalización no es 100% segura.

El tweet también tiene un campo user con información del usuario y en el objeto JSON user hay varios campos que nos pueden dar información de la geolocalización del usuario (location y time\_zone). Pero tenemos el mismo problema de antes, esta información la rellena el usuario y no indica para nada la posición exacta de donde se encuentra podemos extraer la ciudad o el país, pero siempre teniendo en cuenta que puede que ni siquiera eso sea real, ya que un usuario puede rellenar que está en Miami cuando realmente está en Valladolid.

En nuestra solución decidimos utilizar únicamente el campo coordinates del tweets que es el único que nos da la geolocalización exacta del usuario pero en trabajos futuros se podría trabajar para utilizar también los otros campos.

### PROCESADO DE DATOS: PYTHON

Para el módulo de procesado de datos se utilizó un programa en Python.

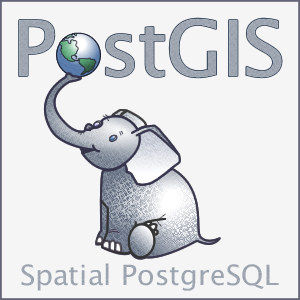
Python es un lenguaje de programación interpretada cuya filosofía hace hincapié en una sintaxis que favorezca un código legible.

Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, usa tipado dinámico y es multiplataforma.

Es administrado por la Python Software Foundation y posee una licencia de código abierto.

Se escogió Python porque cumple los requisitos de sencillez y facilidad de uso, así como lo de ser código abierto. Además hay un amplia comunidad online que permite encontrar ejemplos que hacen la implementación de un programa mucho más sencilla.

### ALMACENAMIENTO DE DATOS: POSTGIS



PostGIS convierte al sistema de administración de bases de datos PostgreSQL en una base de datos espacial mediante la adición de tres características: tipos de datos espaciales, índices espaciales y funciones que operan sobre ellos. Debido a que está construido sobre PostgreSQL, PostGIS hereda automáticamente las características de las bases de datos empresariales, así como los estándares abiertos que implementan un Sistema de Información Geográfica dentro del motor de base de datos.

PostGIS es un módulo que añade soporte de objetos geográficos a la base de datos objeto-relacional PostgreSQL, convirtiéndola en una base de datos espacial para su utilización en Sistemas de Información Geográfica. Se publica bajo la Licencia Pública General de GNU.

PostGIS es a día de hoy un proyecto maduro que ha demostrado su eficiencia versión a versión. En relación a otros productos, PostGIS ha demostrado ser muy superior a la extensión geográfica de MySQL y a juicio de muchos es muy similar a la versión geográfica de la base de datos ORACLE.

Para el almacenamiento de datos se decidió utilizar POSTGIS principalmente porque es software libre, porque es un sistema de almacenamiento de datos creado para almacenar y trabajar con datos geolocalizados y porque se integra muy bien con el servidor de mapas que íbamos a utilizar.

### GENERACION Y SERVIDOR DE MAPAS: GEOSERVER



Geoserver es un servidor de código abierto escrito en Java que permite a los usuarios compartir y editar datos geoespaciales. Diseñado para la interoperabilidad publica datos de las principales fuentes de datos espaciales usando estándares abiertos. Geoserver ha evoluacionado hasta llegar a ser un método sencilla de conectar información existente a globos virtuales tales como Google Earth y NASA World Wind (veáse asi como mapas basados en web como OpenLayers, Google Maps y Bing Maps). Geoserver sirve de implementación de referencia del estándar Open Geospatial Consortium Web Feature Service y también implementa las especificaciones Web Map Service y Web Coverage Service.

Para la generación y servicio de mapas se utilizó geoserver porque también cumple los requisitos de ser sencillo de usar, ser software libre y suministrar los mapas con un protocolo estándar que permite la conexión con múltiples tecnologías.

### VISUALIZACION DE MAPAS: WEB + OPENLAYERS

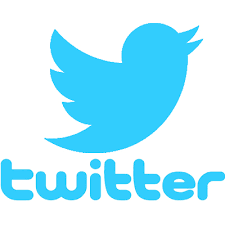


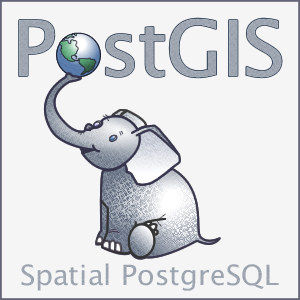
OpenLayers es una biblioteca de JavaScript de código abierto bajo una derivación de la licencia BSD para mostrar mapas interactivos en los navegadores web. OpenLayers ofrece un API para acceder a diferentes fuentes de información cartográfica en la red: Web Map Services, Mapas comerciales (tipo Google Maps, Bing, Yahoo), Web Features Services, distintos formatos vectoriales, mapas de OpenStreetMap, etc.

Para la visualización de los mapas se creó una web muy sencilla que utilizaba openlayers para mostrar los mapas servidor por el servidor de mapas. Se escogio openlayers porque es sencillo de usar, es opensource y permite conectarse a prácticamente cualquier servidor de mapas de los que hay actualmente utilizando el protocolo estándar WMS.

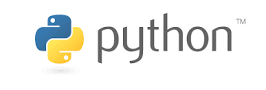
## DISEÑO.

Tras la elección de las tecnologías a utilizar el diagrama de bloques queda como muestra la siguiente figura.





PROCESADO DATOS



## IMPLEMENTACIÓN E INSTALACIÓN

### FUENTE DE DATOS

Como el objetivo de este proyecto no era profundizar en el modo de obtener tweets con el REST API de twitter se partió de colecciones de tweets ya disponibles. Para la primera prueba con pocos datos se utilizó una colección de tweets en español que se utilizó en una práctica para detectar el idioma y para la prueba con más carga de tweets se partió de una colección de tweets relacionados con la entrega de los Premios Oscars 2017.

Por lo tanto, la fuente de datos serán ficheros que contienen tweets en formato json.

### PROCESADO DE DATOS.

El procesado de datos implementado es muy sencillo. Se trata de un programa en Python que para cada tweet extrae los campos considerados de interés, realiza algunas transformaciones en dichos campos y almacena el resultado en la base de datos.

Para la implementación del mismo se utilizan ciertos módulos y librerías de Python o de terceros que tendrán que ser instaladas para la ejecución. Estas librerías se detallan a continuación:

* DateUtil. <https://dateutil.readthedocs.io/en/stable/>. Este módulo proporciona un extensión muy potente del módulo estándar datetime disponible en Python. Lo utilizaremos para parsear la cadena de texto que contiene la fecha de creación en formato UTC.
* JSON. Módulo de la distribución de Python para trabajar con texto en formato JSON.
* GDAL. <https://pypi.python.org/pypi/GDAL>. Esta librería se utiliza para trabajar con datos geoposicionados. En nuestro caso en concreto lo utilizaremos para hacer proyecciones de un sistema EPSG a otro.
* PSYCOPG2. <http://initd.org/psycopg/docs/>. Este módulo es el adaptador más conocido para conectarse a una base de datos postgresql con un programa Python y será el módulo que utilizaremos para conectarnos a nuestra base de datos postgis.

Los campos que se intentan extraer de cada tweet son:

* ID del tweet
* Texto del tweet
* Fecha de creación del tweet
* Contador de veces que ha sido marcado como favorito
* Contador de veces que ha sido retweeteado
* Id del tweet al que se responde si el tweet es respuesta a otro
* Id del tweet que se está citando si el tweet está citando a otro.
* Id del tweet que se está retweeteando si el tweet es un retweet.
* Id del usuario que escribe el tweet.
* Coordenadas de la posición geolocalizada del lugar donde se escribió el tweet.

Sobre los únicos campos que se va a hacer alguna trasformación son sobre la fecha de creación y sobre las coordenadas del tweet.

La fecha de creación almacenada en el tweet esta en formato cadena de texto y nosotros queremos almacenarla en nuestra base de datos como un entero formado por la siguiente formula:

AÑO \* 1000000 + MES \* 10000 + DIA \* 100 + HORA.

La almacenaremos así porque luego nos facilita las funciones de agregado. Ya que haciendo una agrupación por este campo ya tenemos la información agregada por hora y haciendo una agregación por este campo divido por 100 tenemos la agregación de la información por día.

Respecto a las coordenadas del tweets tenemos la longitud y la latitud en EPSG4326, es decir, en coordenadas geográficas WGS84y los mapas que vamos a utilizar como base para mostrar nuestros datos sobre ellos que obtendremos de servidor de mapas gratuitos como openstreetmap trabajan con la proyección EPSG3857 Proyección WGS84 / Pseudo-Mercator. Por lo que vamos a proyectar las coordenadas del tweet para almacenarlas en EPSG3857 que es la proyección que vamos a utilizar para mostrarla. Además el problema que nos ocupa no precisa la posición del tweet con tanta precisión , como mucho vamos a agregar la información en cuadrados de 50x50 metros por lo que almacenaremos la posición con esta resolución. Es decir, para cada posición proyectada veremos dentro de que cuadrado de 50x50 metros cae y nos almacenaremos las coordenadas del centro de dicho cuadrado.

La fórmula para obtener las coordenadas del centro del cuadrado a partir de la posición del tweet se puede ver a continuación:

X’ = math.floor(x/50)\*50 + 25;

Y’ = amth.floor(y/50)\*50 + 25;

Tras la ejecución del bloque de procesado de datos tendremos en nuestra base de datos la información de los tweets almacenadas con un resolución temporal horaria y con una resolución espacial de 50 metros en web mercator.

Ejemplos del código utilizado se puede encontrar en el siguiente repositorio de GitHUB:

<https://github.com/patrosau/TFM>

### ALMACENAMIENTO DE DATOS.

En nuestro caso instalamos PostGIS en una máquina virtual Linux siguiendo las instrucciones encontradas en la documentación online de PostIGIS pero una de las razones para escoger PostGIS fue que es multiplataforma por lo que se puede instalar también sobre Windows o Mac.

Una vez instalado el PostGIS vamos a detallar a continuación las decisiones tomadas sobre el diseño de la base de datos implementada.

El diseño de la base de datos se hizo teniendo en cuenta como iban a ser los accesos a los datos. La idea es que los mapas mostrados sean de una zona espacial concreta y de un momento temporal concreto (una hora o un día) por lo que el diseño se hace para que el filtrado de una zona espacial concreta y para un momento temporal concreto sea muy rápido.

Para conseguir esto se utilizan en el diseño los índices espaciales y el particionado temporal de la base de datos. Todas las tablas creadas se definen con un índice espacial y particionadas por tiempo, por hora o por día según corresponda.

El primer almacenamiento de los tweets que se hace en la fase de salida del procesado de datos se hace en tablas particionadas por hora y con índice espacial en la tabla tweets.

La información almacenada inicialmente se agrega por hora y se almacena en la tabla countershour que también tiene particionamiento por hora y índice espacial.

Por último la información inicialmente almacenada se agrega por dia y se almacena en la tabla countersday que también tiene particionamiento, pero en este caso por día e índice espacial.

El particionamiento en postGIS se consigue creando una tabla padre y tantas tablas hijas como particiones se deseen y generando reglas de inserción que hacen que al insertar sobre la tabla padre los datos se inserten en la tabla hija correspondiente según la partición.

Explicaremos esto un poco mejor con un ejemplo práctico. Supongamos que queremos almacenar tweets de las 24 horas del día 1 de Enero del 2017. Para esto crearemos una tabla padre tweets que tenga la definición que queramos y una tabla hija que hereda la definición de la padre para cada una de las horas de partición. Es decir, 24 tablas hijas con nombre tweets\_2017010100, tweets\_2017010101, …, tweets\_2017010123. Además, se crearan reglas en la tabla padre para que si se va a insertar un registro en la tabla cuyo valor de la columna hourslot sea 2017010100 se inserte en la tabla 2017010100 y así sucesivamente. El acceso a estos datos particionados esta optimizado en el diseño de postgis y si se hace un acceso a la tabla padre con un filtro de tiempo para la hora 2017010100 los datos se van a buscar directamente a la tabla hija que corresponda y no se recorren todas las tablas hijas de la tabla padre. Optimizando así las consultas tal y como las va a hacer nuestra herramienta de visualización y en consecuencia nuestro servidor de mapas.

Si el registro que se va a insertar no cumple ninguna de las reglas definidas se inserta directamente en la tabla padre y la consulta de estos datos será más lenta cuantos más datos se hayan insertado en la tabla padre, pero sin penalizar las consultas que sí que tengan tabla particionada definida.

Cuando se quiera desplegar una base de datos propia lo primero que hay que decidir es en que rango de fechas están los datos con los que vamos a trabajar para definir correctamente las particiones.

En el repositorio de código de GitHub hay scripts de base de datos que crean funciones que ayudan una vez definida la tabla padre a crear las tablas hijas y las reglas de inserción. Estas funciones funcionan si en la tabla padre hay una columna llamada hourslot o dayslot que contiene la información temporal y a partir de la cual se quieren hacer las particiones y una columna geom que contiene el punto en EPSG3857 que representa el centro del cuadrado de 50x50 metros.

Por ejemplo, si la definición de la tabla tweets es la siguiente:

CREATE TABLE maps.tweets

(

id bigint NOT NULL,

tweet character varying,

x integer NOT NULL,

y integer NOT NULL,

hourslot bigint NOT NULL,

favourite\_count bigint,

retweet\_count bigint,

in\_reply\_to bigint,

quoted\_status\_id bigint,

retweeted\_status\_id bigint,

user\_id bigint,

geom geometry(Point,3857) NOT NULL

)

y la columna por la que se quiere particionar es hourslot y la columna sobre la que se quiere crear un índice espacial es la geom, la siguiente función crear una tabla hija con el índice espacial y para la partición indicada como parámetro de entrada.

CREATE OR REPLACE FUNCTION dpl.createtable(

tableschema character varying,

tablename character varying,

timeslot character varying,

index\_geom boolean)

RETURNS void AS

$BODY$

DECLARE

table\_temp text := '

CREATE TABLE %%tableschema%%.%%tablename%%\_%%timeslot%% (CHECK (%%condition%%)) INHERITS (%%tableschema%%.%%tablename%%);

CREATE INDEX idx\_%%tablename%%\_%%timeslot%%\_%%timecolumn%% on %%tableschema%%.%%tablename%%\_%%timeslot%% (%%timecolumn%%);

';

indexgeom\_temp text := '

CREATE INDEX idx\_%%tablename%%\_%%timeslot%%\_geom ON %%tableschema%%.%%tablename%%\_%%timeslot%% USING gist(geom);

CLUSTER idx\_%%tablename%%\_%%timeslot%%\_geom ON %%tableschema%%.%%tablename%%\_%%timeslot%%;';

cluster\_temp text := '

CLUSTER idx\_%%tablename%%\_%%timeslot%%\_%%timecolumn%% on %%tableschema%%.%%tablename%%\_%%timeslot%%;';

sql\_exec text := '';

timecolumn varchar(128);

BEGIN

RAISE NOTICE '%: Begin CREATE %.%\_%' , CLOCK\_TIMESTAMP(), $1,$2,$3;

IF char\_length($3)= 10 THEN

timecolumn = 'hourslot';

ELSIF char\_length($3) = 12 THEN

timecolumn = 'timeslot';

ELSE

timecolumn = 'dayslot';

END IF;

IF index\_geom

THEN

sql\_exec := table\_temp || indexgeom\_temp;

ELSE

sql\_exec := table\_temp || cluster\_temp;

END IF;

sql\_exec := REPLACE (sql\_exec,'%%tableschema%%',tableschema);

sql\_exec := REPLACE (sql\_exec,'%%tablename%%',tablename);

sql\_exec := REPLACE (sql\_exec,'%%timeslot%%',timeslot);

sql\_exec := REPLACE (sql\_exec,'%%timecolumn%%',timecolumn);

sql\_exec := REPLACE (sql\_exec,'%%condition%%',format('%s = %s',timecolumn,timeslot));

EXECUTE sql\_exec;

--END;

--END IF;

RAISE NOTICE '%: End CREATE %.%\_%' , CLOCK\_TIMESTAMP(), $1,$2,$3;

END;

$BODY$

LANGUAGE plpgsql VOLATILE

COST 100;

ALTER FUNCTION dpl.createtable(character varying, character varying, character varying, boolean)

OWNER TO postgres;

Mientras que la siguiente función crea la regla de inserción correspondiente a una partición indicada como parámetro de entrada.

CREATE OR REPLACE FUNCTION dpl.createpartitionrules(

\_tableschema text,

\_tablename text,

\_partitioncolumn text,

\_timeslot text)

RETURNS void AS

$BODY$

DECLARE

\_templaterule text := 'CREATE OR REPLACE RULE %%tablename\_timeslot%%\_rule AS ON INSERT TO %%tableschema%%.%%tablename%% WHERE %%condition%% DO INSTEAD INSERT INTO %%tableschema%%.%%tablename\_timeslot%% VALUES (NEW.\*);';

\_sqlrule text := '';

BEGIN

--RAISE NOTICE '%: Creating rules for %.%' , CLOCK\_TIMESTAMP(), \_tableschema, \_tablename;

\_sqlrule := \_templaterule;

\_sqlrule := REPLACE (\_sqlrule,'%%tableschema%%',\_tableschema);

\_sqlrule := REPLACE (\_sqlrule,'%%tablename%%',\_tablename);

\_sqlrule := REPLACE (\_sqlrule,'%%tablename\_timeslot%%',format('%s\_%s', \_tablename, \_timeslot));

\_sqlrule := REPLACE (\_sqlrule,'%%condition%%',format('%s = %s',\_partitioncolumn,\_timeslot));

--RAISE NOTICE '%' , \_sqlrule;

EXECUTE \_sqlrule;

END;

$BODY$

LANGUAGE plpgsql VOLATILE

COST 100;

ALTER FUNCTION dpl.createpartitionrules(text, text, text, text)

OWNER TO postgres;

La siguiente función dados el esquema y nombre de la tabla padre y la fecha de inicio y de fin de las particiones llama a las funciones anteriormente indicadas y genera todas las tablas hijas y reglas necesarias para tener la tabla correctamente particionada.

CREATE OR REPLACE FUNCTION dpl.createhourlychildtables(

tableschema character varying,

tablename character varying,

fromdate date,

todate date)

RETURNS void AS

$BODY$

declare hourslot text:= '';

BEGIN

FOR hourslot in (select dpl.gethourslots(fromdate, todate)) LOOP

EXECUTE dpl.createtable(tableschema,tablename,hourslot,true);

EXECUTE dpl.createpartitionrules(tableschema, tablename,'hourslot', hourslot);

END LOOP;

END;

$BODY$

LANGUAGE plpgsql VOLATILE

COST 100;

ALTER FUNCTION dpl.createhourlychildtables(character varying, character varying, date, date)

OWNER TO postgres;

Todos los scripts utilizados para definir las tablas, particiones, índices y reglas de nuestra implementación se pueden encontrar en el repositorio de github:

La query para agregar la información de los tweets por hora es la siguiente:

INSERT INTO maps.countershour(

hourslot, x, y, tweets\_count, users\_count, avgfavourite, avgretweets,

geom)

SELECT timeslot, x, y, count(\*), count(distinct user\_id), avg(favourite\_count), avg(retweet\_count), st\_setsrid(st\_point(x,y),3857)

FROM maps.tweets

group by timeslot, x, y;

Para agregar la información por día es la siguiente:

INSERT INTO maps.countersday(

dayslot, x, y, tweets\_count, users\_count, avgfavourite, avgretweets,

geom)

SELECT timeslot/100, x, y, count(\*), count(distinct user\_id), avg(favourite\_count), avg(retweet\_count), st\_setsrid(st\_point(x,y),3857)

FROM maps.tweets

group by timeslot/100, x, y;

Luego una vez desplegada nuestra base de datos, ejecutamos el bloque de procesado de datos para rellenar la tabla maps.tweets y ejecutamos las dos queries de agregado mostradas más arriba para rellenar las tabla countersday y countershour que almacenan los datos agregados.

Así ya tenemos todos los datos listos para la generación de nuestro mapas.

### GENERACIÓN DE MAPAS

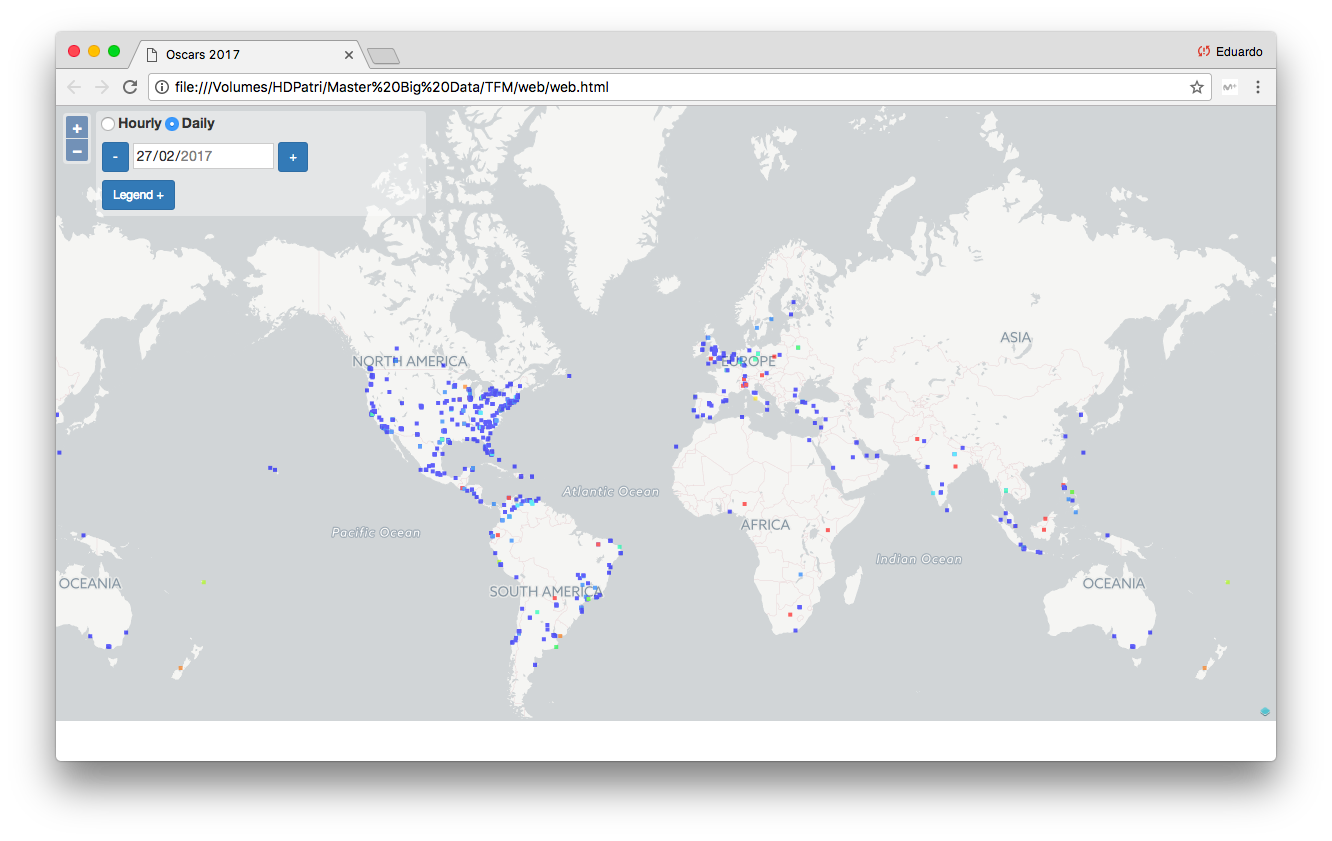
Para implementar el bloque de generación de mapas y para demostrar que cumplimos nuestra máxima de hacer un sistema multiplataforma instalamos Geoserver sobre un sistema operativo MAC OS, siguiendo las instrucciones de instalación proporcionadas en la documentación online de Geoserver.

Una vez instalado Geoserver utilizamos la interfaz web para configurar el mismo para nuestra solución. Tenemos que definir un espacio de trabajo como punto de partida y en este definir un almacén de datos que nos permita conectarnos a nuestra base de datos de PostGIS, como Geoserver está muy bien integrado con PostGIS este paso es muy sencillo. Una vez definido nuestro almacen de datos definimos dos capas una contra la tabla countershour y otra contra la tabla countersday.

Por último, para dar estilo a estas tablas generamos dos estilos de geoserver tweets\_count y tweets\_count\_point. Ambos estilos dibujan un cuadrado en cada posición de un color distinto según el número de tweets que caen en dicho cuadrado, la única diferencia entre uno y otro es el tamaño del cuadrado que dibujan. En el caso del estilo tweets\_count el cuadrado dibujado tiene un tamaño de 50x50 metros en unidades reales del mapa y en el caso de tweets\_count\_point muestra un cuadrado de 4 pixeles de lado. Se crean estos dos estilos porque para niveles de zoom muy alejados el cuadrado de 50x50 metros apenas se ve, por lo que a estos niveles se utilizará el estilo tweets\_count\_point y a niveles más cercanos de zoom el estilo tweets\_count.

Los ficheros sld con la definición de ambos estilos se puede encontrar en el servidor de código de GitHub.

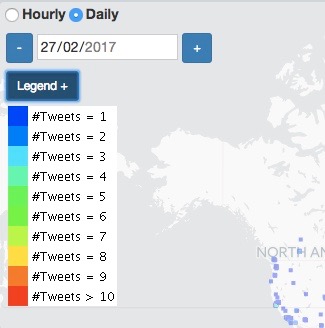
### VISUALIZACIÓN DE LOS MAPAS.



La web creada es una web muy sencilla que permite seleccionar la agregación temporal con la que se quieren ver los datos (horaria o diaria) y permite seleccionar el día y la hora (si corresponde) para la que se quieren filtrar los datos.

Tal y como hemos dicho antes se utiliza openlayers para combinar un mapa de información del terreno facilitado por cartoDB con el mapa generado y servido por geoserver a partir de nuestros datos almacenados en la base de datos y aplicándole uno de nuestros estilos definidos.

Además la web permite visualizar la leyenda del estilo aplicado tal y como aparece en la figura mostrada a continuación.



El código html de la web así como del javascript asociado se puede encontrar también en el repositorio de GitHub.

## PUESTA EN MARCHA.

Primero se hicieron pruebas con un conjunto pequeño de tweets del que se disponía para comprobar la integración entre todos los bloques del sistema y ver que todo funcionaba correctamente.

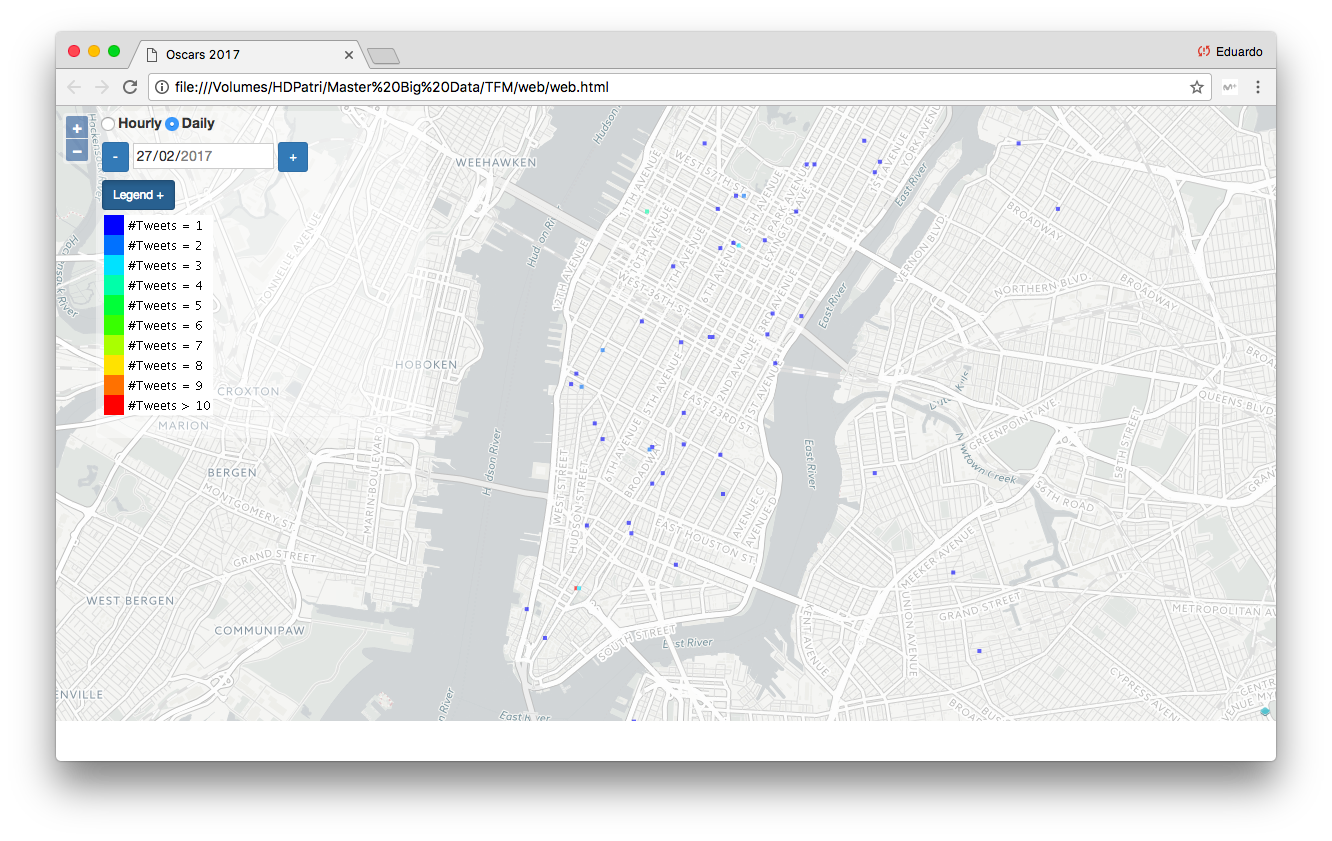
Más tarde se probó el sistema con un conjunto mayor de tweets, un conjunto de tweets relativos a la entrega de los premios oscars del 2017. La colección de tweets contenía algo más de 9 millones de tweets de los cuales únicamente 9000 aproximadamente tenía relleno el campo de las coordenadas por lo que ahí nos encontramos con el primer problema de nuestro sistema.

El correcto funcionamiento de nuestro sistema depende muchísimo de la cantidad de tweets geolocalizados de los que disponemos por lo que si el porcentaje de tweets con la geolocalización activada es muy bajo los datos visualizados igual no son representativos del problema.

En el caso de los tweets de los oscars donde la información más relevante a extraer sería donde se habla de los oscars y durante cuánto tiempo antes y después e la gala se sigue hablando este porcentaje de tweets sí que podría ser representativo. De todas formas, para cada caso de estudio, se debería de hacer un estudio estadístico para determinar si el porcentaje de tweets geolocalizados es suficiente para representar el problema o no.

Otro de los problemas con los que nos encontramos es que el estilo definido inicialmente que es el que dibuja cuadrados de 50x50 metros en unidades reales de mapas no se ve para zooms muy alejados, por lo que se dicidió crear otro estilo con un tamaño del cuadrado en pixels que se adapta al nivel de zoom y que se mostraría en los niveles de zooms alejados donde el original no se ve.

En la siguiente figura se puede ver un ejemplo de cómo el estilo de cuadrados de 50x50 se empieza a ver pequeño para ciertos niveles de zoom.



En el repositorio de GitHub hay un video donde se puede ver el sistema en acción. Para valorar la bondad de la solución hay que tener en cuenta que el hardware sobre el que se ha montado el sistema está muy limitado y están montados tanto la base de datos como el servidor de mapas en el mismo sistema. Luego las mejoras en cuanto al rendimiento son infinitas utilizando servidores de base de datos y de mapas en equipos diferentes mucho más potentes que sobre el que se hizo la prueba.

Aun así la prueba sirve para conseguir el objetivo del proyecto que es montar un sistema que permita visualizar información de los tweets sobre un mapa.

# CONCLUSIONES

# CASOS DE USO

# TRABAJO FUTURO

# ANEXO 1. ENLACES DE INTERES

# ANEXO 2. REPOSITORIO DE CÓDIGO