

**A tool for exploring the tempo-spatial
distribution of Flickr tags**

Part II

Technical Report

Table of Contents

1	Introduction	1
2	Overview of technologies used	2
2.1	PostgreSQL with PostGIS.....	2
2.2	GeoServer	3
2.3	Apache with PHP 5.....	3
2.4	JavaScript.....	4
3	Architecture.....	5
4	Requirements and assumptions	6
5	Database design.....	7
5.1	Entities	7
5.2	Relationships.....	10
5.3	Domains.....	11
6	Physical database.....	13
6.1	Tables	13
6.2	Views.....	20
6.3	Functions	24
7	Classes.....	27
8	GeoServer layers.....	30
9	Web application structure	31
10	Installation.....	34
10.1	Full install.....	34
10.2	Virtual machine install.....	42
	References.....	45
	Appendices.....	47

List of Figures

Figure 1: Architecture used by the tool.....	5
Figure 2: Photo entity and its attributes.....	7
Figure 3: Tag entity and its attribute	8
Figure 4: Data grab entity and its attributes.....	8
Figure 5: Study area entity and its attributes	9
Figure 6: Cluster entity and its attributes	9
Figure 7: Entity-relationship diagram for the application.....	11
Figure 8: Physical database diagram.....	13
Figure 9: Class diagram for photo class	27
Figure 10: Class diagram for tag class.....	27
Figure 11: Class diagram for data_grab class.....	28
Figure 12: Class diagram for cluster class.....	28
Figure 13: Class diagram for dbscan_algorithm class.....	28
Figure 14: Class diagram for study_area class	29
Figure 15: Structure of the root of the web application	31
Figure 16: Structure of the <i>include</i> and <i>action</i> folders within the web application.....	32
Figure 17: Structure of the <i>js</i> and <i>css</i> folders within the web application	33
Figure 18: Configuration example for a GeoServer data store.....	36
Figure 19: Example of the list of layers in GeoServer	37
Figure 20: Example of how to set up a SQL view layer	39
Figure 21: Virtual machine installed on a Windows host.....	43

List of Tables

Table 1: Structure of table <i>photo</i>	14
Table 2: Structure of table <i>tag</i>	15
Table 3: Structure of table <i>data_grab</i>	15
Table 4: Structure of table <i>photo_tag</i>	16
Table 5: Structure of table <i>grabbed_photo</i>	17
Table 6: Structure of table <i>filtered_photo</i>	17
Table 7: Structure of table <i>study_area</i>	18
Table 8: Structure of table <i>cluster</i>	19
Table 9: Structure of table <i>cluster_photo</i>	20
Table 10: Structure of view <i>cluster_summary</i>	20
Table 11: Structure of view <i>photo_reduced</i>	21
Table 12: Structure of view <i>photo_summary</i>	22
Table 13: Structure of view <i>study_area_summary</i>	22
Table 14: Structure of view <i>photos_on_study_area</i>	23
Table 15: Structure of the output of function <i>cluster_neighbours</i>	24
Table 16: Structure of the output of function <i>study_area_as_hex_quadrat</i>	25
Table 17: Structure of the output of function <i>cluster_neighbours</i>	26
Table 18: User logins and passwords for services on the virtual machine	44

1 Introduction

This document provides the technical documentation that describes how the tool was developed and how it works internally. It is structured progressively, starting with a descriptive overview of the technologies used to develop the application and how these technologies were translated into the architecture used by the application.

The document then describes the requirements used as basis for the development of the tool, and how the progressively become part of the database design and physical implementation, and part of the developed object-oriented software solution.

The document also describes how the web application is structured and what layers are published as web services for consumption by the tool.

The last part of the document provides detailed instructions on how to install the application, whether for deployment on a production environment with full documentation on detailed configuration, or for testing purposes on a ready-to-run virtual machine that can be deployed anywhere.

2 Overview of technologies used

The development of the tool required the selection of a set of components that could work together and perform well. For this task, an array of open source technologies was selected based on the author's familiarity with each technology.

From a design point of view, the technologies used can be divided into server-side and client-side (Green and Bossomaier, 2002). Server-side technologies include the database management system which is required to store the information extracted from Flickr, a WMS/WFS compliant web server which can read geographic information from the database and serve it over a network on the necessary formats, and an HTTP web server with a scripting language to retrieve and store information from the database and serve the HTML web pages that make the frontend of the web application.

Client-side technologies include various JavaScript APIs and libraries, and CSS styling to give enhanced interactivity to the end user of the web application.

2.1 PostgreSQL with PostGIS

PostgreSQL is an open source object-relational database system. Originally named Postgres, it was created by Michael Stonebreaker at the University of California at Berkeley in 1986; in 1995 an extended subset of the SQL query language was added to Postgres and in 1996 the project was taken by the open source community (PostgreSQL Global Development Group, 2014b). PostgreSQL was chosen because of its open source nature and the availability of the PostGIS extension.

The PostGIS extension enables support for spatial objects in a PostgreSQL database. PostGIS implements the OpenGIS Implementation for Geographic Information, Simple Features Access for SQL (PostGIS Development Group, 2014b). PostGIS supports planar geometry and geography data types, the former is used on the project to store features such as the locations of the photos, study areas, quadrats and clusters. PostGIS also implements a number of functions to process geometries and calculate spatial relationships and measures, many of these functions are used in the project to find spatial relationships and patterns in the data. Together, these technologies form the spatial processing core of the application.

2.2 GeoServer

GeoServer is a geospatial web server written in Java that implements the open standards set by the Open Geospatial Consortium, like Web Feature Service (WFS) and Web Map Service (WMS) (The GeoServer Project, 2014a). GeoServer can either run as a standalone service or on top a Java servlet container. An Apache Tomcat web application server is used for this purpose.

GeoServer is an open source project and provides an easy to use frontend to manage the publishing of geospatial data. It can connect to a number of vector and raster data sources such as shapefiles and GeoTIFF files, and most importantly it supports native connections to the DBMS chosen for the project (PostgreSQL) (The GeoServer Project, 2014b). The great strength of GeoServer is the wide array of output formats that it supports: KML, JPEG, PNG, SVG, GeoJSON and GML are among them. The data exchange format used in the project is GeoJSON.

2.2.1 GeoJSON

GeoJSON is based on the JavaScript Object Notation (JSON) format, a data exchange format published as the ECMA-404 standard (ECMA International, 2013). GeoJSON is a format specification to encode a series of geographic data structures as a JSON object (Butler et al., 2008). It supports the Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon and GeometriCollection geometry types. This format is used through the project for all queries from GeoServer since it is easily parsed by the JavaScript engines in modern browsers.

2.3 Apache with PHP 5

The Apache HTTP Server together with PHP 5 are the technologies used to serve dynamic web pages by the tool. PHP is used as a server module and is in charge of generating the web pages with the data retrieved from the database. PHP provides functions to connect and query the PostgreSQL instance containing the application's tables and functions via a module which is enabled in the configuration file.

2.3.1 Phlickr

Phlickr is an API kit written in PHP which connects to the REST endpoint of Flickr's web services and provides access to all the functionality exposed by the Flickr Public API. Phlickr is used in the project to query Flickr and retrieve information on the photos and tags.

2.4 JavaScript

JavaScript is a scripting language used mostly in web pages. All modern web browsers include an engine capable of parsing and processing code written in JavaScript. It can access and manipulate the DOM giving client-side interaction and processing capabilities to a web page (Mozilla Developer Network, 2014a).

Three JavaScript libraries are used by the application:

- jQuery¹ is used to provide easy DOM manipulation and traversing
- Bootstrap² is used to provide theming and interactivity
- OpenLayers 3³ is used to provide web mapping capabilities to the application

¹ jQuery, <http://www.jquery.com/>

² Bootstrap, <http://www.getbootstrap.com/>

³ OpenLayers 3, <http://www.ol3js.org/>

3 Architecture

The tool uses the typical client-server architecture for a web application. Requests to the tool are made via a web browser to the application’s main webpage. The browser connects to the web server and sends requests for web pages to be loaded. These web pages contain PHP code which is used to retrieve information and data from the database based on the request made by the user. The PHP parser reads these PHP files and creates the necessary HTML code which is sent back to the browser.

The requests for geographical information on the client side are made by OpenLayers and are handled by GeoServer on the server side. Since Apache Tomcat runs on a different port than the Apache web server, a request from the browser to retrieve information in JSON format from Tomcat would be considered a cross-domain request and would be in violation of the browser’s same-origin policy (Mozilla Developer Network, 2014b). To get around this problem, all calls to GeoServer on Tomcat need to be proxied through a virtual host on Apache. This architecture is shown on Figure 1.

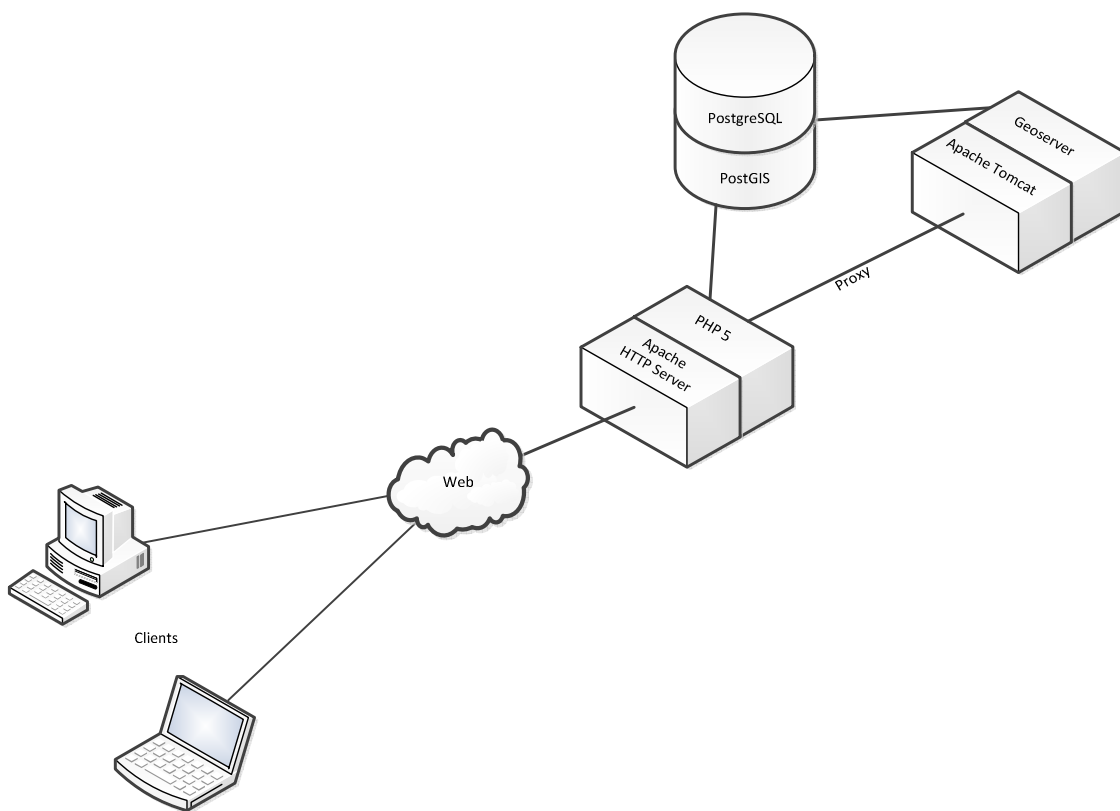


Figure 1: Architecture used by the tool

4 Requirements and assumptions

The purpose of the tool as described in (Rodas Rivera, 2014) is to use photos from Flickr as a source of geographical information and perform spatial and temporal querying and analysis on these datasets, with this in mind the following requirements and assumptions were defined prior to the development of the tool:

REQ 01: The tool must let the user define a set of tags to be retrieved from Flickr.

REQ 02: The tool must be able to connect to Flickr, query photos for a particular tag, and store relevant information relating these photos.

REQ 03: The tool must simplify the dataset of photos, filtering out those taken by the same user on the same day on the same location.

REQ 04: Spatial and temporal analysis must be performed on top of the reduced datasets.

REQ 05: The tool must let the user define a set of study areas for a particular tag. The study area must have an extent both in space and time.

REQ 06: For each study area the tool must allow the user to calculate a set of clusters for every consecutive time unit.

REQ 07: The tool must show the clusters on a map, showing for each cluster on a particular time unit all clusters that overlap with it in the neighbouring time units.

REQ 08: The tool must allow the user to navigate through different time units.

REQ 09: For each study area the user must be able to see the distribution of clusters on the timeframe.

REQ 10: The user must be able to validate the results of each study area with quadrat count analysis.

The analysis performed by the tool assumes that every photo on the reduced dataset represents one event.

5 Database design

5.1 Entities

Taking into consideration the requirements described on the previous section, five entities were defined for the application.

5.1.1 Photo entity

This entity represents a photo that has been downloaded from Flickr. Flickr stores a great number of attributes as metadata for each photo that is uploaded to their servers. From these attributes the tool will query and store the photo id, and the id of the owner, additional data on the owner such as names or usernames will not be downloaded since the tool makes no use or attempt to identify a single user for any particular kind of analysis. Temporal data on the photos will be downloaded in the form of two attributes, the date the photo was taken, and the date the photo was downloaded. Spatial data will include the latitude, the longitude and the accuracy of the locational data on the photo. Finally, three other attributes are necessary to build the URL used to show previews of the photo, these are the codes for the farm, the server, and the secret which is a unique string for each photo. The photo entity and its attributes are shown on Figure 2.

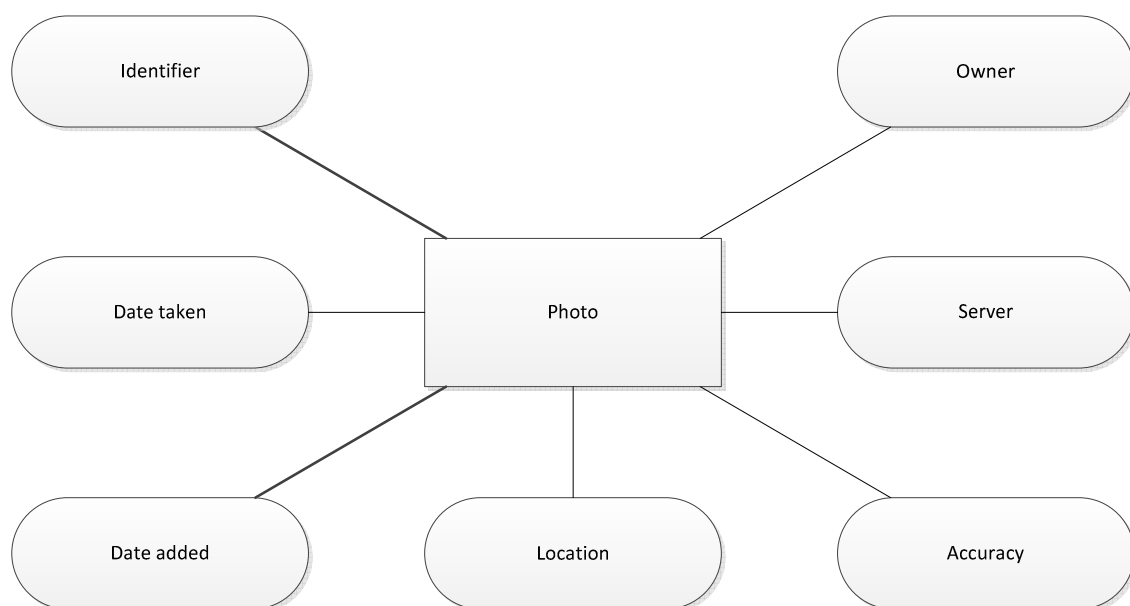


Figure 2: Photo entity and its attributes

5.1.2 Tag entity

This entity represents a tag, defined as a keyword used to identify a photo. The tag entity is very simple and constitutes only the name of the keyword describing the tag. This is shown on Figure 3.

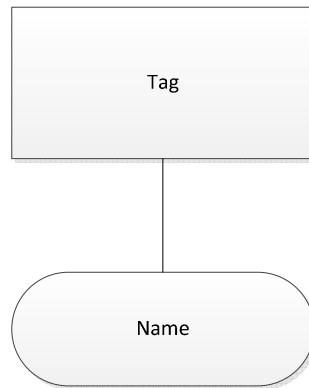


Figure 3: Tag entity and its attribute

5.1.3 Data grab entity

This entity represents the act of downloading a set of photos from Flickr and stores attributes that can act as a log of the download process. These attributes include temporal timestamps of the moment the download process was started and the moment the download process was finished, as well as the total number of photos available indicated by the Flickr API. Two boolean flags are used to denote a row which is marked as the latest download for a particular tag, and another is used to denote a particular data set which has been reduced. The data grab entity and its attributes are shown on Figure 4.

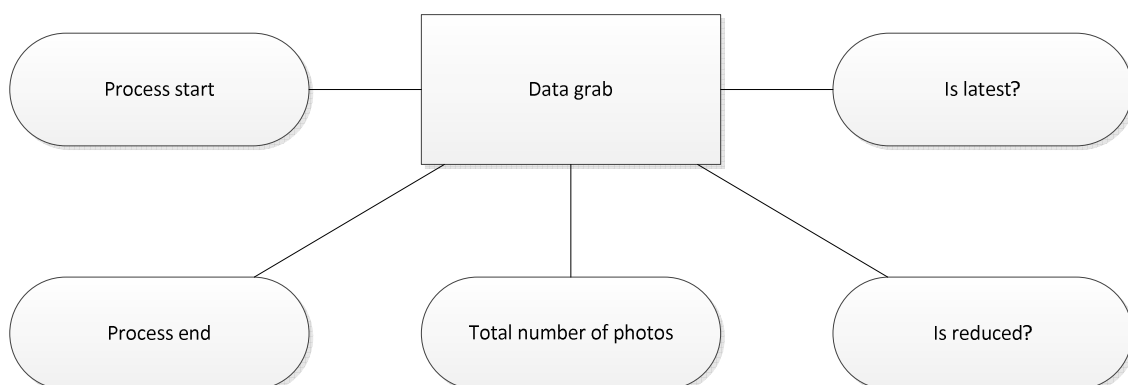


Figure 4: Data grab entity and its attributes

5.1.4 Study area entity

This entity represents a location in space and time related to a particular tag that will be analysed in the search for clustering and possible relationships in the data. A study area has a textual description that identifies it, and exists in both spatial and temporal extents. The spatial extent comprises the geometry object drawn by the user defining the boundaries of the study region. The temporal extent comprises a timeframe with start and end dates, a time unit, and the number of time units in the timeframe.

The study area also keeps information on the parameters needed to search for the clusters, the maximum distance and the minimum number of points. The quadrat size is also stored in the study area. Finally, a timestamp is used to keep track of the last modification made to a study area. The study area entity and its attributes are shown on Figure 5.

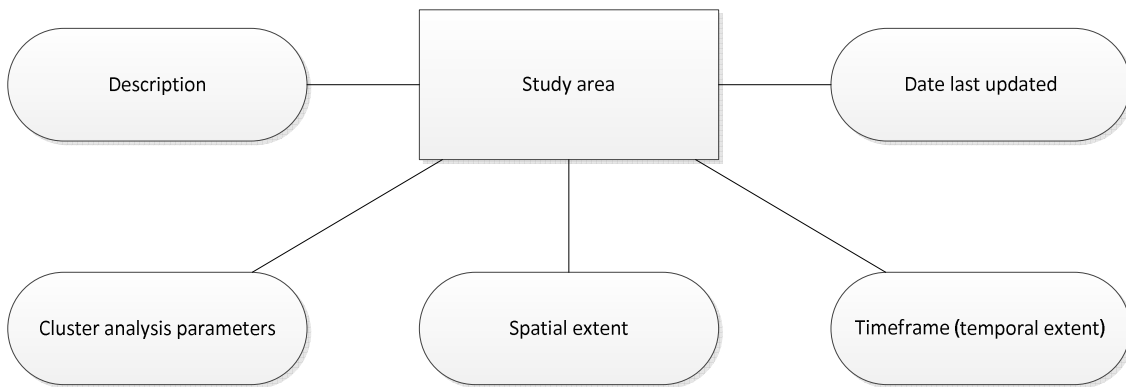


Figure 5: Study area entity and its attributes

5.1.5 Cluster entity

The final entity defined for the application is the cluster. A cluster is tied to a study area and has a spatial extent and a sequential number that describes to which of the time units of the study area it belongs. The cluster entity and its attributes are shown on Figure 6.

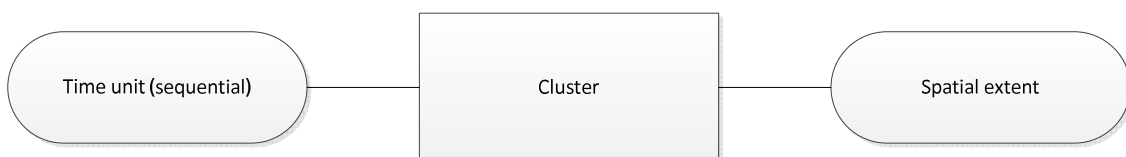


Figure 6: Cluster entity and its attributes

5.2 Relationships

There are many relationships between the entities defined in the previous section. The photo entity has three many-to-many relationships; one with the tag entity, one with the data grab entity and one with the cluster entity. Photos on Flickr can have multiple tags associated with them, and conversely a tag can be applied to multiple photos. A data grab involves the download of multiple photos, and a photo can be involved in multiple data downloads if, for example, the data for the photo is being downloaded for a different tag that the one that already exists on the database. A photo can also be included in many clusters for different study areas, and every cluster includes as an absolute minimum two points.

The tag entity has two one-to-many relationships besides the many-to-many relationship defined above; one with the data grab entity, and one with the study area entity. Photos for a tag can be downloaded in more than one process if new photos are being obtained for an existent tag, but a single download process can perform photo downloads for only one tag. A tag can also have multiple study areas associated with it, but a particular study area can only refer to one tag.

The study area entity has a one-to-many relationship with the cluster entity, besides the one-to-many relationship defined above. A study area can have multiple clusters associated with it, but a particular cluster can only belong to one study area.

These relationships are shown on the E-R diagram on Figure 7.

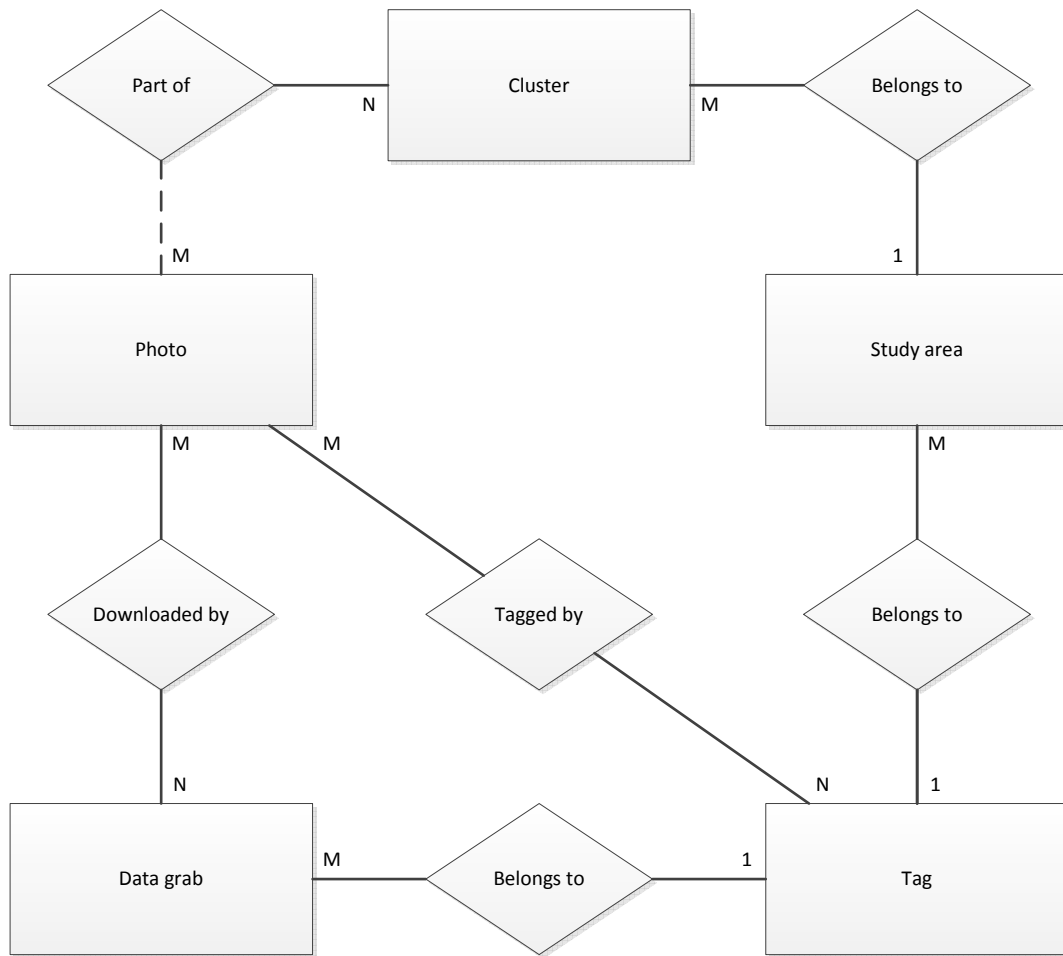


Figure 7: Entity-relationship diagram for the application

5.3 Domains

All entities defined earlier use an auto-increment integer as identifier, which is declared as serial data type in PostgreSQL, this was selected mostly for simplicity when inserting new records. The photo entity could potentially use the photo id assigned by Flickr as an identifier, but although it is unique for every photo, its data type is bigint which uses twice the storage size of a regular integer (8 bytes vs. 4 bytes) (PostgreSQL Global Development Group, 2014a).

Fields which store integer numbers are divided in three data types: bigint (used above for Flickr’s photo id), integer, and smallint. Fields which use the integer data type have different kinds of restrictions, for example a cluster sequential number starts at zero, so this field can store values greater or equal to zero, the number of time units of a study area cannot be zero since there must be at least one time unit in the timeframe

definition, so this field can only store values greater than zero, and, the minimum number of points needed to define a cluster is two, since a cluster cannot be created with just one point, so this field can only store values greater than 1. Maximum values for these fields are not restricted and are only capped by the upper limit of the data type (2147483647).

Fields which are known to use small integer ranges use the smallint data type. These fields include all boolean fields, which are restricted to values between 0 and 1, fields which refer to the accuracy of Flickr's photos are restricted to values between 1 and 16, the field which stores the number of the farm for a particular Flickr photo, which is restricted to values from 1 to 9, and the field which stores the number of the server used to host a particular Flickr photo which is restricted to values greater than zero.

Fields which store numbers with decimal values will use the real data type that on PostgreSQL has a precision of 6 decimal digits, which is enough when storing values such as distances in metres, where 6 decimal digits would mean a precision of up to a micrometre. In the case of the coordinates of photos, the latitude is restricted to values between -90 and 90, and the longitude to values between -180 and 180; all other fields which store decimal numbers are restricted to values greater than zero.

Fields which contain textual descriptions for entities, like the tag name or the study area description, will be declared as variable length character fields with a length limit of 50 characters. A user id on Flickr can have up to 15 alphanumeric characters. The secret for each photo is a fixed length string of 10 characters.

Fields which refer to dates will use the timestamp data type and will be limited to dates greater or equal to January 1st 2000. When an entity has two fields referring to dates, such as a start and an end date, the end date must be greater than the start date.

The field used for defining the time unit on the study area entity will use an interval data type, and will allow values greater or equal to one day, thus the minimal time unit that the application will allow to be used in cluster calculations is one day.

6 Physical database

6.1 Tables

This section describes the physical database tables that were created based on the entity-relationship model described on the last section. The physical database diagram used by the application is shown on Figure 8. The script for the creation of the tables is included in Appendix A.

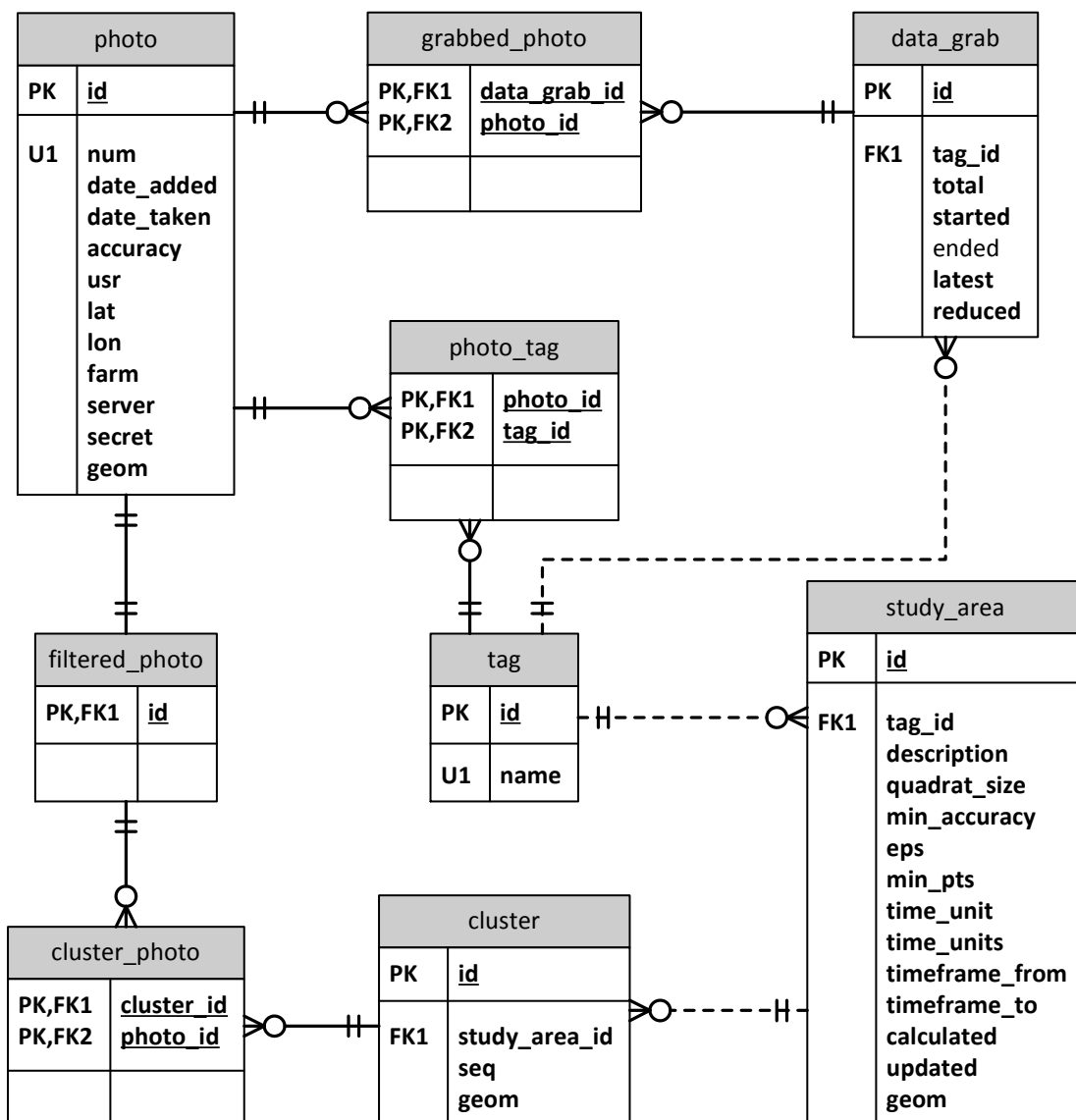


Figure 8: Physical database diagram

6.1.1 Table *photo*

The table *photo* stores the metadata on the photos downloaded from Flickr. It has the structure shown on Table 1.

Table 1: Structure of table *photo*

Attribute	Data type	Allow nulls	Description
id	serial	No	Unique identifier
num	bigint	No	Unique identifier from Flickr
date_added	timestamp	No	Date photo was uploaded
date_taken	timestamp	No	Date photo was taken
accuracy	smallint	No	Declared accuracy
usr	character varying (15)	No	Owner of the photo
lat	real	No	Latitude where photo was taken
lon	real	No	Longitude where photo was taken
farm	smallint	No	Server farm hosting the photo
server	smallint	No	Server hosting the photo
secret	character (10)	No	Secret code of the photo
geom	geometry (POINT, 3857)	No	Photo location

The *id* attribute is the primary key of the table and is auto-incremental.

The *num* attribute is the unique identifier used by Flickr for the photo and constitutes an alternate key. A unique constraint is declared for this attribute so two photos with the same *num* cannot be inserted.

The *geom* attribute is spatially indexed to improve query performance.

Check constraints are included for the following attributes:

- The *num* and *server* attributes, to only allow values greater than zero.
- The *date_added* and *date_taken* attributes, to only allow values greater or equal than January 1st, 2000.

- The *accuracy* attribute, to only allow values between 1 and 16.
- The *lat* attribute, to only allow values between -90 and 90.
- The *lon* attribute, to only allow values between -180 and 180.
- The *farm* attribute, to only allow values between 1 and 9.

6.1.2 Table *tag*

The table *tag* stores the tags defined by the user for tempo-spatial exploration. It has the structure shown on Table 2.

Table 2: Structure of table *tag*

Attribute	Data type	Allow nulls	Description
id	serial	No	Unique identifier
name	character varying (50)	No	Tag description

The *id* attribute is the primary key of the table and is auto-incremental.

The *name* attribute constitutes an alternate key since two tags with identical names on the application cannot exist. A unique constraint is declared for this attribute so two tags with the same *name* cannot be inserted.

6.1.3 Table *data_grab*

The table *data_grab* stores information on each photo download process for a particular tag. It has the structure shown on Table 3.

Table 3: Structure of table *data_grab*

Attribute	Data type	Allow nulls	Description
id	serial	No	Unique identifier
tag_id	integer	No	Identifier of the tag for which photos are downloaded
total	integer	No	Total number of photos for this download as reported by Flickr
started	timestamp	No	Date and time the download process

			started
ended	timestamp	Yes	Date and time the download process ended
latest	smallint	No	Whether this is the latest download for this tag
reduced	smallint	No	Whether photos for this download have been reduced

The *id* attribute is the primary key of the table and is auto-incremental.

The *tag_id* attribute is a foreign key to the table *tag* and is configured to perform cascaded deletes.

Check constraints are included for the following attributes:

- The *total* attribute, to only allow positive values.
- The *started* attribute, to only allow values greater or equal than January 1st, 2000.
- The *ended* attribute, to only allow values greater than the *started* attribute.
- The *latest* and *reduced* attributes, to only allow either 0 or 1 (Boolean).

6.1.4 Table *photo_tag*

The table *photo_tag* stores information on which tags have been applied for each photo. It has the structure shown on Table 4.

Table 4: Structure of table *photo_tag*

Attribute	Data type	Allow nulls	Description
photo_id	integer	No	Identifier of the tagged photo
tag_id	integer	No	Identifier of the tag used on the photo

The *photo_id* and *tag_id* attributes are the compound primary key of the table.

The *photo_id* attribute is a foreign key to the table *photo*, and the *tag_id* attribute is a foreign key to the table *tag*. Both are configured to perform cascaded deletes.

6.1.5 Table *grabbed_photo*

The table *grabbed_photo* stores information on which download process downloaded which photo. It has the structure shown on Table 5.

Table 5: Structure of table *grabbed_photo*

Attribute	Data type	Allow nulls	Description
<i>data_grab_id</i>	integer	No	Identifier of the photo download process
<i>photo_id</i>	integer	No	Identifier of the downloaded photo

The *data_grab_id* and *photo_id* attributes are the compound primary key of the table.

The *data_grab_id* attribute is a foreign key to the table *data_grab*, and the *photo_id* attribute is a foreign key to the table *photo*. Both are configured to perform cascaded deletes.

6.1.6 Table *filtered_photo*

The table *filtered_photo* stores the identifiers of the photos which have passed the pre-processing process. It has the structure shown on Table 6.

Table 6: Structure of table *filtered_photo*

Attribute	Data type	Allow nulls	Description
<i>id</i>	integer	No	Identifier of the photo

The *id* attribute is the primary key of the table.

The *id* attribute is a foreign key to the table *photo* and is configured to perform cascaded deletes.

6.1.7 Table *study_area*

The table *study_area* stores the study areas defined by the user and the parameters to perform cluster analysis. It has the structure shown on Table 7.

Table 7: Structure of table *study_area*

Attribute	Data type	Allow nulls	Description
<i>id</i>	serial	No	Unique identifier
<i>tag_id</i>	integer	No	Identifier of the tag for the study area
<i>description</i>	character varying (50)	No	A textual description of the study area
<i>quadrat_size</i>	real	No	Size of the quadrats to use in cluster validation
<i>min_accuracy</i>	smallint	No	Minimum accuracy of the photos
<i>eps</i>	real	No	Search radius for neighbours
<i>min_pts</i>	integer	No	Minimum number of points to define a cluster
<i>time_unit</i>	interval	No	Time unit of the timeframe
<i>time_units</i>	integer	No	Number of time units in the timeframe
<i>timeframe_from</i>	timestamp	No	Lower boundary of the timeframe
<i>timeframe_to</i>	timestamp	No	Upper boundary of the timeframe
<i>calculated</i>	smallint	No	Whether clusters have been calculated for this study area
<i>updated</i>	timestamp	No	Date and time of last update to the study area
<i>geom</i>	geometry (POLYGON, 3857)	No	Spatial extent of the study area

The *id* attribute is the primary key of the table and is auto-incremental.

The *tag_id* attribute is a foreign key to the table *tag* and is configured to perform cascaded deletes.

The *geom* attribute is spatially indexed to improve query performance.

Check constraints are included for the following attributes:

- The *quadrat_size*, *eps*, and *time_units* attributes, to only allow values greater than zero.
- The *min_accuracy* attribute, to only allow values between 1 and 16.
- The *min_pts* attribute, to only allow values greater than one.
- The *time_unit* attribute, to only allow intervals greater than one day.
- The *timeframe_from* and *updated* attributes, to only allow values greater or equal than January 1st, 2000.
- The *timeframe_to* attribute, to only allow values greater than the *timeframe_from* attribute.
- The *calculated* attribute, to only allow either 0 or 1 (Boolean).

6.1.8 Table *cluster*

The table *cluster* stores the clusters found by the DBSCAN algorithm for each study area. It has the structure shown on Table 8.

Table 8: Structure of table *cluster*

Attribute	Data type	Allow nulls	Description
id	serial	No	Unique identifier
study_area_id	integer	No	Identifier of the study area on which the cluster was found
seq	integer	No	Sequential number of the time unit on which the cluster was found
geom	geometry (POLYGON, 3857)	No	Spatial extent of the cluster (area of influence of the core points)

The *id* attribute is the primary key of the table and is auto-incremental.

The *study_area_id* attribute is a foreign key to the table *study_area* and is configured to perform cascaded deletes.

The *geom* attribute is spatially indexed to improve query performance.

A check constraint is included for the *seq* attribute, to only allow positive values.

6.1.9 Table *cluster_photo*

The table *cluster_photo* stores information on which photos belong to a cluster. It has the structure shown on Table 9.

Table 9: Structure of table *cluster_photo*

Attribute	Data type	Allow nulls	Description
<i>cluster_id</i>	integer	No	Identifier of the cluster
<i>photo_id</i>	integer	No	Identifier of the photo

The *cluster_id* and *photo_id* attributes are the compound primary key of the table.

The *cluster_id* attribute is a foreign key to the table *cluster*, and the *photo_id* attribute is a foreign key to the table *filtered_photo*. Both are configured to perform cascaded deletes.

6.2 Views

Five views were defined in the database to allow easier querying by the application. These views are mostly used to present summaries of the data to the user. The script for the creation of the views is included in Appendix A.

6.2.1 View *cluster_summary*

The view *cluster_summary* summarises the area and the number of clusters for each time unit on each study area. It has the structure shown on Table 10.

Table 10: Structure of view *cluster_summary*

Attribute	Data type	Description
<i>study_area_id</i>	integer	Study area identifier
<i>seq</i>	integer	Sequential of time unit
<i>area</i>	double precision	Area of clusters on this time unit
<i>count</i>	bigint	Number of clusters on this time unit

6.2.2 View *photo_reduced*

The view *photo_reduced* presents the metadata for the subset of photos that have been pre-processed. It has the structure shown on Table 11.

Table 11: Structure of view *photo_reduced*

Attribute	Data type	Description
id	integer	Identifier of the photo
num	bigint	Photo identifier from Flickr
date_added	timestamp	Date photo was uploaded
date_taken	timestamp	Date photo was taken
accuracy	smallint	Declared accuracy
usr	character varying (15)	Owner of the photo
lat	real	Latitude where photo was taken
lon	real	Longitude where photo was taken
farm	smallint	Server farm hosting the photo
server	smallint	Server hosting the photo
secret	character (10)	Secret code of the photo
geom	geometry (POINT, 3857)	Photo location
tag_id	integer	Identifier of the tag used on the photo
tag	character varying (50)	The tag used on the photo

6.2.3 View *photo_summary*

The view *photo_summary* presents a summary of the photos that exist per tag. It is used on the application's main page. It has the structure shown on Table 12.

Table 12: Structure of view *photo_summary*

Attribute	Data type	Description
id	integer	Identifier of the tag
name	character varying (50)	Name of the tag
count	bigint	Number of photos for this tag
started	timestamp	Date and time this tag was last updated
reduced	smallint	Whether photos for this tag have been reduced
num_reduced	bigint	Number of reduced photos for this tag

6.2.4 View *study_area_summary*

The view *study_area_summary* summarises data for all study areas. It is used on the application's main page. It has the structure shown on Table 13.

Table 13: Structure of view *study_area_summary*

Attribute	Data type	Description
id	integer	Identifier of the study area
description	character varying (50)	Textual description of the study area
tag_id	integer	Identifier of the tag for the study area
name	character varying (50)	Name of the tag
quadrat_size	real	Size of the quadrats to use in cluster validation
min_accuracy	smallint	Minimum accuracy of the photos
eps	real	Search radius for neighbours
min_pts	integer	Minimum number of points to define a cluster
time_unit	interval	Time unit of the timeframe
time_units	integer	Number of time units in the timeframe
timeframe_from	timestamp	Lower boundary of the timeframe
timeframe_to	timestamp	Upper boundary of the timeframe
updated	timestamp	Date and time of last update to the study area

clusters	bigint	Number of cluster found in the study area
----------	--------	---

6.2.5 View *photos_on_study_area*

The view *study_area_summary* lists all photos that are located within all study areas. It has the structure shown on Table 14.

Table 14: Structure of view *photos_on_study_area*

Attribute	Data type	Description
study_area_id	integer	Identifier of the study area
tag_id	integer	Identifier of the tag
id	integer	Identifier of the photo
num	bigint	Photo identifier from Flickr
date_added	timestamp	Date photo was uploaded
date_taken	timestamp	Date photo was taken
accuracy	smallint	Declared accuracy
usr	character varying (15)	Owner of the photo
lat	real	Latitude where photo was taken
lon	real	Longitude where photo was taken
farm	smallint	Server farm hosting the photo
server	smallint	Server hosting the photo
secret	character (10)	Secret code of the photo
geom	geometry (POINT, 3857)	Photo location

6.3 Functions

Five functions were defined in the database to perform spatial querying and analysis. The scripts are written on PL/PGSQL. The script for the creation of the views is included in Appendix A.

6.3.1 Function *reduce_photos*

This function populates the *filtered_photo* table by selecting only one photo from the *photo* table for each user on a particular day and location, thus removing all photos taken by the same user on the same day from the analysis. Photos are selected based on their distance to the geographic mean of the group of points and the accuracy of the photos.

The function receives the identifier of the tag for which photos need to be reduced, and outputs the final number of reduced photos.

6.3.2 Function *cluster_neighbours*

This function finds all clusters that overlap on consecutive time units from a specific time unit. The function first loops through all clusters in the specified time unit and for each cluster it loops again from two time units in the past to two time units in the future to find clusters that overlap with it.

The function receives the identifier of the study area and the time unit on which to look for overlapping clusters on consecutive time units, and outputs the list of clusters that overlap indicating for each its relative position in time with the specified time unit. The structure of the output generated by the *cluster_neighbours* function is shown on Table 15.

Table 15: Structure of the output of function *cluster_neighbours*

Attribute	Data type	Description
id	integer	Identifier of the cluster
seq_delta	integer	Relative position in time to specified time unit
geom	geometry (POLYGON, 3857)	Spatial extent of the cluster (area of influence of the core points)

6.3.3 Function *clusters_overlap*

This function checks whether clusters on a specified time unit have clusters overlapping in other specified time unit.

The function receives the identifier of the study area, the sequential of the current time unit, and the relative position for the time unit on which to look for overlapping clusters, and outputs one when there are overlapping clusters or zero otherwise.

6.3.4 Function *study_area_as_hex_quadrat*

This function creates a hexagonal grid (beehive) that fills the specified study area, without overlapping on the borders. The length of the sides of each hexagon is equal to the value entered for quadrat size by the user on the study area. The origin of the grid is located on the lower corner of the bounding box of the study area. A hexagonal grid was selected for the application since it proved to full arbitrary shaped, non-rectangular, study areas better than regular square grids.

The function receives the identifier of the study for which to build the hexagonal grid, and outputs the list of hexagons that make up the grid. The structure of the output generated by the *study_area_as_hex_quadrat* function is shown on Table 16.

Table 16: Structure of the output of function *study_area_as_hex_quadrat*

Attribute	Data type	Description
id	integer	Identifier of the quadrat
geom	geometry (POLYGON, 3857)	Spatial extent of the hexagonal quadrat

6.3.5 Function *quadrat_count*

This function calculates the number of photos that fall within each quadrat in a specified study area and time unit.

The function receives the identifier of the study area and the sequential of the time unit for which to calculate the quadrat count, and outputs the list of quadrats with their

associated photo count. The structure of the output generated by the *quadrat_count* function is shown on Table 17.

Table 17: Structure of the output of function *cluster_neighbours*

Attribute	Data type	Description
id	integer	Identifier of the quadrat
count	bigint	Number of photos in the quadrat
geom	geometry (POLYGON, 3857)	Spatial extent of the hexagonal quadrat

7 Classes

PHP is the server-side language chosen to develop the web application. Although PHP is a scripting language, it contains support for object-oriented programming (The PHP Group, 2014). Six classes were created to build the logic of the application. Five of them represent the entities defined on the database design section and provide direct access to data storage and retrieval with their respective tables. The last class is an implementation of the DBSCAN clustering algorithm for cluster analysis.

The code for the classes together with a complete description and documentation on each of the classes' attributes and functions is included in Appendix G. Class diagrams for the six classes are shown on Figures 9 to 14.

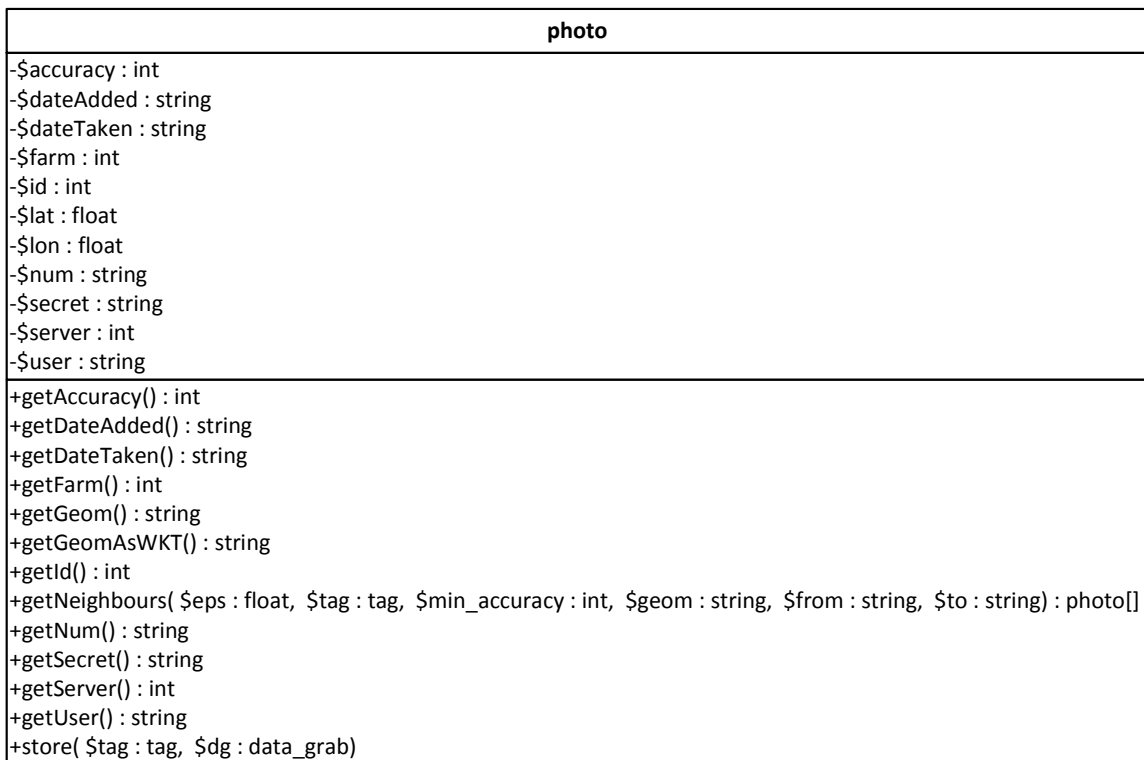


Figure 9: Class diagram for photo class

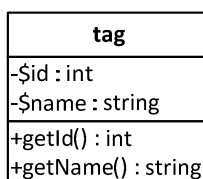


Figure 10: Class diagram for tag class

data_grab
-\$api : Phlickr_Api -\$ended : DateTime -\$id : int -\$n_photos : int -\$page : int -\$pages : int -\$parms : Mixed[] -\$started : DateTime -\$tag : tag -\$total : int
-done() +getEnded() : string +getId() : int -getMinUploadDate() : string +getNPhotos() : int +getStarted() : string +getTag() : tag +getTotal() : int +grab() -recalculateParams(\$lt : string)

Figure 11: Class diagram for data_grab class

cluster
-\$density_reachable : photo[] -\$id : int -\$points : photo[]
+add(\$point : photo, \$density_reachable : bool) +getNumPoints() : int +getPoints() : photo[] +store(\$id : int, \$seq : int, \$eps : float)

Figure 12: Class diagram for cluster class

dbscan_algorithm
-\$clusters : cluster[] -\$eps : float -\$from : string -\$geom : string -\$min_accuracy : int -\$min_pts : int -\$points : photo[] -\$tag : tag -\$to : string -\$visited : int[]
-expand_cluster(\$cluster : cluster, \$point : photo, \$neighbours : photo[]) : cluster +find_clusters() +getClusters() : cluster[] +getEps() : float +getMinPoints() : int +getNumClusters() : int -merge(\$seeds : photo[], \$neighbours : photo[]) : photo[] +store(\$id : int, \$seq : int)

Figure 13: Class diagram for dbscan_algorithm class

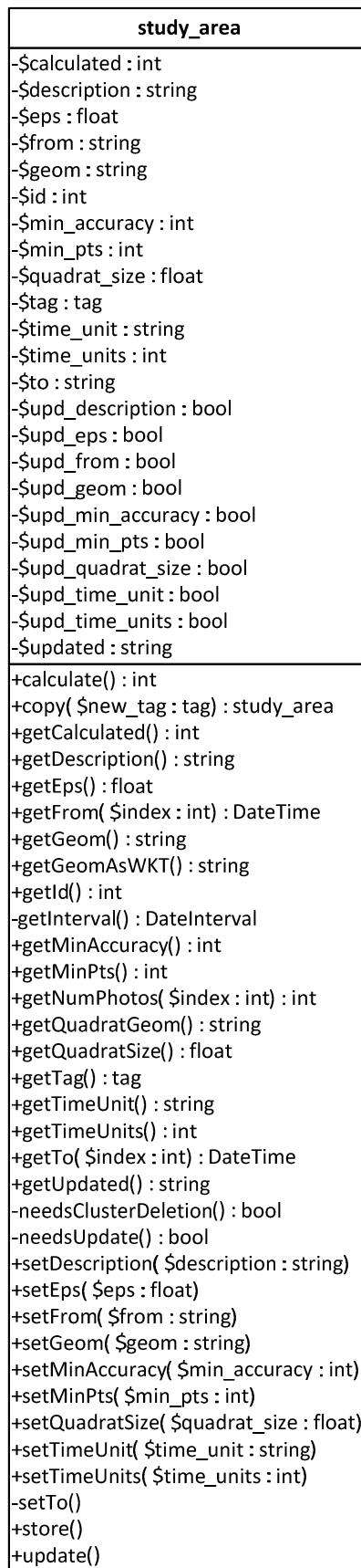


Figure 14: Class diagram for study_area class

8 GeoServer layers

GeoServer is the spatial web server used by the application to host the WMS and WFS web services used by the viewer. Six web services are defined on GeoServer to be consumed by the tool.

The *cluster_neighbours* layer is an SQL view defined on GeoServer that publishes the clusters found on a study area. It requires two parameters, the identifier of the study area and the time unit for which to retrieve the clusters. It returns two attributes, the identifier of the cluster, and the time difference of the cluster to the current time unit. The geometry type is Polygon.

The *photo* layer is also an SQL view defined on GeoServer that publishes the non-reduced set of photos with tag information. It does not require any parameters. It returns 14 parameters, which are all the parameters from the photo table plus the tag identifier and the tag name associated to a photo. The geometry type is Point.

The *photo_reduced* layer publishes the view with the same name on the database. It returns the same parameters and has the same geometry type as the *photo* layer, but for the reduced dataset of photos.

The *quadrat_count* layer is an SQL view defined on GeoServer that publishes the quadrats on a study area. It requires two parameters, the identifier of the study area and the time unit for which to calculate the quadrat count. It returns two attributes, the identifier of the quadrat and the photo count on that quadrat. The geometry type is Polygon.

The *study_area* layer publishes the table with the same name on the database. It returns all the attributes that are part of this table. The geometry type is Polygon.

The *study_area_quadrat* layer is also an SQL view defined on GeoServer that publishes the border of the area defined by the hexagonal quadrat grid for a study area. It requires the identifier of the study area as a parameter. The geometry type is Polygon.

9 Web application structure

The web application has four main web pages which can be accessed by the user: *index.php*, *viewer.php*, *manual.php* and *license.php*. The first gives the user access to the control panel where data maintenance and processing options can be accessed for tags, study areas and clusters. The second gives the user access to the viewer in which tempo-spatial exploration and cluster analysis and validation options can be performed. The third contains the user manual which has detailed instructions on how to use the application. The fourth displays the license of the application.

On the root of the application there are also six directories which contain other files used by the main web pages to load content, access the database and provide styling.

The root of the web application is structured as shown on Figure 15.

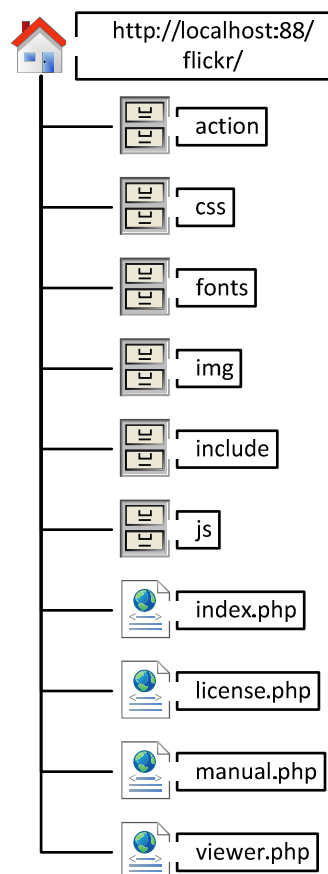


Figure 15: Structure of the root of the web application

The *action* directory contains all the scripts used by the main web pages when making AJAX requests to insert, update, delete, or retrieve data from the database, or perform server side processing like the reduction of photos, the calculation of clusters, or quadrat count analysis.

The *include* directory contains files meant to be used by the main web pages or by the action scripts to perform actions on the database objects. Files in this directory include all the classes described on the previous section as well as the application’s configuration file (*vars.php*) which contains global configuration parameters for the application. This directory also contains geoPHP⁴, a PHP library to perform geometry operations which is used by the study area class to read the geometry column from the database.

The structure of the files on these folders is shown on Figure 16.

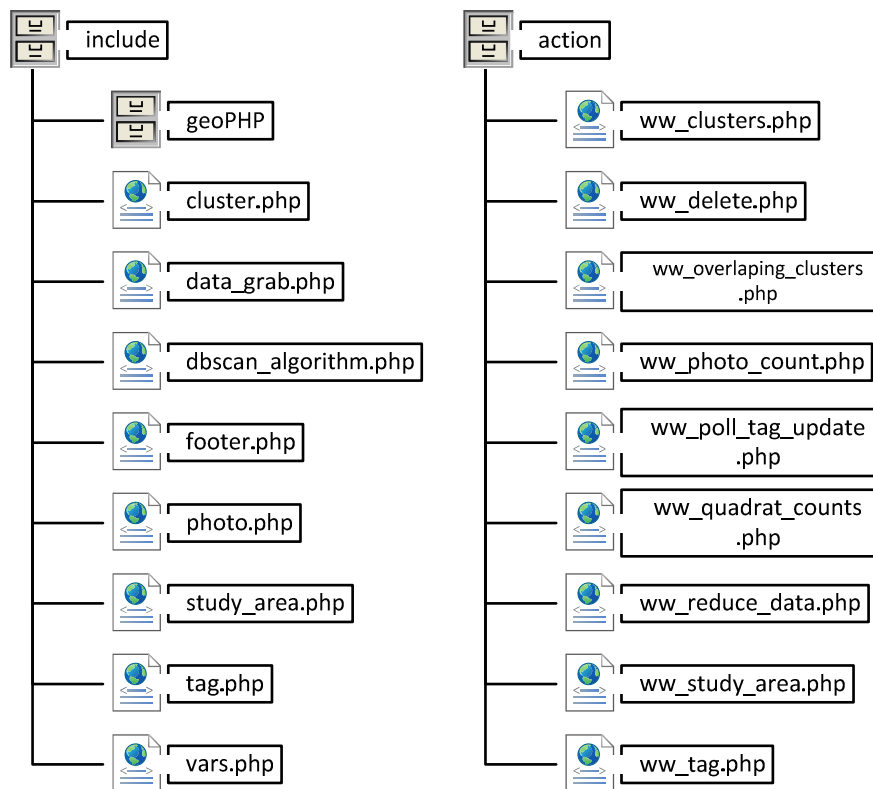


Figure 16: Structure of the *include* and *action* folders within the web application

⁴ geoPHP, <https://github.com/phayes/geoPHP>

The *js* and *css* folders contain the JavaScript and CSS scripts used by the main web pages to provide styling and interactivity. These folders also contain the files needed for the jQuery, Bootstrap, and OpenLayers 3 libraries.

The structure of the files on these folders is shown on Figure 17.

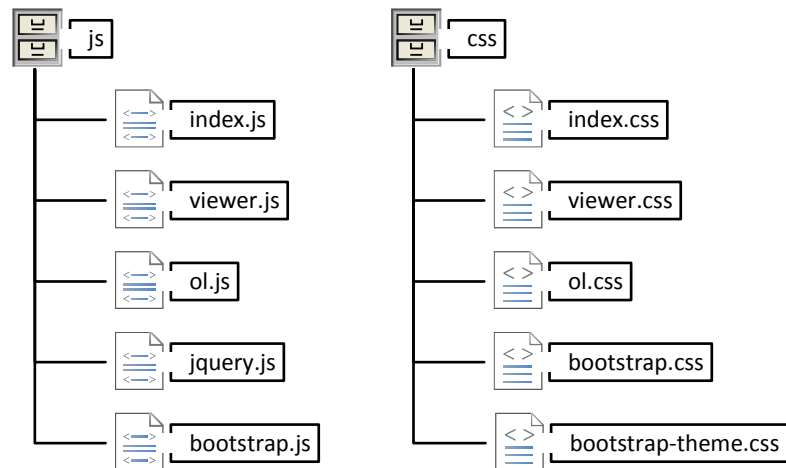


Figure 17: Structure of the *js* and *css* folders within the web application

The *img* folder contains the images used on the web application, this include the images used on the creation of the bar chart for temporal analysis of clusters. The *fonts* folder contains Bootstrap's glyphicons used by the web application on the main web pages to show importance in some headers.

10 Installation

The tool is provided in the accompanying DVD in two formats, source code that can be deployed on a web server with the correct set up, and as a virtual machine that can be run straight from a computer by installing VirtualBox⁵. The procedure to install and run the software for both of these scenarios is described in this section.

10.1 Full install

The full installation should be performed when the tool is going to be installed on a production server, or on a location where it will be deployed permanently.

It is not recommended to perform the full installation on a machine if the tool is going to be used just for testing or browsing since it will be lengthy and involve the installation of various programs, services, and the editing of configuration files. If the tool is to be used for testing or browsing then the virtual machine installation described on the next section is more appropriate.

10.1.1 Requirements

The application can be installed on either a Windows or a Linux host. Installation on Mac hosts should be similar but this has not been tested.

The following is a list of the software used to develop and test the application. These are considered the minimum requirements to run the tool. Newer versions of these programs should be used if available.

- Apache HTTP Server 2.4.9 with PHP 5.5.12
- GeoServer 2.5.2 on Apache Tomcat 8.0.9
- PostgreSQL 9.3.4 with PostGIS 2.1

Step by step instructions on how to install each of these programs are out of the scope of this technical report. Guides on how to install these programs are readily available online. This document will only present the specific configuration steps that need to be taken for the application to run on a hypothetical newly installed system running the above mentioned programs.

⁵ VirtualBox, <http://www.virtualbox.org/>

10.1.2 PostgreSQL configuration

PostGIS is an extension to PostgreSQL that needs to be enabled in the database that will host the data for the tool (PostGIS Development Group, 2014a). To enable PostGIS for a database, the following command can be run from PostgreSQL's prompt:

```
CREATE EXTENSION postgis;
```

Or if the user has access to pgAdmin III then right-clicking on the extensions list of any database and selecting *New Extension...* will bring a graphical interface from which PostGIS can be selected and installed.

It is recommended to create a new database for the use of the tool as to not interfere with other existing applications.

After enabling PostGIS in the database that will be used by the application it is necessary to run the database creation script. This script is included on the accompanying DVD on the folder *sql* and is also listed on Appendix A. Running this script will create all the necessary tables, views and functions needed by the application to work. Please note that the script will create the database objects in the public schema.

10.1.3 GeoServer configuration

GeoServer can either be installed standalone or on top of Apache Tomcat. Both kinds of installations will work fine for the application. Once GeoServer has been installed, head to its main web page and login, the default username is admin and the default password is GeoServer.

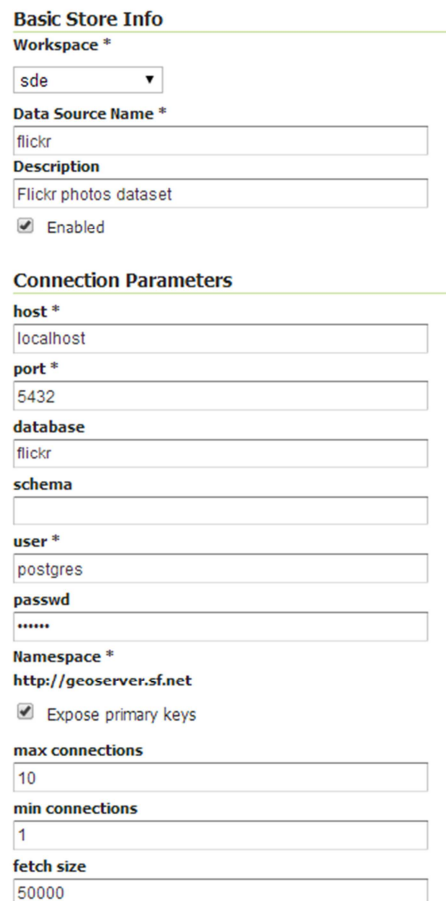
For this example GeoServer will be located in <http://localhost:8080/GeoServer/>.

To add a new data store that connects to the recently created database for the application perform the following steps:

- On the menu on the left select *Stores* and then *Add new store*.
- Under *Vector Data Sources* select *PostGIS*.
- Select a workspace for the new data store and add a name and a description.

- On the database connection parameters specify the address where PostgreSQL is installed, the port (or leave the default), the name of the database where the application objects were created, and the username and password to connect.
- Select *Expose primary keys* and increase fetch size to 50000.
- Click *Save*.

An example of this configuration can be seen on Figure 18.



The image shows a web-based configuration form for a GeoServer data store. It is divided into two main sections: 'Basic Store Info' and 'Connection Parameters'.
Basic Store Info:
- **Workspace ***: A dropdown menu with 'sde' selected.
- **Data Source Name ***: A text input field containing 'flickr'.
- **Description**: A text input field containing 'Flickr photos dataset'.
- **Enabled**: A checked checkbox.
Connection Parameters:
- **host ***: A text input field containing 'localhost'.
- **port ***: A text input field containing '5432'.
- **database**: A text input field containing 'flickr'.
- **schema**: An empty text input field.
- **user ***: A text input field containing 'postgres'.
- **passwd**: A text input field containing '.....'.
- **Namespace ***: A text input field containing 'http://geoserver.sf.net'.
- **Expose primary keys**: A checked checkbox.
- **max connections**: A text input field containing '10'.
- **min connections**: A text input field containing '1'.
- **fetch size**: A text input field containing '50000'.

Figure 18: Configuration example for a GeoServer data store

The application uses six layers served from GeoServer to display geographical information. Two of these layers are configured to serve tables or views from the database, the other four layers are views defined on GeoServer. The list of layers is shown under the sde workspace in Figure 19.

To create the layers that serve objects from the database perform the following steps for photo_reduced and study_area:

1. On the menu on the left select *Layers* and then *Add a new resource*.
2. In the *Add layer from* dropdown select the data store created earlier.
3. Select *Publish* on one of the three objects mentioned above (repeat for the rest).
4. On *Declared SRS* select EPSG:3857.
5. Compute both bounding boxes.
6. Click *Save*.

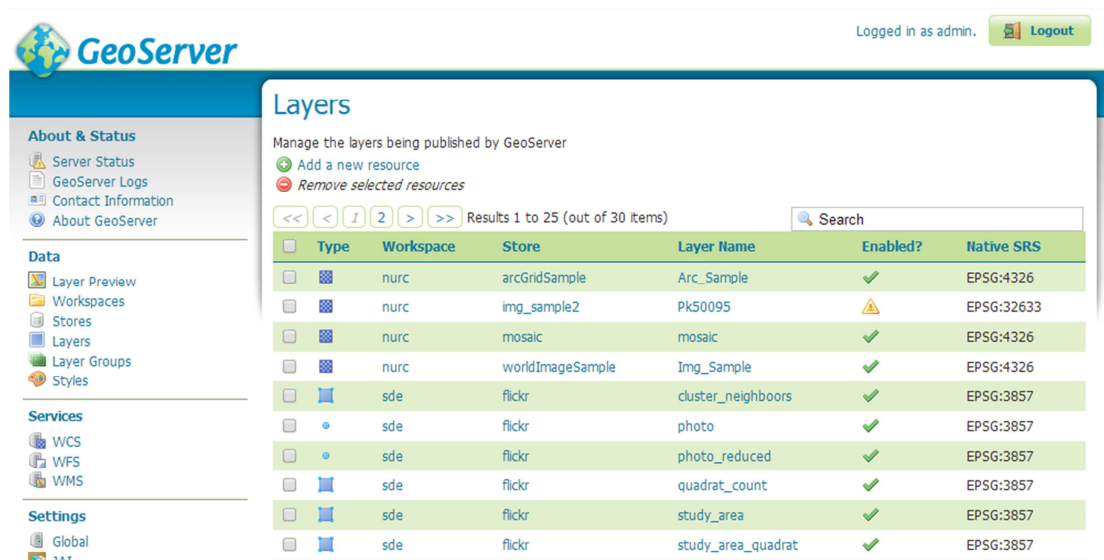


Figure 19: Example of the list of layers in GeoServer

To create the layers defined as views from GeoServer perform the following steps for each of the four views:

1. On the menu on the left select *Layers* and then *Add a new resource*.
2. In the *Add layer from* dropdown select the data store created earlier.
3. Select *Configure new SQL view...*
4. Add the view name
5. Add the SQL statement
6. Select *Guess parameters from SQL* and set the default values and regular expression.
7. Click *Refresh* under *Attributes* and set geometry type, SRID and id as layer identifier.
8. Click *Save*.
9. On *Declared SRS* select EPSG:3857.

10. Compute both bounding boxes.
11. Click *Save*.

In steps 4 to 7 add the following for each view:

- View name: cluster_neighboors
- SQL statement: `SELECT * FROM cluster_neighboors(%id%, %seq%)`
- SQL view parameters:
 - id 1 `^[\d]+$`
 - seq 0 `^[\d]+$`
- Geometry type: Polygon 3857

- View name: photo
- SQL statement: `SELECT a.*, b.tag_id, c.name AS tag FROM photo a INNER JOIN photo_tag b ON a.id = b.photo_id INNER JOIN tag c ON b.tag_id = c.id`
- No SQL view parameters
- Geometry type: Point 3857

- View name: quadrat_count
- SQL statement: `SELECT * FROM quadrat_count(%id%, %seq%)`
- SQL view parameters:
 - id 1 `^[\d]+$`
 - seq 0 `^[\d]+$`
- Geometry type: Polygon 3857

- View name: study_area_quadrat
- SQL statement: `SELECT %id% AS id, ST_Union(ST_SnapToGrid(geom, 0.0001)) AS geom FROM study_area_as_hex_quadrat(%id%)`
- SQL view parameters:
 - id 1 `^[\d]+$`
- Geometry type: Polygon 3857

There should be seven layers on the layer list in GeoServer after following these steps. An example of a view layer correctly configured is shown on Figure 20.

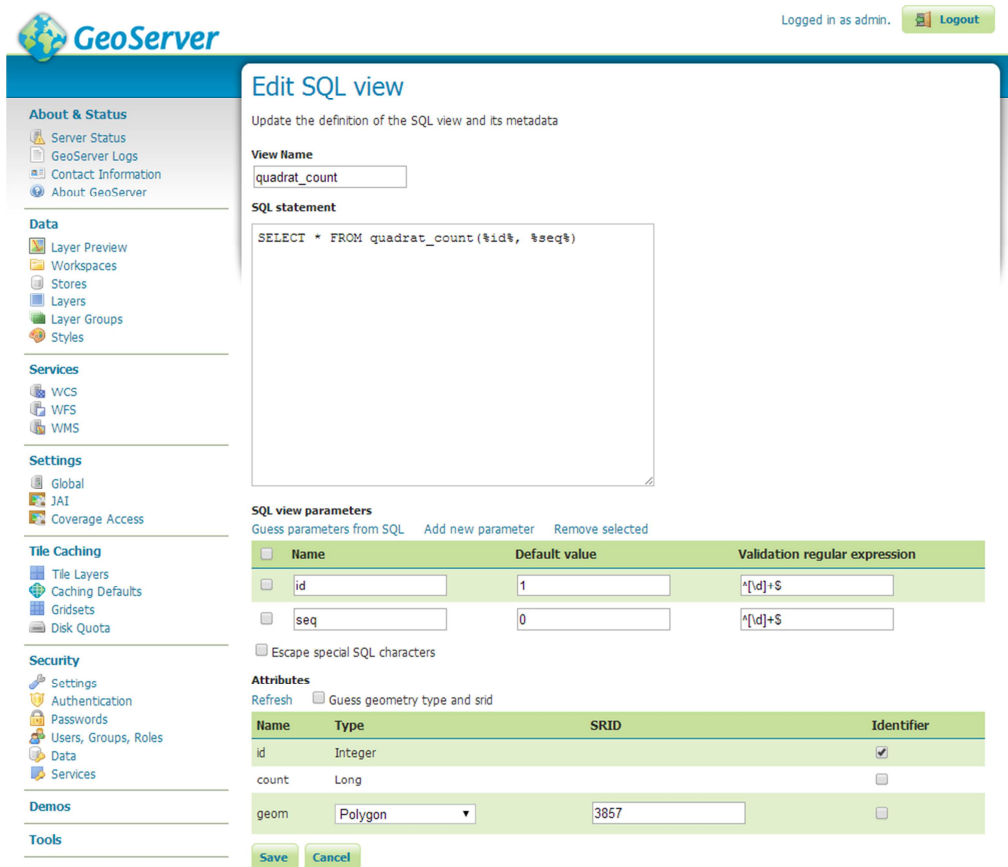


Figure 20: Example of how to set up a SQL view layer

10.1.4 Apache and PHP configuration

For a detailed guide on how to set up Apache with PHP and Phlickr see (Kunkle and Morton, 2006).

A working installation of Apache HTTP Server configured with PHP is required for the application to run. If the user needs to install Apache and has a Windows host then a good solution is to download a pre-compiled and ready-to-run installation of Apache and PHP for Windows like WAMP Server⁶ or XAMPP⁷, please note that these packages come with MySQL installed by default, so they may need to be configured to connect to PostgreSQL.

⁶ WAMP Server, <http://www.wampserver.com/>

⁷ XAMPP, <https://www.apachefriends.org/>

PHP needs to have the following extensions enabled:

- php_curl
- php_gd2
- php_pdo_pgsql

These extensions can be enabled by uncommenting their respective lines in the php.ini configuration file.

The PEAR framework is required to install Phlickr. PEAR stands for PHP Extension and Application Repository. PEAR can be enabled on Windows hosts by running the go-pear.bat script on PHP's installation directory.

After installing PEAR copy the file Phlickr-0.2.8.tgz located on the Phlickr folder on the accompanying DVD to a location on the hard drive. Using a command prompt use cd to change folder to the location where the file was copied and run:

```
C:\> pear install Phlickr-0.2.8.tgz
```

This will install Phlickr. As of June 27th, 2014 the Flickr API went SSL-only (Martin, 2014). This change made calls to the Flickr public API REST endpoint with Phlickr to stop working. Code on the project's website has not been upgrade so it needs to be updated manually after installing Phlickr. The following steps must be followed to allow Phlickr to communicate to the new Flickr API REST endpoint:

1. Go to the directory where Phlickr was installed, either in the directory where PHP was installed or in a subdirectory there called PEAR.
2. Open the file Request.php.
3. Look for the submitHttpPost function.
4. On the second line of this function, after the declaration of the variable \$ch, add the following:

```
curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, false);
```

5. Save the file.
6. Open the file Api.php.
7. Look for the constant called REST_ENDPOINT_URL and change its value to:

```
const REST_ENDPOINT_URL = 'https://api.flickr.com/services/rest/';
```

8. Save the file.

This will complete the configuration and installation of Phlickr.

Apache needs to proxy all calls made to GeoServer, this is because calls from Apache to GeoServer are considered by the browser as cross-domain request and thus are blocked by the browser's same-origin policy. To configure Apache to proxy calls made to GeoServer the following modules need to be enabled on Apache's httpd.conf configuration file:

- proxy_module
- proxy_http_module
- headers_module
- deflate_module
- xml2enc_module
- proxy_html_module

Also, add the following block of text to the end of Apache's httpd.conf configuration file:

```
<VirtualHost *:80>
  ProxyPass /GeoServer http://localhost:8080/GeoServer
  ProxyPassReverse /GeoServer http://localhost:8080/GeoServer
</VirtualHost>
```

Please make sure to add the correct address for the installation used for GeoServer in the previous section. After this configuration is done and Apache has been restarted, all calls made to localhost/GeoServer on the Apache web server will be proxied to localhost:8080/GeoServer.

Finally, the source code for the application can be copied to the www directory of Apache to publish it and access online. The source code is under the folder *source code* on the accompanying DVD. The default name of the directory of the application is *flickr*, but it can be changed if needed. To access the tool, open a web browser and go to:

<http://localhost/flickr/>

This will load the tool's control panel. The tool needs access to the internet to download photos from Flickr and the base map on the viewer.

10.2 Virtual machine install

The virtual machine install is more appropriate if the tool is going to be used for browsing or testing. The virtual machine contains a minimal installation of Ubuntu Server 14.04 for virtual machines and is completely configured with PostgreSQL, PostGIS, Tomcat, GeoServer, Apache and PHP, and is ready to run the tool without major configuration from the user.

10.2.1 Requirements

The only software requirement to run the tool on a virtual machine is the installation of VirtualBox. VirtualBox can run on any operating system so this installation method will work on Windows, Linux and Mac.

The accompanying DVD includes the installer for version 4.3.12-93733 of VirtualBox for Windows which was used to create and setup the virtual machine.

The virtual machine installation requires approximately 3 GB of free disk space.

Please note that hardware virtualisation technologies (VT-x/AMD-V) may need to be enabled on the CPU from the BIOS of the host computer to run virtual machines.

10.2.2 VirtualBox configuration

To deploy the pre-configured virtual machine to an installation of VirtualBox perform the following steps:

1. Open VirtualBox and on the *File* menu select *Import Appliance...*
2. On the *Appliance to import* dialogue select *Choose a virtual appliance file to import...*
3. Browse to the *vm* folder on the accompanying DVD and select the *Dissertation.ova* file.
4. Click *Next*.

5. Review the details of the virtual machine to be deployed.
6. Click *Import*.
7. Accept the software licence.

VirtualBox will import the virtual machine file and once the process has finished it will be added to the list of installed virtual machines as seen on Figure 21.

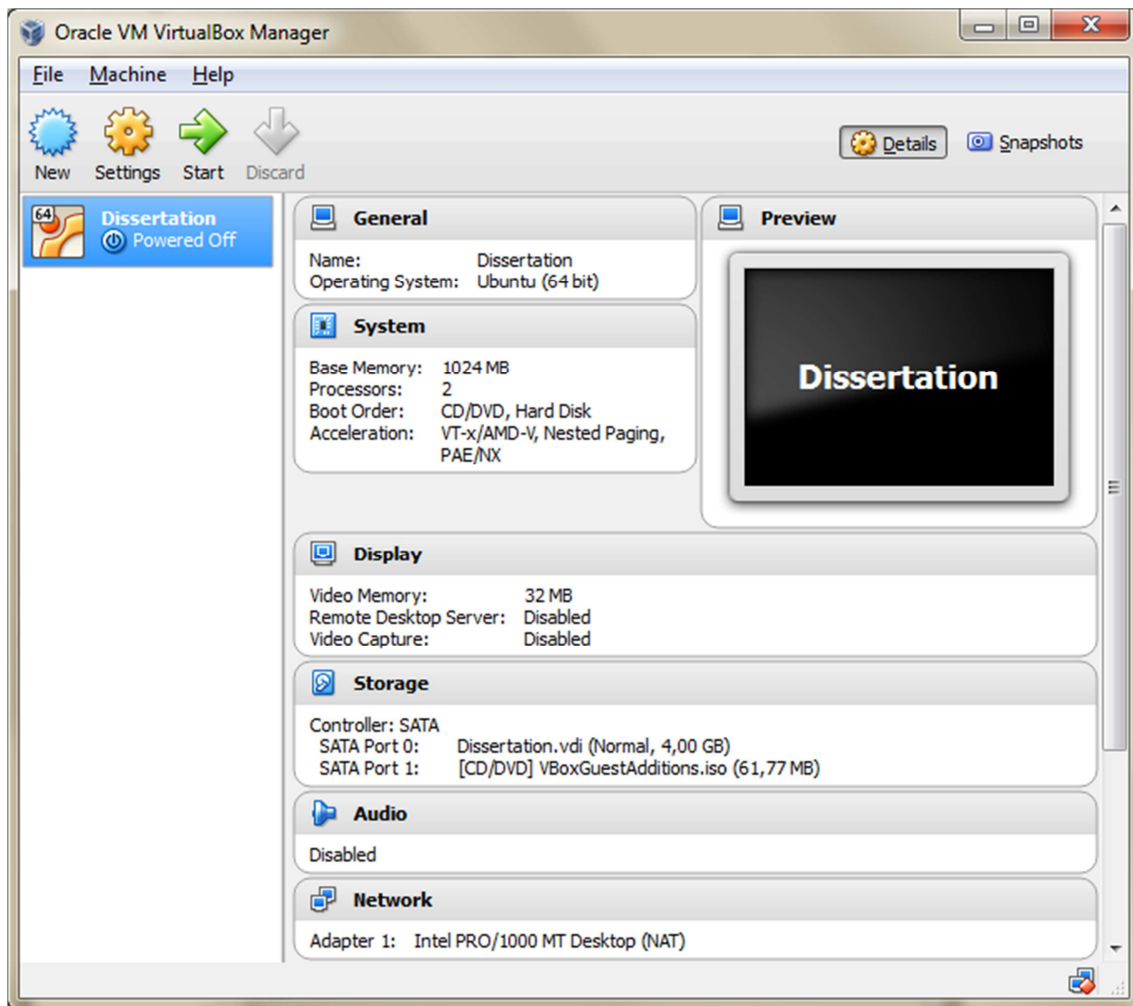


Figure 21: Virtual machine installed on a Windows host

To run the tool, simply launch the virtual machine selecting *Start*. When Ubuntu finishes loading a prompt will appear. To access the tool, open a web browser and go to:

<http://localhost:88/flickr/>

This will load the tool's control panel. The tool needs access to the internet to download photos from Flickr and the base map on the viewer.

To access the GeoServer instance on the virtual machine, open a web browser and go to:

`http://localhost:88/GeoServer/`

To connect to the PostgreSQL instance on the virtual machine, use port 5433 on localhost.

Table 18 lists all the user logins and passwords for the services installed on the virtual machine.

Table 18: User logins and passwords for services on the virtual machine

Service	Username	Password
System user	juan	flickr
PostgreSQL	postgres	flickr
Tomcat application manager	admin	flickr
GeoServer	admin	GeoServer

To shut down the virtual machine simply go to its prompt, login if not logged in already, and type:

```
juan@s1366017:~$ sudo shutdown -P now
```

Type the system user password on Table 18 and the virtual machine will shut down.

References

Butler, H., Daly, M., Doyle, A., Gillies, S., Schaub, T. and Schmidt, C., 2008. *GeoJSON Specification*. [online] Available at: <<http://www.geojson.org/geojson-spec.html>> [Accessed 14 Jun. 2014].

ECMA International, 2013. *The JSON Data Interchange Format*. [online] Geneva, Switzerland, p.14. Available at: <<http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf>>.

Green, D. and Bossomaier, T., 2002. *Online GIS and Spatial Metadata*. London, UK: Taylor & Francis, p.222.

Kunkle, R. and Morton, A., 2006. *Building Flickr Applications with PHP*. Berkeley, CA, USA: Apress, p.216.

Martin, C., 2014. *Flickr API going SSL-Only on June 27th, 2014*. [online] Available at: <<http://code.flickr.net/2014/04/30/flickr-api-going-ssl-only-on-june-27th-2014/>> [Accessed 27 Jun. 2014].

Mozilla Developer Network, 2014a. *About JavaScript - JavaScript | MDN*. [online] Available at: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript> [Accessed 15 Jun. 2014].

Mozilla Developer Network, 2014b. *Same-origin policy - Web security | MDN*. [online] Available at: <https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy> [Accessed 11 Jun. 2014].

PostGIS Development Group, 2014a. *PostGIS -- Installation*. [online] Available at: <<http://www.postgis.net/install>> [Accessed 16 Jun. 2014].

PostGIS Development Group, 2014b. *PostGIS: Home*. [online] Available at: <<http://postgis.refrains.net/>> [Accessed 16 Jun. 2014].

PostgreSQL Global Development Group, 2014a. *PostgreSQL: Documentation: 9.3: Numeric Types*. [online] Available at: <<http://www.postgresql.org/docs/9.3/static/datatype-numeric.html>> [Accessed 11 Jul. 2014].

PostgreSQL Global Development Group, 2014b. *PostgreSQL: History*. [online] Available at: <<http://www.postgresql.org/about/history/>> [Accessed 14 Jun. 2014].

Rodas Rivera, J.L., 2014. *A tool for exploring the tempo-spatial distribution of Flickr tags*. University of Edinburgh, p.31.

The GeoServer Project, 2014a. *About - GeoServer*. [online] Available at: <<http://www.geoserver.org/about/>> [Accessed 12 Jun. 2014].

The GeoServer Project, 2014b. *GeoServer User Manual -- GeoServer 2.5.x User Manual*. [online] Available at: <<http://docs.geoserver.org/stable/en/user/>> [Accessed 20 Jun. 2014].

The PHP Group, 2014. *PHP: Introduction - Manual*. [online] Available at: <<https://www.php.net/manual/en/oop5.intro.php>> [Accessed 15 Jun. 2014].

Appendices

List of Appendices

Appendix A – SQL script to create the database objects	49
Appendix B – SQL script to remove the database objects	58
Appendix C – PHP code for the main web pages	59
Appendix D – JavaScript code for the main web pages	75
Appendix E – CSS code for the main web pages	95
Appendix F – PHP code for the files in the action folder	99
Appendix G – PHP code for the files in the include folder	112

Appendix A – SQL script to create the database objects

```

-----
-- TABLES -----
-----

-- Table photo
CREATE TABLE photo
(
    id serial NOT NULL,
    num bigint NOT NULL CHECK (num > 0),
    date_added timestamp NOT NULL CHECK (date_added >= '2000-01-01
00:00:00'::timestamp),
    date_taken timestamp NOT NULL CHECK (date_taken >= '2000-01-01
00:00:00'::timestamp),
    accuracy smallint NOT NULL CHECK (accuracy BETWEEN 1 AND 16),
    usr character varying(15) NOT NULL,
    lat real NOT NULL CHECK (lat BETWEEN -90 AND 90),
    lon real NOT NULL CHECK (lon BETWEEN -180 AND 180),
    farm smallint NOT NULL CHECK (farm BETWEEN 1 AND 9),
    server smallint NOT NULL CHECK (server > 0),
    secret character(10) NOT NULL,
    geom geometry(POINT, 3857) NOT NULL,
    CONSTRAINT pk_photo PRIMARY KEY (id),
    CONSTRAINT uk_photo UNIQUE (num)
);

-- Table tag
CREATE TABLE tag
(
    id serial NOT NULL,
    name character varying(50) NOT NULL,
    CONSTRAINT pk_tag PRIMARY KEY (id),
    CONSTRAINT uk_tag UNIQUE (name)
);

-- Table data_grab
CREATE TABLE data_grab
(
    id serial NOT NULL,
    tag_id integer NOT NULL,
    total integer NOT NULL CHECK (total >= 0),
    started timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP CHECK
(started >= '2000-01-01 00:00:00'::timestamp),
    ended timestamp CHECK (started < ended),
    latest smallint NOT NULL DEFAULT 1 CHECK (latest BETWEEN 0 AND 1),
    reduced smallint NOT NULL DEFAULT 0 CHECK (reduced BETWEEN 0 AND
1),
    CONSTRAINT pk_data_grab PRIMARY KEY (id),
    CONSTRAINT fk_data_grab_tag FOREIGN KEY (tag_id)
    REFERENCES tag (id) MATCH SIMPLE
    ON UPDATE RESTRICT ON DELETE CASCADE
);

-- Table photo_tag
CREATE TABLE photo_tag
(
    photo_id integer NOT NULL,
    tag_id integer NOT NULL,
    CONSTRAINT pk_photo_tag PRIMARY KEY (photo_id, tag_id),

```

```

    CONSTRAINT fk_photo_tag_photo FOREIGN KEY (photo_id)
      REFERENCES photo (id) MATCH SIMPLE
      ON UPDATE RESTRICT ON DELETE CASCADE,
    CONSTRAINT fk_photo_tag_tag FOREIGN KEY (tag_id)
      REFERENCES tag (id) MATCH SIMPLE
      ON UPDATE RESTRICT ON DELETE CASCADE
  );

-- Table grabbed_photo
CREATE TABLE grabbed_photo
(
  data_grab_id integer NOT NULL,
  photo_id integer NOT NULL,
  CONSTRAINT pk_grabbed_photo PRIMARY KEY (data_grab_id, photo_id),
  CONSTRAINT fk_grabbed_photo_grab FOREIGN KEY (data_grab_id)
    REFERENCES data_grab (id) MATCH SIMPLE
    ON UPDATE RESTRICT ON DELETE CASCADE,
  CONSTRAINT fk_grabbed_photo_photo FOREIGN KEY (photo_id)
    REFERENCES photo (id) MATCH SIMPLE
    ON UPDATE RESTRICT ON DELETE CASCADE
);

-- Table filtered_photo
CREATE TABLE filtered_photo
(
  id integer NOT NULL,
  CONSTRAINT pk_filtered_photo PRIMARY KEY (id),
  CONSTRAINT fk_filtered_photo_photo FOREIGN KEY (id)
    REFERENCES photo (id) MATCH SIMPLE
    ON UPDATE RESTRICT ON DELETE CASCADE
);

-- Table study_area
CREATE TABLE study_area
(
  id serial NOT NULL,
  tag_id integer NOT NULL,
  description character varying(50) NOT NULL DEFAULT 'No
description',
  quadrat_size real NOT NULL DEFAULT 0 CHECK (quadrat_size > 0),
  min_accuracy smallint NOT NULL DEFAULT 16 CHECK (min_accuracy
BETWEEN 1 AND 16),
  eps real NOT NULL CHECK (eps > 0),
  min_pts integer NOT NULL CHECK (min_pts > 1),
  time_unit interval NOT NULL CHECK (time_unit >= '1
day'::interval),
  time_units integer NOT NULL CHECK (time_units > 0),
  timeframe_from timestamp NOT NULL CHECK (timeframe_from >= '2000-
01-01'::timestamp),
  timeframe_to timestamp NOT NULL CHECK (timeframe_from <
timeframe_to),
  calculated smallint NOT NULL DEFAULT 0 CHECK (calculated BETWEEN 0
AND 1),
  updated timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP CHECK
(updated >= '2000-01-01 00:00:00'::timestamp),
  geom geometry(POLYGON, 3857) NOT NULL,
  CONSTRAINT pk_study_area PRIMARY KEY (id),
  CONSTRAINT fk_study_area_tag FOREIGN KEY (tag_id)
    REFERENCES tag (id) MATCH SIMPLE
    ON UPDATE RESTRICT ON DELETE CASCADE
);

```

```

-- Table cluster
CREATE TABLE cluster
(
  id serial NOT NULL,
  study_area_id integer NOT NULL,
  seq integer NOT NULL CHECK (seq >= 0),
  geom geometry(POLYGON, 3857) NOT NULL,
  CONSTRAINT pk_cluster PRIMARY KEY (id),
  CONSTRAINT fk_cluster_study_area FOREIGN KEY (study_area_id)
    REFERENCES study_area (id) MATCH SIMPLE
    ON UPDATE RESTRICT ON DELETE CASCADE
);

-- Table cluster_photo
CREATE TABLE cluster_photo
(
  cluster_id integer NOT NULL,
  photo_id integer NOT NULL,
  CONSTRAINT pk_cluster_photo PRIMARY KEY (cluster_id, photo_id),
  CONSTRAINT fk_cluster_photo_cluster FOREIGN KEY (cluster_id)
    REFERENCES cluster (id) MATCH SIMPLE
    ON UPDATE RESTRICT ON DELETE CASCADE,
  CONSTRAINT fk_cluster_photo_filtered_photo FOREIGN KEY (photo_id)
    REFERENCES filtered_photo (id) MATCH SIMPLE
    ON UPDATE RESTRICT ON DELETE CASCADE
);

-----
-- INDEXES -----
-----

-- Geographic index for photos
CREATE INDEX gx_photo ON photo USING GIST (geom);

-- Geographic index for study areas
CREATE INDEX gx_study_area ON study_area USING gist (geom);

-- Geographic index for clusters
CREATE INDEX gx_cluster ON cluster USING GIST (geom);

-----
-- VIEWS -----
-----

-- View that summarises area and count of the clusters in the database
CREATE OR REPLACE VIEW cluster_summary AS
SELECT study_area_id, seq, SUM(ST_Area(geom)) AS area, COUNT(seq) AS
count
FROM cluster
GROUP BY study_area_id, seq;

-- View that presents all the data for the reduced photo dataset
CREATE OR REPLACE VIEW photo_reduced AS
SELECT a.id, a.num, a.date_added, a.date_taken, a.accuracy, a.usr,
a.lat, a.lon, a.farm, a.server, a.secret, a.geom, c.tag_id, d.name AS
tag
FROM photo a INNER JOIN filtered_photo b ON a.id = b.id
INNER JOIN photo_tag c ON a.id = c.photo_id
INNER JOIN tag d ON c.tag_id = d.id;

```

```

-- View that summarises number of photos per tag, used in the
application's main page
CREATE OR REPLACE VIEW photo_summary AS
SELECT a.id, a.name, COUNT(c.photo_id), b.started, b.reduced, b.count
AS num_reduced
FROM tag a LEFT OUTER JOIN (
    SELECT a.tag_id, a.started, a.reduced, COUNT(c.id) AS count
    FROM data_grab a INNER JOIN photo_tag b ON a.tag_id = b.tag_id
LEFT OUTER JOIN filtered_photo c ON b.photo_id = c.id
    WHERE a.latest = 1
    GROUP BY a.tag_id, a.started, a.reduced
) b ON a.id = b.tag_id
LEFT OUTER JOIN photo_tag c ON a.id = c.tag_id
GROUP BY a.id, a.name, b.started, b.reduced, b.count;

-- View that summarises data for the study areas, used in the
application's main page
CREATE OR REPLACE VIEW study_area_summary AS
SELECT a.id, a.description, a.tag_id, b.name, a.quadrat_size,
a.min_accuracy, a.eps, a.min_pts, a.time_unit, a.time_units,
a.timeframe_from::date, a.timeframe_to::date, a.updated,
CASE WHEN a.calculated = 0 THEN -1 ELSE COALESCE(c.count, 0) END AS
clusters
FROM study_area a INNER JOIN tag b ON a.tag_id = b.id
LEFT OUTER JOIN (
    SELECT study_area_id, COUNT(*) as count FROM cluster GROUP BY
study_area_id
) c ON a.id = c.study_area_id;

-- View that selects the identifiers of the photos located within each
study area
CREATE OR REPLACE VIEW photos_on_study_area AS
SELECT a.id AS study_area_id, b.tag_id, d.*
FROM study_area a INNER JOIN photo_tag b ON a.tag_id = b.tag_id INNER
JOIN filtered_photo c ON b.photo_id = c.id INNER JOIN photo d ON c.id
= d.id
WHERE d.accuracy >= a.min_accuracy AND d.date_taken BETWEEN
a.timeframe_from AND a.timeframe_to AND ST_Within(d.geom, a.geom);

-----|
-- FUNCTIONS -----|
-----|

-- Selects the photos to be reduced for a particular tag
CREATE OR REPLACE FUNCTION reduce_photos(a_tag_id integer) RETURNS
integer AS $$
DECLARE
    f1 record; -- Outer loop, every unique pair of user and date taken
    f2 record; -- Inner loop, every photo by the same user on the same
day
    a_x double precision; -- Stores cumulative x coordinates
    a_y double precision; -- Stores cumulative y coordinates
    i integer := 0; -- Counts the number of photos
    j integer; -- Stores the number of photos by a same user on the
same day to calculate the average
BEGIN
    -- Delete existing filtered photos
    DELETE FROM filtered_photo WHERE id IN (SELECT a.id FROM photo a
INNER JOIN photo_tag b ON a.id = b.photo_id WHERE b.tag_id =
a_tag_id);
    -- Delete existing clusters for this tag

```



```

DELETE FROM cluster WHERE id IN (SELECT a.id FROM cluster a INNER
JOIN study_area b ON a.study_area_id = b.id WHERE b.tag_id =
a_tag_id);
-- Set all study areas for this tag as not calculated
UPDATE study_area SET calculated = 0, updated = now() WHERE tag_id
= a_tag_id;
-- Loop through every unique pair of user and date taken
FOR f1 IN
    SELECT DISTINCT a.usr, a.date_taken::date, COUNT(*) AS count
FROM photo a INNER JOIN photo_tag b ON a.id = b.photo_id WHERE
b.tag_id = a_tag_id GROUP BY a.usr, a.date_taken::date
LOOP
    i := i + 1;
    -- If there is only one photo insert it in the filtered list
    IF f1.count = 1 THEN
        INSERT INTO filtered_photo SELECT a.id FROM photo a INNER
JOIN photo_tag b ON a.id = b.photo_id WHERE b.tag_id = a_tag_id AND
a.usr = f1.usr AND a.date_taken::date = f1.date_taken;
    ELSE
        a_x := 0;
        a_y := 0;
        j := 0;
        -- Else, add the x and y coordinates of each photo
        FOR f2 IN
            SELECT ST_X(a.geom) AS x, ST_Y(a.geom) AS y FROM photo
a INNER JOIN photo_tag b ON a.id = b.photo_id WHERE b.tag_id =
a_tag_id AND a.usr = f1.usr AND a.date_taken::date = f1.date_taken
        LOOP
            j := j + 1;
            a_x := a_x + f2.x;
            a_y := a_y + f2.y;
        END LOOP;
        -- Get the average
        a_x := a_x / j;
        a_y := a_y / j;
        -- Insert the photo located closest to the average to the
filtered list
        FOR f2 IN
            SELECT a.id, ST_Distance(a.geom,
ST_GeomFromText('POINT(' || a_x || ' ' || a_y || ')', 3857)) AS
distance FROM photo a INNER JOIN photo_tag b ON a.id = b.photo_id
WHERE b.tag_id = a_tag_id AND a.usr = f1.usr AND a.date_taken::date =
f1.date_taken ORDER BY a.accuracy DESC, distance FETCH FIRST ROW ONLY
        LOOP
            INSERT INTO filtered_photo VALUES (f2.id);
        END LOOP;
    END IF;
END LOOP;
-- Set dataset as reduced
UPDATE data_grab SET reduced = 1 WHERE tag_id = a_tag_id;
-- Return the number of photos in the reduced dataset
RETURN i;
END;
$$ LANGUAGE plpgsql;

-- Returns all clusters that overlap clusters up to two time units
before and after of the specified sequence
CREATE OR REPLACE FUNCTION cluster_neighbors(a_id integer, a_seq
integer) RETURNS TABLE(id integer, seq_delta integer, geom geometry)
AS $$
DECLARE

```

```

    f1 record; -- Outer loop, all clusters in the specified sequence
    f2 record; -- Inner loop, all the clusters that intersect
BEGIN
    -- Loop through all clusters in the specified sequence
    FOR f1 IN
        SELECT a.id, a.geom FROM cluster a WHERE a.study_area_id =
a_id AND a.seq = a_seq
    LOOP
        -- Loop from two time units before to two time units after the
specified sequence
        FOR i IN -2..2 LOOP
            -- Return current cluster if 0
            IF i = 0 THEN
                RETURN QUERY SELECT f1.id, i, f1.geom;
                CONTINUE;
            END IF;
            -- Return all the clusters that intersect
            FOR f2 IN
                SELECT a.id, a.geom FROM cluster a WHERE
a.study_area_id = a_id AND a.seq = a_seq + i AND
ST_Intersects(f1.geom, a.geom)
            LOOP
                RETURN QUERY SELECT f2.id, i, f2.geom;
            END LOOP;
        END LOOP;
    END LOOP;
END;
$$ LANGUAGE plpgsql;

-- Returns whether clusters on a specified sequence of a study area
have overlapping clusters in another sequence defined by a delta
CREATE OR REPLACE FUNCTION clusters_overlap(a_id integer, a_seq
integer, a_delta integer) RETURNS integer AS $$
DECLARE
    f1 record; -- Outer loop, all clusters in this sequence
    f2 record; -- Inner loop, all clusters in the delta sequence
BEGIN
    -- Loop through all clusters in this sequence
    FOR f1 IN
        SELECT geom FROM cluster WHERE study_area_id = a_id AND seq =
a_seq
    LOOP
        -- Loop through all clusters in the delta sequence
        FOR f2 IN
            SELECT geom FROM cluster WHERE study_area_id = a_id AND
seq = a_seq + a_delta
        LOOP
            -- If one of them intersect then return 1
            IF ST_Intersects(f1.geom, f2.geom) THEN
                RETURN 1;
            END IF;
        END LOOP;
    END LOOP;
    -- Return 0 if none overlap
    RETURN 0;
END;
$$ LANGUAGE plpgsql;

-- Creates a hexagonal grid (beehive) that fills the specified study
area and returns it as a set of geometries

```

```

CREATE OR REPLACE FUNCTION study_area_as_hex_quadrat(a_id integer)
RETURNS TABLE (id integer, geom geometry) AS $$
DECLARE
    fl record; -- Stores data on the study area
    a_cx double precision; -- x coordinate of the centre of a hexagon
    a_cy double precision; -- y coordinate of the centre of a hexagon
    a_x double precision; -- x coordinate of a vertex in a hexagon
    a_y double precision; -- y coordinate of a vertex in a hexagon
    a_quadrat geometry; -- Geometry object of a hexagon
    a_width double precision; -- Width of all hexagons
    a_height double precision; -- Height of all hexagons
    a_vert double precision; -- Vertical spacing between adjacent
hexagons
    a_horiz double precision; -- horizontal spacing between adjacent
hexagons
    a_wkt text; -- Stores the WKT representation of a hexagon
    a_1st_point text; -- Stores the WKT representation of the first
point in a hexagon (needed to close the polygon)
    i integer := 0; -- Incremental counter which gives each hexagon an
id
    r integer := 0; -- Incremental row counter
    a_offset double precision; -- Offset to add to odd rows
BEGIN
    -- Get data on the study area, particularly the coordinates of the
two defining points of the bounding box (min and max x and y)
    FOR fl IN
        SELECT
ST_X(ST_StartPoint(ST_ExteriorRing(ST_Envelope(a.geom)))) AS min_x,
ST_Y(ST_StartPoint(ST_ExteriorRing(ST_Envelope(a.geom)))) AS min_y,
        ST_X(ST_PointN(ST_ExteriorRing(ST_Envelope(a.geom)), 3)) AS
max_x, ST_Y(ST_PointN(ST_ExteriorRing(ST_Envelope(a.geom)), 3)) AS
max_y,
        a_quadrat_size, a.geom FROM study_area a WHERE a.id = a_id
    LOOP
        -- Formulas for this function where taken from Hexagonal Grids
by Red Blob Games (http://www.redblobgames.com/grids/hexagons)
        -- The length of a side of a hexagon is equal to the radius of
the bounding circle (quadrat_size)
        -- These are 'pointy-topped' hexagons
        -- The height of the hexagon is equal to the length of a side
times 2
        a_height := fl.quadrat_size * 2;
        -- The width of the hexagon is equal to the square root of 3
times the height divided by 2
        a_width := (|/ 3) * a_height / 2;
        -- The vertical spacing between adjacent hexagons is 3/4 the
height
        a_vert := 0.75 * a_height;
        -- The horizontal spacing between adjacent hexagons is the
width of a hexagon
        a_horiz := a_width;
        -- Start at the lower corner of the bounding box
        a_cy := fl.min_y;
        -- Repeat while the centre of the hexagon is within the
bounding box plus an offset
        WHILE a_cy < fl.max_y + (a_height / 2) LOOP
            -- If the row is odd then add an offset to the centre of
the x coordinate
            IF r % 2 = 0 THEN
                a_offset := 0;
            ELSE

```

```

        a_offset := a_width / 2;
    END IF;
    a_cx := fl.min_x + a_offset;
    -- Repeat while the centre of the hexagon is within the
    bounding box plus an offset
    WHILE a_cx < fl.max_x + (a_width / 2) LOOP
        -- Increase hexagon count
        i := i + 1;
        -- Build WKT for the hexagon (POLYGON)
        a_wkt := 'POLYGON(';
        -- Vertices of 'pointy topped' hexagons start at 30°
        up to 330° in increases of 60°
        FOR angle IN 30..330 BY 60 LOOP
            -- Calculate position of the vertex
            a_x := a_cx + (fl.quadrat_size * cos(pi() * angle
/ 180));
            a_y := a_cy + (fl.quadrat_size * sin(pi() * angle
/ 180));

            -- Add to WKT
            a_wkt := a_wkt || a_x || ' ' || a_y || ',';
            -- Store WKT of the first point (needed to close
the polygon)
            IF angle = 30 THEN
                a_1st_point := a_x || ' ' || a_y || '))';
            END IF;
        END LOOP;
        -- Add WKT of first point to close the polygon
        a_wkt := a_wkt || a_1st_point;
        -- Create geometry object
        a_quadrat := ST_GeomFromText(a_wkt, 3857);
        -- If hexagon is completely within the study area
        include it in the results
        IF ST_Within(a_quadrat, fl.geom) THEN
            RETURN QUERY SELECT i, a_quadrat;
        END IF;
        -- Move to next hexagon
        a_cx := a_cx + a_horiz;
    END LOOP;
    -- Move to next row
    a_cy := a_cy + a_vert;
    -- Increase row count
    r := r + 1;
END LOOP;
END LOOP;
END;
$$ LANGUAGE plpgsql;

-- Calculates the number of photos per quadrat
CREATE OR REPLACE FUNCTION quadrat_count(a_id integer, a_seq integer)
RETURNS TABLE (id integer, count bigint, geom geometry) AS $$
DECLARE
    fl record; -- Loop, each quadrat in the study area
    tag integer; -- Id of the tag on the study area
    tf_from timestamp; -- Lower boundary of the timeframe
    tf_unit interval; -- Time unit
    tf_to timestamp; -- Upper boundary of the timeframe
    min_acc smallint; -- Minimum accuracy
BEGIN
    -- Retrieve information from the study area to filter the photo
    dataset

```

```
SELECT a.tag_id, a.timeframe_from, a.time_unit, a.min_accuracy
FROM study_area a
WHERE a.id = a_id INTO tag, tf_from, tf_unit, min_acc;
-- Calculate the lower and upper boundaries of the timeframe
tf_from := tf_from + (a_seq * tf_unit);
tf_to := tf_from + tf_unit - interval '1 second';
-- Loop through each quadrat in the study area
FOR f1 IN
    SELECT * FROM study_area_as_hex_quadrat(a_id)
LOOP
    -- Return the number photos that are within each quadrat
    RETURN QUERY SELECT f1.id, COUNT(*) AS count, f1.geom
    FROM photo a INNER JOIN photo_tag b ON a.id = b.photo_id INNER
JOIN filtered_photo c ON a.id = c.id
    WHERE ST_Within(a.geom, f1.geom) AND b.tag_id = tag AND
a.date_taken BETWEEN tf_from AND tf_to AND a.accuracy >= min_acc;
END LOOP;
END;
$$ LANGUAGE plpgsql;
```

Appendix B – SQL script to remove the database objects

```
-----  
-- DROPS -----  
-----
```

```
DROP VIEW cluster_summary;  
DROP VIEW photo_reduced;  
DROP VIEW photo_summary;  
DROP VIEW study_area_summary;  
DROP VIEW photos_on_study_area;
```

```
-----  
DROP FUNCTION cluster_neighbors(integer, integer);  
DROP FUNCTION clusters_overlap(integer, integer, integer);  
DROP FUNCTION quadrat_count(integer);  
DROP FUNCTION reduce_photos(integer);  
DROP FUNCTION study_area_as_hex_quadrat(integer);  
-----
```

```
DROP TABLE cluster_photo;  
DROP TABLE cluster;  
DROP TABLE study_area;  
DROP TABLE filtered_photo;  
DROP TABLE grabbed_photo;  
DROP TABLE photo_tag;  
DROP TABLE data_grab;  
DROP TABLE tag;  
DROP TABLE photo;
```

Appendix C – PHP code for the main web pages

Code for manual.php and license.php are not listed here for simplicity.

index.php

```
<!DOCTYPE html>
<!--
This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program. If not, see <http://www.gnu.org/licenses/>.
-->
<!--
Author      : Juan Luis Rodas Rivera
Copyright   : (c) 2014, Juan Luis Rodas Rivera and The University
of Edinburgh
-->
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-
scale=1" />
    <title>Control Panel - Tempo-spatial explorer of Flickr
tags</title>
    <link rel="icon" type="image/png" href="img/map.png" />
    <link rel="stylesheet" href="css/bootstrap.css"
type="text/css" />
    <link rel="stylesheet" href="css/bootstrap-theme.css"
type="text/css" />
    <link rel="stylesheet" href="css/index.css" type="text/css" />
    <script src="js/jquery-2.1.1.js"
type="text/javascript"></script>
    <script src="js/bootstrap.js" type="text/javascript"></script>
    <script src="js/index.js" type="text/javascript"></script>
  </head>
  <body data-spy="scroll" data-target="#navbar" data-offset="80">
    <!-- Navigation bar -->
    <div id="navbar" class="navbar navbar-default navbar-fixed-
top" role="navigation">
      <div class="container">
        <div class="navbar-header">
          <button type="button" class="navbar-toggle" data-
toggle="collapse" data-target=".navbar-collapse">
            <span class="sr-only">Toggle navigation</span>
            <span class="icon-bar"></span>
            <span class="icon-bar"></span>
            <span class="icon-bar"></span>
          </button>

```

```

        <a class="navbar-brand" href="#"><span
class='glyphicon glyphicon-globe'></span> Tempo-spatial explorer of
Flickr tags</a>
    </div>
    <div class="collapse navbar-collapse">
        <ul class="nav navbar-nav">
            <li class="active"><a href="#tags">Current
tags</a></li>
            <li><a href="#add_tag">Add a tag</a></li>
            <li><a href="#study_areas">Current study
areas</a></li>
            <li><a href="#add_study_area">Add a study
area</a></li>
            <li><a href="manual.php">Manual</a></li>
            <li><a href="license.php">License</a></li>
        </ul>
    </div>
</div>
</div>
<!-- Container for delete confirmation -->
<div class="modal fade" id="delete_confirmation" tabindex="-1"
role="dialog" aria-labelledby="modalLabel" aria-hidden="true">
    <div class="modal-dialog modal-sm">
        <div class="modal-content">
            <div class="modal-header">
                <button type="button" class="close" data-
dismiss="modal"><span aria-hidden="true">&times;</span><span
class="sr-only">Close</span></button>
                <h4 class="modal-title"
id="modalLabel">Confirm deletion</h4>
            </div>
            <div class="modal-body">
                <p>Are you sure you want to delete:</p>
                <h5>
                    <span id="del_type" class="label label-
info"></span>
                    #<span id="del_id"></span>
                    <small id="del_desc"></small>
                </h5>
            </div>
            <div class="modal-footer">
                <button type="button" class="btn btn-default"
data-dismiss="modal">No</button>
                <button type="button" class="btn btn-danger"
data-dismiss="modal" onclick="del_db();">Delete</button>
            </div>
        </div>
    </div>
</div>
</div>
<!-- Main body container -->
<div class="container">
    <?php
require_once 'include/vars.php';
require_once 'Phlickr/Api.php';

// Create Phlickr object
$api = new Phlickr_Api(Flicker_API_KEY, Flicker_API_SECRET);
// Test connectivity with Flickr
try {
    $rsp = $api->ExecuteMethod('flickr.test.echo',
array('foo'=>'bar'));

```



```

// Check if photos for this tag have
been reduced

if ($row[4] == 1) {
    $reduced = "Yes";
}
// Assign number of reduced photos
if ($row[4] == 1 || $row[2] == 0) {
    $n_reduced = "<button
type='button' class='btn btn-sm btn-default btn-block hidden' data-
loading-text='Please wait...' onclick='reduce_data({$row[0]})';"><span
class='glyphicon glyphicon-screenshot'></span> Reduce</button><span>"
. $row[5] . "</span>";
}
// Print row
echo "<tr id='tag_{$row[0]}'><td
class='text-right'>{$row[0]}</td><td>{$row[1]}</td><td class='text-
right'>{$row[2]}</td><td class='text-right'>{$row[3]}</td><td
id='pu_{$row[0]}'><button type='button' class='btn btn-sm btn-default
btn-block' data-loading-text='Please wait...'
onclick='update_tag({$row[0]})';"><span class='glyphicon glyphicon-
refresh'></span></button></td><td>{$reduced}</td><td class='text-
right' id='rd_{$row[0]}'>{$n_reduced}</td>",
    "<td><div class='btn-
group'><button class='btn btn-sm btn-default'
disabled='disabled'><span class='glyphicon glyphicon-download-
alt'></span></button><a class='btn btn-sm btn-default'
href='{ $geoserver }/geoserver/sde/ows?service=WFS&version=1.0.0&request
=GetFeature&typeName=sde:photo_reduced&outputFormat=SHAPE-
ZIP&cql_filter=tag_id={$row[0]}' target='_blank'>Reduced</a><a
class='btn btn-sm btn-default'
href='{ $geoserver }/geoserver/sde/ows?service=WFS&version=1.0.0&request
=GetFeature&typeName=sde:photo&outputFormat=SHAPE-
ZIP&cql_filter=tag_id={$row[0]}'
target='_blank'>Complete</a></div></td>",
    "<td><button type='button'
class='btn btn-sm btn-default btn-block' onclick='del(\"Tag\",
{$row[0]}, \"{$row[1]}\");'"><span class='glyphicon glyphicon-
trash'></span></button></td></tr>", PHP_EOL;
}
}
?>
</tbody>
</table>
<div class="panel-footer">
    <div class="row">
        <div class="col-md-5 text-left">
            <small>Note: Updating a tag may take a
long time, please don't reload the page.</small>
        </div>
        <div class="col-md-7 text-right">
            <div class="progress hidden">
                <div id="update_progress"
class="progress-bar progress-bar-info progress-bar-striped"
role="progressbar" aria-valuenow="0" aria-valuemin="0" aria-
valuemax="100" style="width: 0%">
                    <span>0%</span>
                </div>
            </div>
        </div>
    </div>
</div>
</div>

```

```

</div>
<!-- Add tag section -->
<div id="add_tag" class="panel panel-default">
  <div class="panel panel-heading">
    <h2 class="panel-title"><span class='glyphicon
glyphicon-tag'></span> Add a tag</h2>
  </div>
  <div class="panel-body">
    <form action="action/ww_tag.php" method="post"
role="form" class="form-horizontal">
      <div class="form-group">
        <label for="tag_name" class="col-sm-1
control-label">Tag</label>
        <div class="col-sm-5">
          <input type="text" name="tag"
id="tag_name" class="form-control" maxlength="50" required="required"
placeholder="Enter the tag to add" />
        </div>
      </div>
      <div class="form-group">
        <div class="col-sm-offset-1 col-sm-5">
          <button type="submit" class="btn btn-
default"><span class='glyphicon glyphicon-plus'></span> Add</button>
        </div>
      </div>
    </form>
  </div>
</div>
<!-- List of study areas section -->
<div id="study_areas" class="panel panel-default">
  <div class="panel panel-heading">
    <h2 class="panel-title"><span class='glyphicon
glyphicon-map-marker'></span> Current study areas</h2>
  </div>
  <div class="panel-body">
    <p>This is the list of study areas defined in the
tool for data exploration.</p>
    <form action="viewer.php" method="post"
id="sa_action" role="form">
      <input type="hidden" name="id" id="sa_id" />
      <input type="hidden" name="id2" id="sa_id2" />
      <input type="hidden" name="mode" id="sa_mode"
/>
    </form>
  </div>
  <table class="table table-striped table-condensed">
    <thead>
      <tr>
        <th class='text-
right'>#</th><th>Description</th><th>Tag</th><th class='text-
right'>Quadrat size</th><th class='text-right'>Min. accuracy</th><th
class='text-right'>eps</th><th class='text-right'>Min.
pts.</th><th>Time unit</th><th class='text-right'>No. time
units</th><th class='text-right'>Timeframe</th><th class='text-
right'>Last update</th><th class='text-right'>Find
clusters</th><th>Explore</th><th>Edit</th><th>Delete</th>
      </tr>
    </thead>
    <tbody>
      <?php
// Retrieve list of study areas

```

```

        $result2 = pg_query($conn, "SELECT * FROM
study_area_summary ORDER BY name, description;");
        if (pg_num_rows($result2) == 0) {
            echo "<tr><td class='text-right'>-
</td><td>Empty</td><td>-</td><td class='text-right'>-</td><td
class='text-right'>-</td><td class='text-right'>-</td><td class='text-
right'>-</td><td>-</td><td class='text-right'>-</td><td class='text-
right'>-</td><td class='text-right'>-</td><td class='text-right'>-
</td><td>-</td><td>-</td><td>-</td></tr>", PHP_EOL;
        } else {
            // Populate list
            while ($row = pg_fetch_row($result2)) {
                // Check if clusters have been
calculated and get number of clusters
                $clusters = "<button type='button'
class='btn btn-sm btn-default btn-block' data-loading-text='Please
wait...' onclick='calculate_clusters({$row[0]});'><span
class='glyphicon glyphicon-record'></span></button>";
                if ($row[13] >= 0) {
                    $clusters = $row[13];
                }
                // Print row
                echo "<tr id='study_area_{$row[0]}'
class='study_area_tag_{$row[2]}'><td class='text-
right'>{$row[0]}</td><td>{$row[1]}</td><td>{$row[3]}</td><td
class='text-right'>{$row[4]}</td><td class='text-
right'>{$row[5]}</td><td class='text-right'>{$row[6]}</td><td
class='text-right'>{$row[7]}</td><td>{$row[8]}</td><td class='text-
right'>{$row[9]}</td><td class='text-right'>{$row[10]} <span
class='label label-info'>to</span> {$row[11]}</td><td class='text-
right'>{$row[12]}</td><td class='text-right'
id='cl_{$row[0]}'>{$clusters}</td><td><button type='button' class='btn
btn-sm btn-default btn-block' onclick='explore({$row[0]});'><span
class='glyphicon glyphicon-globe'></span></button></td><td><button
type='button' class='btn btn-sm btn-default btn-block'
onclick='edit({$row[0]});'><span class='glyphicon glyphicon-
pencil'></span></button></td><td><button type='button' class='btn btn-
sm btn-default btn-block' onclick='del(\"Study area\", {$row[0]},
\"{$row[1]}\");'><span class='glyphicon glyphicon-
trash'></span></button></td></tr>", PHP_EOL;
            }
        }
    ?>
</tbody>
</table>
</div>
<!-- Add study area section -->
<div id="add_study_area" class="panel panel-default">
    <div class="panel panel-heading">
        <h2 class="panel-title"><span class='glyphicon
glyphicon-map-marker'></span> Add study area</h2>
    </div>
    <div class="panel-body">
        <form role="form" class="form-horizontal">
            <div class="form-group">
                <label for="sa_tag" class="col-sm-2
control-label">For tag</label>
                <div class="col-sm-5">
                    <select name='tag' id='sa_tag'
class="form-control" required='required' >
                        <?php

```

```

// Fill select with the tag names
$result3 = pg_query($conn, 'SELECT *
FROM tag ORDER BY name;');
while ($row = pg_fetch_row($result3))
{
    echo "<option
value='{ $row[0] }'>{ $row[1] }</option>", PHP_EOL, "\t\t";
}
?>
</select>
</div>
</div>
<div class="form-group">
    <div class="col-sm-offset-2 col-sm-5">
        <button type="button" class="btn btn-
default" onclick="add();"><span class='glyphicon glyphicon-
plus'></span> Add</button>
    </div>
</div>
<div class="form-group">
    <span class="col-sm-2 label label-info">Or
copy geometry from</span>
</div>
<div class="form-group">
    <label for="sa_copy_id" class="col-sm-2
control-label">Study area</label>
    <div class="col-sm-5">
        <select name='id' id='sa_copy_id'
class="form-control" required='required' >
            <?php
// Fill select with the study areas
$result4 = pg_query($conn, 'SELECT
a.id, a.description, b.name FROM study_area a INNER JOIN tag b ON
a.tag_id = b.id ORDER BY a.description, b.name;');
while ($row = pg_fetch_row($result4))
{
    echo "<option
value='{ $row[0] }'>{ $row[1] } ({ $row[2] })</option>", PHP_EOL, "\t\t";
}
pg_close($conn);
?>
</select>
</div>
</div>
<div class="form-group">
    <div class="col-sm-offset-2 col-sm-5">
        <button type="button" class="btn btn-
default" onclick="copy();"><span class='glyphicon glyphicon-
export'></span> Copy</button>
    </div>
</div>
</form>
</div>
</div>
</div>
<!-- Footer -->
<?php include 'include/footer.php' ; ?>
</body>
</html>

```

viewer.php

```

<?php if (!array_key_exists("mode", $_POST)) { header("Location:
index.php"); die(); } ?>
<!DOCTYPE html>
<!--
This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program. If not, see <http://www.gnu.org/licenses/>.
-->
<!--
    Author      : Juan Luis Rodas Rivera
    Copyright   : (c) 2014, Juan Luis Rodas Rivera and The University
of Edinburgh
-->
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-
scale=1" />
    <title>Study Area Explorer | Tempo-spatial explorer of Flickr
tags</title>
    <link rel="icon" type="image/png" href="img/map.png" />
    <link rel="stylesheet"
href="http://ol3js.org/en/master/css/ol.css" type="text/css" />
    <link rel="stylesheet" href="css/bootstrap.css"
type="text/css" />
    <link rel="stylesheet" href="css/bootstrap-theme.css"
type="text/css" />
    <link rel="stylesheet" href="css/viewer.css" type="text/css"
/>
    <!-- script src="http://ol3js.org/en/master/build/ol.js"
type="text/javascript"></script-->
    <script src="js/ol-whitespace.js"
type="text/javascript"></script>
    <script src="js/jquery-2.1.1.js"
type="text/javascript"></script>
    <script src="js/bootstrap.js" type="text/javascript"></script>
  </head>
  <body>
    <!-- Navigation bar -->
    <div id="navbar" class="navbar navbar-default navbar-fixed-
top" role="navigation">
      <div class="container">
        <div class="navbar-header">
          <button type="button" class="navbar-toggle" data-
toggle="collapse" data-target=".navbar-collapse">
            <span class="sr-only">Toggle navigation</span>
            <span class="icon-bar"></span>
            <span class="icon-bar"></span>
            <span class="icon-bar"></span>
          </button>
        </div>
      </div>
    </div>
  </body>
</html>

```

```

        </button>
        <a class="navbar-brand" href="#"><span
class='glyphicon glyphicon-globe'></span> Tempo-spatial explorer of
Flickr tags</a>
    </div>
    <div class="collapse navbar-collapse">
        <ul class="nav navbar-nav">
            <li><a href="index.php#study_areas">Control
panel</a></li>
            <li class="active"><a href="#">Map
explorer</a></li>
            <li><a
href="manual.php#explore">Manual</a></li>
            <li><a href="license.php">License</a></li>
        </ul>
    </div>
</div>
</div>
<!-- Main body container -->
<div class="container">
    <?php
require_once 'include/study_area.php';
require_once 'include/tag.php';

// Retrieve mode to initialise the viewer
$mode = htmlspecialchars($_POST["mode"]);
?>
    <!-- Page header -->
    <div class='page-header'>
        <h4><span class='glyphicon glyphicon-map-
marker'></span>&nbsp;<?php
// Print page header depending on the mode
switch ($mode) {
    case "add":
        $obj = new
tag(htmlspecialchars($_POST["id"]));
        echo "<small>Add</small> study area <small>for
tag</small> {<small>{$obj->getName()}";
        break;
    case "explore":
    case "edit":
        $obj = new
study_area(htmlspecialchars($_POST["id"]));
        echo ($mode === "explore") ? "#{"<small>{$obj->getId()}
| <small>Study area</small> {"<small>{$obj->getDescription()} <small>for
tag</small> {"<small>{$obj->getTag()->getName()} <small>{"<small>{$obj->getFrom()-
>format("Y-m-d H:i:s")} to {"<small>{$obj->getTo()->format("Y-m-d
H:i:s")}</small>"} : "<small>Edit</small> study area #{"<small>{$obj->getId()}
<small>for tag</small> {"<small>{$obj->getTag()->getName()}";
        break;
    case "copy":
        $tag = new
tag(htmlspecialchars($_POST["id2"]));
        $sa = new
study_area(htmlspecialchars($_POST["id"]));
        // Duplicate study area
        $obj = $sa->copy($tag);
        $obj->store();
        echo "<small>Copy</small> study area #{"<small>{$sa-
>getId()} <small>for tag</small> {"<small>{$tag->getName()}";
        // Change mode to edit

```

```

        $mode = "edit";
        break;
    } ?>
</h4>
</div>
<?php

/**
 * Prints the form to add or edit an study area
 * @param integer $tag_id The unique identifier of the tag
 * @param string $desc The description of the study area
 * @param real $squad_size The quadrat size of the study
area
 * @param integer $min_acc The minimum accuracy for photos
in this study area
 * @param real $seps The search radius for the DBSCAN
algorithm
 * @param integer $min_pts The minimum number of points to
create a cluster on the DBSCAN algorithm
 * @param string $from The starting date of the timeframe
 * @param string $time_unit The time unit of the timeframe
 * @param integer $time_units The number of time units in
the timeframe
 */
function printStudyAreaForm($tag_id = 0, $desc = '',
$squad_size = 50000, $min_acc = 16, $seps = 500000, $min_pts = 5, $from
= '2010-01-01', $time_unit = '1 month', $time_units = '12') {
    echo "<!-- Study area form -->", PHP_EOL, "<div
class='panel panel-default'><div class='panel-body'>";
    // Prints the opening tag of the form, depending on
the action taken by the user, adding or creating a study area
    echo ($tag_id > 0) ? "<form
action='action/ww_study_area.php' id='add_study_area' method='POST'
class='form-horizontal' role='form'>" : "<form class='form-horizontal'
role='form'>";
    // Prints the inputs used to define the study area
    echo "<div class='form-group'><label
for='sa_description' class='col-sm-3 control-
label'>Description</label>";
    echo "<div class='col-sm-3'><input type='text'
name='description' id='sa_description' class='form-control'
maxlength='50' value='{ $desc}' required='required' placeholder='Enter
a description for the study area' /></div>";
    echo "<label for='sa_quadrat_size' class='col-sm-1
control-label'>Quadrat size</label>";
    echo "<div class='col-sm-2'><input type='number'
name='quadrat_size' id='sa_quadrat_size' class='form-control' min='1'
max='999999999' step='0.01' value='{ $squad_size}' required='required'
/></div>";
    echo "<label for='sa_min_accuracy' class='col-sm-
1'>Minimum accuracy</label>";
    echo "<div class='col-sm-2'><input type='number'
name='min_accuracy' id='sa_min_accuracy' class='form-control' min='1'
max='16' step='1' value='{ $min_acc}' required='required'
/></div></div>";
    // Prints the inputs that contain the DBSCAN
clustering algorithm parameters
    echo "<div class='form-group'><span class='col-sm-2
label label-info'>DBSCAN clustering parameters</span><label
for='sa_eps' class='col-sm-1 control-label'>eps</label>";

```



```

        echo "<div class='col-sm-2'><input type='number'
name='eps' id='sa_eps' class='form-control' min='1' max='999999999'
step='0.01' value='{ $eps}' required='required' /></div>";
        echo "<label for='sa_min_pts' class='col-sm-2 control-
label'>min pts</label>";
        echo "<div class='col-sm-2'><input type='number'
name='min_pts' id='sa_min_pts' class='form-control' min='2'
max='999999999' step='1' value='{ $min_pts}' required='required'
/></div></div>";
        // Prints the inputs that contain the timeframe
definition
        echo "<div class='form-group'><span class='col-sm-2
label label-info'>Timeframe definition</span><label
for='sa_timeframe_from' class='col-sm-1 control-label'>From</label>";
        echo "<div class='col-sm-3'><input type='text'
name='timeframe_from' id='sa_timeframe_from' class='form-control'
value='{ $from}' maxlength='10' required='required' /></div>";
        echo "<label for='sa_time_unit' class='col-sm-1
control-label'>Time unit</label>";
        echo "<div class='col-sm-2'><input type='text'
name='time_unit' id='sa_time_unit' class='form-control' maxlength='50'
value='{ $time_unit}' required='required' /></div>";
        echo "<label for='sa_time_units' class='col-sm-1
control-label'>No. time units</label>";
        echo "<div class='col-sm-2'><input type='number'
name='time_units' id='sa_time_units' class='form-control' min='1'
max='999999999' step='1' value='{ $time_units}' required='required'
/></div></div>";
        echo "<input id='sa_geom' type='hidden' name='geom'
/>";
        echo (" $tag_id > 0 ) ? "<input type='hidden' name='tag'
value='{ $tag_id}' />" : "";
        // Prints a hidden input with seq=0 since there is no
navigation
        echo "<input type='hidden' id='seq' value='0' />";
    }

/**
 * Prints the quadrat count analysis dialogue
 */
function printQuadratCounts() { ?>
    <!-- Container for quadrat count analysis -->
    <div class='modal fade' id='quadrat_count' tabindex='-
1' role='dialog' aria-labelledby='modalLabel' aria-hidden='true'>
        <div class='modal-dialog modal-lg'>
            <div class='modal-content'>
                <div class='modal-header'>
                    <button type='button' class='close'
data-dismiss='modal'><span aria-hidden='true'>&times;</span><span
class='sr-only'>Close</span></button>
                    <h4 class='modal-title'
id='modalLabel'>Quadrat count analysis</h4>
                </div>
                <div class='modal-body'>
                    <div id='qc_container' class='panel
panel-primary'>
                        <!-- Results of quadrat count
analysis go here -->
                    </div>
                </div>
            </div>
        </div>
    </div>
}

```

```

        <button type='button' class='btn btn-
default' data-dismiss='modal'>Close</button>
    </div>
</div>
</div>
</div>
<?php }

// Print elements specific for each mode
switch ($mode) {
    case "add":
        // Print study area form with default values
        printStudyAreaForm($obj->getId());
        echo "<div class='form-group'><div class='col-sm-
offset-3 col-sm-1'><button type='button' class='btn btn-default'
onclick='store();' id='btn_store' disabled='disabled'><span
class='glyphicon glyphicon-save'></span> Store</button></div>";
        echo "<div class='col-sm-2'><button type='button'
class='btn btn-default' onclick='clearDrawings();'><span
class='glyphicon glyphicon-trash'></span> Clear
drawing</button></div></div></form></div></div>", PHP_EOL;
        break;
    case "edit":
        // Print study area form with the values from the
existing study area
        printStudyAreaForm(0, $obj->getDescription(),
$obj->getQuadratSize(), $obj->getMinAccuracy(), $obj->getEps(), $obj-
>getMinPts(), $obj->getFrom()->format("Y-m-d"), $obj->getTimeUnit(),
$obj->getTimeUnits());
        echo "<div class='form-group'><div class='col-sm-
offset-3 col-sm-2'><button type='button' class='btn btn-default'
onclick='saveEdits();'><span class='glyphicon glyphicon-save'></span>
Save edits</button></div></div></form></div></div>", PHP_EOL;
        break;
    case "explore":
        // Print quadrat count analysis dialogue
        printQuadratCounts();
        $conn = pg_connect(CONN_STRING);
        // Retrieve area of biggest cluster in this study
area
        $result1 = pg_query($conn, "SELECT
COALESCE(MAX(area), 0) FROM cluster_summary WHERE study_area_id =
{$obj->getId()}");
        $max_area = pg_fetch_result($result1, 0, 0);
        // Print time slider
        echo "<!-- Time slider -->", PHP_EOL, "<div
class='panel panel-default'><table id='navigator'><tbody>";
        $data = array();
        // Retrieve area of clusters for each time unit
        $result2 = pg_query($conn, "SELECT seq, area,
count FROM cluster_summary WHERE study_area_id = {$obj->getId()} ORDER
BY seq;");
        while($row = pg_fetch_row($result2)) {
            => $row[2];
            $data[$row[0]] = ['area' => $row[1], 'count'
            ]
        }
        pg_close($conn);
        // First row contains the bar chart
        $row1 = "<tr class='bars'>";
        // Second row contains the cluster count
        $row2 = "<tr>";

```

```

// Third row contains the year reference
$row3 = "<tr>";
// Colspan for row 3 elements
$k = 1;
// Get first year
$current_year = $obj->getFrom(0)->format('Y');
// Build rows by looping through all time units
for($i = 0; $i < $obj->getTimeUnits(); $i++) {
    // First row
    $row1 = $row1 . "<td id='seq_area_{ $i }'
onclick='move({ $i });'>";
    // If there are clusters in this time unit
    if (array_key_exists($i, $data)) {
        // Calculate relative size
        $pct_area = 10 * $data[$i]['area'] /
$max_area;

        // Print bar chart
        for($j = 0; $j < $pct_area; $j++) {
            $row1 = $row1 . "<img
src='img/bar.png' alt='bar' class='bar' />";
        }
        // Second row
        $row2 = $row2 . "<td id='seq_count_{ $i }'
onclick='move({ $i });'>{$data[$i]['count']}</td>";
        // If there are no clusters set default values
        } else {
            $row1 = $row1 . "<img src='img/blank.png'
alt='bar' class='bar' />";
            $row2 = $row2 . "<td id='seq_count_{ $i }'
onclick='move({ $i });'>&middot;</td>";
        }
        $row1 = $row1 . "</td>";
        // Get year of next time unit
        $new_year = $obj->getFrom($i + 1)-
>format('Y');

        // If different from current year
        if ($current_year !== $new_year) {
            // Add an extra CSS class when the cell is
the first or last in the row
            $extra_class = "";
            if ($row3 === "<tr>") {
                $extra_class = " first";
            } elseif ($i == $obj->getTimeUnits() - 1)
{
                $extra_class = " last";
            }
            // Third row
            $row3 = $row3 . "<td
class='year{$extra_class}' colspan='{ $k }'>{$current_year}</td>";
            // Reset colspan
            $k = 0;
            // Assign new current year
            $current_year = $new_year;
        }
        // Increase colspan
        $k++;
    }
}
// Print all rows of the time slider
echo $row1 . "</tr>", PHP_EOL, $row2 . "</tr>",
PHP_EOL, $row3 . "</tr>", PHP_EOL;
echo "</tbody></table>";

```

```

// Print the time slider controls
echo "<div class='panel-footer'><div
class='row'>";
    $max = $obj->getTimeUnits() - 1;
    echo "<div class='col-sm-4 text-
left'><small><input type='range' id='seq' min='0' max='{$max}'
step='1' value='0' class='hidden' />";
    echo "<span id='from'>{$obj->getFrom()->format("Y-
m-d H:i:s")}</span>";
    echo " <span class='label label-info'>to</span> ";
    echo "<span id='to'>{$obj->getTo(0)->format("Y-m-d
H:i:s")}</span></small></div>", PHP_EOL;
    echo "<div class='col-sm-4 text-center'><div
class='btn-group btn-group-xs'><button type='button' class='btn btn-
default' onclick='first();'><span class='glyphicon glyphicon-fast-
backward'></span></button>", PHP_EOL;
    echo "<button type='button' class='btn btn-
default' onclick='rewind();'><span class='glyphicon glyphicon-
backward'></span></button>", PHP_EOL;
    echo "<button type='button' class='btn btn-
default' onclick='prev();'><span class='glyphicon glyphicon-step-
backward'></span></button>", PHP_EOL;
    echo "<button type='button' class='btn btn-
default' onclick='next();'><span class='glyphicon glyphicon-step-
forward'></span></button>", PHP_EOL;
    echo "<button type='button' class='btn btn-
default' onclick='forward();'><span class='glyphicon glyphicon-
forward'></span></button>", PHP_EOL;
    echo "<button type='button' class='btn btn-
default' onclick='last();'><span class='glyphicon glyphicon-fast-
forward'></span></button></div></div>", PHP_EOL;
    echo "<div class='col-sm-4 text-right'><button
type='button' class='btn btn-xs btn-primary'
onclick='quadrat_counts();'>Quadrat count analysis</button></div>",
PHP_EOL;
    echo '</div></div></div>', PHP_EOL;
    break;
}
?>
<!-- Map container -->
<div class="panel panel-default">
    <!-- Map header -->
    <div class="panel-heading text-right">
        <div class="row">
            <div class="col-sm-2 text-left">
                <span class="label label-
primary">Viewer</span>
            </div>
            <?php if ($mode === "explore") { ?>
                <!-- Number of photos in time unit -->
                <div class="col-md-4 text-right">
                    <span class="label label-info">Photos
<span class="badge"><?php echo $obj->getNumPhotos(); ?></span></span>
                </div>
                <div class="col-md-2 text-left">
                    <span class="label label-info">This time
unit <span id="photo_count" class="badge"><?php echo $obj-
>getNumPhotos(0); ?></span></span>
                </div>
                <!-- Layer list -->
                <div class="col-sm-4 text-right">

```

```

        <div class="btn-group btn-group-xs" data-
toggle="buttons">
            <label class="btn btn-default active">
                Base<input type="checkbox"
id="layer0" checked="checked" />
            </label>
            <label class="btn btn-default active">
                Study area<input type="checkbox"
id="layer1" checked="checked" />
            </label>
            <label class="btn btn-default">
                Quadrats<input type="checkbox"
id="layer2" />
            </label>
            <label class="btn btn-default active">
                Clusters<input type="checkbox"
id="layer3" checked="checked" />
            </label>
            <label class="btn btn-default active">
                Photos<input type="checkbox"
id="layer4" checked="checked" />
            </label>
        </div>
    </div>
    <?php } ?>
</div>
</div>
<!-- Map -->
<div id="map" class="map">
    <!-- Popuo container for photo display -->
    <div id="popup" class="ol-popup">
        <div class="panel panel-default">
            <div class="panel-heading"><h3
class="panel-title"><span></span> <small></small></h3></div>
            <div id="popup-content"></div>
            <div class="panel-
footer"><small></small><button type='button' class='close' id='popup-
closer'><span aria-hidden='true'>&times;</span><span class='sr-
only'>Close</span></button></div>
        </div>
    </div>
</div>
</div>
<!-- Footer -->
<?php include 'include/footer.php'; echo PHP_EOL; ?>
<!-- Global variables needed by viewer.js -->
<script>
    /**
     * Mode in which the viewer was loaded
     * @type string
     */
    var mode = '<?php echo $mode; ?>';

    /**
     * Unique identifier of the study area on view
     * @type integer
     */
    var id = <?php echo ($mode != "add") ? $obj->getId() : -1;
?>;

```

```

    /**
     * Unique identifier of the tag of the study area on view
     * @type integer
     */
    var tag_id = <?php echo ($mode != "add") ? $obj->getTag()-
>getId() : $obj->getId(); ?>;

    /**
     * Geometry of the study area on view
     * @type string
     */
    var geom = '<?php echo ($mode != "add") ? str_replace(" ",
"%20", $obj->getGeomAsWKT()) : ""; ?>';

    /**
     * Minimum accuracy of the study area on view
     * @type integer
     */
    var min_acc = <?php echo ($mode != "add") ? $obj-
>getMinAccuracy() : -1; ?>;

    /**
     * Boundaries of the timeframe or time units used by the
time slider
     * @type array
     */
    var times = [<?php
switch ($mode) {
    // Build an array used by the time slider to define
the boundaries of each time unit in explore mode
    case "explore":
        for($i = 0; $i < $obj->getTimeUnits(); $i++) {
            echo "[" . $obj->getFrom($i)->format("Y-m-
d\TH:i:s\Z"), ", ", " ", $obj->getTo($i)->format("Y-m-d\TH:i:s\Z"), " "];
            if ($i < $obj->getTimeUnits() - 1) {
                echo ',';
            }
        }
        break;
    // Boundaries of the timeframe
    case "edit":
        echo "[" . $obj->getFrom()->format("Y-m-
d\TH:i:s\Z"), ", ", " ", $obj->getTo()->format("Y-m-d\TH:i:s\Z"), " "];
        break;
    // Default boundaries
    case "add":
        echo "['2000-01-01T00:00:00Z', '2100-01-
01T00:00:00Z']";
        break;
} ?>];

    /**
     * Address of host where geoserver is running
     * @type string
     */
    var geoserver = '<?php echo GEOSERVER; ?>';
</script>
<script src="js/viewer.js" type="text/javascript"></script>
</body>
</html>

```

Appendix D – JavaScript code for the main web pages

index.js

```
/*
 * This program is free software: you can redistribute it and/or
 * modify
 * it under the terms of the GNU General Public License as published
 * by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see
 * <http://www.gnu.org/licenses/>.
 */

/*
 * Scripting for index.php
 * Author      : Juan Luis Rodas Rivera
 * Copyright   : (c) 2014, Juan Luis Rodas Rivera and The University
 * of Edinburgh
 */

/**
 * Updates a tag checking for new photos on Flickr
 * @param {integer} tag The unique identifier of the tag
 */
function update_tag(tag) {
    // Get cell that called the action
    var caller_pu = "#pu_" + tag;
    // Get button that called the action
    var btn = $(caller_pu).find('button');
    // Set button to loading state
    btn.button('loading');
    // Reset progress bar
    set_progress(0);
    // Show progress bar
    $("#update_progress").parent().removeClass('hidden');
    // Initiate AJAX request to update photos of this tag
    $.post("action/ww_tag.php", {id: tag})
        // When done
        .done(function(data) {
            // Stop asking for update requests
            clearInterval(poll);
            // Max progress bar
            set_progress(100);
            // Reset button state
            btn.button('reset');
            // Update list of tags with data returned
            $(data.caller).prev().text(data.updated);
            $(data.caller).prev().prev().text(data.count);
            if ($(data.caller).next().text() === "Yes") {
                $(data.caller).next().text("No");
            }
        });
}
```

```

        $(data.caller).next().next().find('span').text('');
    }
    // Show data reduction button

$(data.caller).next().next().find('button').removeClass('hidden');
    // Hide progress bar
    $("#update_progress").parent().addClass('hidden');
    }, "json");
    // Initiate polling to check for progress of update process
    var poll = setInterval(function() {
        // Poll for progress update every 3333 ms
        $.post("action/ww_poll_tag_update.php", {tag: tag})
            // When done
            .done(function(data) {
                // Update progress bar
                set_progress(data.pct);
            }, "json");
    }, 3333);
}

/**
 * Sets the value of the progress bar
 * @param {real} pct The value to set
 */
function set_progress(pct) {
    var bar = $("#update_progress");
    // Set aria value
    bar.attr("aria-valuenow", pct);
    // Set text
    bar.first("span").text(pct + '%');
    // Set width
    bar.css("width", pct + '%');
}

/**
 * Reduces the set of photos for a tag
 * @param {integer} tag The unique identifier of the tag
 */
function reduce_data(tag) {
    // Get cell that called the action
    var caller_rd = "#rd_" + tag;
    // Get button that called the action
    var btn = $(caller_rd).find('button');
    // Set button to loading state
    btn.button('loading');
    // Initiate AJAX request to reduce photos of this tag
    $.post("action/ww_reduce_data.php", {tag: tag})
        // When done
        .done(function(data) {
            // Update list of tags with data returned
            $(data.caller).prev().text("Yes");
            $(data.caller).find('span').text(data.num);
            // Reset button state
            btn.button('reset');
            // Hide button
            btn.addClass('hidden');
        }, "json");
}

/**
 * Calculates clusters for a study area

```



```

* @param {integer} study_area The unique identifier of the study area
*/
function calculate_clusters(study_area) {
    // Get cell that called the action
    var caller_cl = "#cl_" + study_area;
    // Get button that called the action
    var btn = $(caller_cl).find('button');
    // Set button to loading state
    btn.button('loading');
    // Initiate AJAX request to calculate clusters of this study area
    $.post("action/ww_clusters.php", {study_area: study_area})
        // When done
        .done(function(data) {
            // Update list of study areas with data returned
            $(data.caller).text(data.num);
            // Reset button state
            btn.button('reset');
            // Hide button
            btn.addClass('hidden');
        }, "json");
}

/**
 * Sets POST parameters for the viewer and submits the form
 * @param {integer} sa_id The unique identifier to send
 * @param {string} action Mode in which the viewer will be initialised
 */
function goToViewer(sa_id, action) {
    // Set identifier
    $("#sa_id").val(sa_id);
    // Set mode
    $("#sa_mode").val(action);
    // Submit form
    $("#sa_action").submit();
}

/**
 * Go to viewer on explore mode
 * @param {integer} sa_id The unique identifier of the study area
 */
function explore(sa_id) {
    // Send identifier of the study area
    goToViewer(sa_id, "explore");
}

/**
 * Go to viewer on edit mode
 * @param {integer} sa_id The unique identifier of the study area
 */
function edit(sa_id) {
    // Send identifier of the study area
    goToViewer(sa_id, "edit");
}

/**
 * Go to viewer on add mode
 */
function add() {
    // Send identifier of the tag
    goToViewer($("#sa_tag").val(), "add");
}

```

```

/**
 * Go to viewer on copy mode
 */
function copy() {
    // Send identifier of the tag
    $("#sa_id2").val($("#sa_tag").val());
    // Send identifier of the study area
    goToViewer($("#sa_copy_id").val(), "copy");
}

/**
 * Shows the delete confirmation dialogue
 * @param {string} type Either study area or tag
 * @param {integer} id Unique identifier of the object to delete
 * @param {string} description The name or description of the object
to delete
 */
function del(type, id, description) {
    // Sets the values on the delete confirmation dialogue
    $("#del_type").text(type);
    $("#del_id").text(id);
    $("#del_desc").text(description);
    // Shows the delete confirmation dialogue
    $("#delete_confirmation").modal();
}

/**
 * Deletes an object from the database
 */
function del_db() {
    // Checks which object to delete
    if ($("#del_type").text() === "Tag") {
        // Removes the tag from the list
        $("#tag_" + $("#del_id").text()).remove();
        // Removes all related study areas from the list
        $(".study_area_tag_" + $("#del_id").text()).remove();
    } else {
        // Removes the study area from the list
        $("#study_area_" + $("#del_id").text()).remove();
    }
    // Initiates AJAX request to delete the selected object from the
database
    $.post("action/ww_delete.php", {type: $("#del_type").text(), id:
$("#del_id").text()});
}

```

viewer.js

```

/*
 * This program is free software: you can redistribute it and/or
 modify
 * it under the terms of the GNU General Public License as published
 by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see
 <http://www.gnu.org/licenses/>.
 */

/*
 * Scripting for viewer.php
 * Author      : Juan Luis Rodas Rivera
 * Copyright   : (c) 2014, Juan Luis Rodas Rivera and The University
 of Edinburgh
 */

/*****
 * Section: Styles
 */

*****/

/**
 * Style cache for points in the photo layer
 * @type array
 */
var styleCache = {};

/**
 * Style function for points in the photo layer
 * @param {ol.Feature} feature
 * @param {real} resolution
 * @returns {ol.style.Style}
 */
var styleFunction = function(feature, resolution) {
  // Set radius of point according to the resolution
  var radius = 3;
  if (resolution <= 3500) {
    radius = 4;
  } else if (resolution <= 1500) {
    radius = 5;
  } else if (resolution <= 500) {
    radius = 6;
  }
  // Retrieve style from style cache
  var style = styleCache[radius];
  // If style does not exist create it
  if (!style) {

```

```

    // Create new style
    style = [new ol.style.Style({
      image: new ol.style.Circle({
        radius: radius,
        fill: new ol.style.Fill({
          color: 'rgba(255, 0, 0, 1)'
        }),
        stroke: new ol.style.Stroke({
          color: 'rgba(255, 255, 255, 1)',
          width: 1
        })
      })
    })];
    // Add it to style cache
    styleCache[radius] = style;
  }
  // Return the style
  return style;
};

/**
 * Style cache for polygons in the cluster layer
 * @type array
 */
var clusterStyleCache = {};

/**
 * Style function for polygons in the cluster layer
 * @param {ol.Feature} feature
 * @param {real} resolution
 * @returns {ol.style.Style}
 */
var clusterStyleFunction = function(feature, resolution) {
  // Get sequential delta from feature
  var delta = feature.get('seq_delta');
  // Retrieve style from style cache
  var style = clusterStyleCache[delta];
  // If style does not exist create it
  if (!style) {
    // Set colours and stroke according to the sequential delta
    var fill = 'rgba(255,255,191, 0.7)';
    var stroke = '#708090';
    var lineDash = undefined;
    switch(delta) {
      case -2:
        fill = 'rgba(215,25,28, 0.1)';
        stroke = '#d7191c';
        lineDash = [3, 6];
        break;
      case -1:
        fill = 'rgba(253,174,97, 0.1)';
        stroke = '#fdae61';
        break;
      case 1:
        fill = 'rgba(166,217,106, 0.1)';
        stroke = '#a6d96a';
        break;
      case 2:
        fill = 'rgba(26,150,65, 0.1)';
        stroke = '#1a9641';
        lineDash = [3, 6];
    }
  }
};

```

```

        break;
    }
    // Create new style
    style = [new ol.style.Style({
        fill: new ol.style.Fill({
            color: fill
        }),
        stroke: new ol.style.Stroke({
            color: stroke,
            width: 1,
            lineDash: lineDash
        }),
        zIndex: delta + 2
    })];
    // Add it to style cache
    clusterStyleCache[delta] = style;
}
// Return the style
return style;
};

/**
 * Style cache for polygons in the quadrat count layer
 * @type array
 */
var qcStyleCache = {};

/**
 * Style function for polygons in the quadrat count layer
 * @param {ol.Feature} feature
 * @param {real} resolution
 * @returns {ol.style.Style}
 */
var qcStyleFunction = function(feature, resolution) {
    // Set text parameters if mode is explore
    var count = 0;
    var text = '';
    var hasText = 0;
    if (mode === "explore") {
        count = feature.get('count');
        if (resolution < 3900) {
            text = count.toString();
            hasText = 1;
        }
    }
    // Retrieve style from style cache
    var style = qcStyleCache[count + "_" + hasText];
    // If style does not exist create it
    if (!style) {
        // Create new style
        style = [new ol.style.Style({
            fill: new ol.style.Fill({
                color: 'rgba(255,255,255, 0.15)'
            }),
            stroke: new ol.style.Stroke({
                color: 'rgba(186,85,211, 0.35)',
                width: 1.25
            }),
            text: new ol.style.Text({
                textAlign: 'center',
                textBaseline: 'hanging',

```

```

        font: 'normal 10px Arial',
        text: text,
        fill: new ol.style.Fill({color: 'rgba(75,0,130, 1)'}),
        stroke: new ol.style.Stroke({color: '#ffffff', width:
1}))
    })
  });
  // Add it to style cache
  qcStyleCache[count + "_" + hasText] = style;
}
// Return the style
return style;
};

/**
 * Style cache for polygons in the study area layer
 * @type array
 */
var saStyleCache = {};

/**
 * Style function for polygons in the study area layer
 * @param {ol.Feature} feature
 * @param {real} resolution
 * @returns {ol.style.Style}
 */
var saStyleFunction = function(feature, resolution) {
  // Retrieve style from style cache
  var style = saStyleCache[0];
  // If style does not exist create it
  if (!style) {
    // Create new style
    style = [new ol.style.Style({
      stroke: new ol.style.Stroke({
        color: "#6495ED",
        width: 2
      })
    })];
  }
  // Add it to style cache
  saStyleCache[0] = style;
}
// Return the style
return style;
};

/*****
*****
 * Section: Map elements
 *
*****
*****/

/**
 * Tile base layer for MapQuest OSM
 * @type ol.layer.Tile
 */
var baseLayer = new ol.layer.Tile({
  source: new ol.source.MapQuest({
    layer: 'osm'
  })
});

```

```
});  
  
/**  
 * GeoJSON source for the study area layer  
 * @type ol.source.GeoJSON  
 */  
var saSource;  
  
/**  
 * Vector layer for the study area  
 * @type ol.layer.Vector  
 */  
var saLayer;  
  
/**  
 * GeoJSON source for the quadrat count layer  
 * @type ol.source.GeoJSON  
 */  
var qcSource;  
  
/**  
 * Vector layer for the quadrat counts  
 * @type ol.layer.Vector  
 */  
var qcLayer;  
  
/**  
 * GeoJSON source for the cluster layer  
 * @type ol.source.GeoJSON  
 */  
var clusterSource;  
  
/**  
 * Vector layer for the clusters  
 * @type ol.layer.Vector  
 */  
var clusterLayer;  
  
/**  
 * GeoJSON source for the photo layer  
 * @type ol.source.GeoJSON  
 */  
var photoSource;  
  
/**  
 * Vector layer for the photos  
 * @type ol.layer.Vector  
 */  
var photoLayer;  
  
/**  
 * Feature overlay for study area drawing  
 * @type ol.FeatureOverlay  
 */  
var featureOverlay;  
  
/**  
 * Initial view of the map  
 * @type ol.View  
 */  
var view = new ol.View({
```

```

        center: [0, 0],
        zoom: 2
    });

/**
 * Select interaction to modify study areas
 * @type ol.interaction.Select
 */
var select;

/**
 * Modify interaction to modify study areas
 * @type ol.interaction.Modify
 */
var modify;

/**
 * Draw interaction to add a study area
 * @type ol.interaction.Draw
 */
var draw;

/**
 * Whether the study area has been modified by the user
 * @type integer
 */
modified = 0;

/**
 * Adds the relevant layers to the map
 * @param {boolean} firstLoad Whether layers are being loaded for the
first time
 */
function addLayers(firstLoad) {
    // Get sequential
    var seq = parseInt($("#seq").val());
    // If editing or loading for first time, and not adding
    if ((mode === "edit" || firstLoad) && mode !== "add") {
        // Define and add study area layer
        map.removeLayer(saLayer);
        saSource = new ol.source.GeoJSON({
            url: geoserver +
'/geoserver/sde/ows?service=WFS&version=1.0.0&request=GetFeature&typeName=sde:study_area&outputFormat=application/json'
            + '&cql_filter=id=' + id
        });
        saLayer = new ol.layer.Vector({
            source: saSource,
            style: saStyleFunction
        });
        map.addLayer(saLayer);
        // If editing
        if (mode === "edit") {
            // Define and add quadrat count layer
            map.removeLayer(qcLayer);
            qcSource = new ol.source.GeoJSON({
                url: geoserver +
'/geoserver/sde/ows?service=WFS&version=1.0.0&request=GetFeature&typeName=sde:study_area_quadrat&outputFormat=application/json'
                + '&viewparams=id:' + id
            });

```



```

        qcLayer = new ol.layer.Vector({
            source: qcSource,
            style: qcStyleFunction
        });
        map.addLayer(qcLayer);
        // Add select interaction to edit the study area
        map.removeInteraction(select);
        select = new ol.interaction.Select({
            layers: [saLayer]
        });
        map.addInteraction(select);
        // Add modify interaction to edit the study area
        map.removeInteraction(modify);
        modify = new ol.interaction.Modify({
            features: select.getFeatures()
        });
        map.addInteraction(modify);
        // Set listener to update variable when geometry is
changed
        var selected_features = select.getFeatures();
        selected_features.on('add', function(event) {
            var feature = event.element;
            feature.on('change', function(event) {
                modified = 1;
            });
        });
    }
}
// If exploring
if (mode === "explore") {
    // Define and add quadrat count layer
    map.removeLayer(qcLayer);
    qcSource = new ol.source.GeoJSON({
        url: geoserver +
'/geoserver/sde/ows?service=WFS&version=1.0.0&request=GetFeature&typeName=sde:quadrat_count&outputFormat=application/json'
        + '&viewparams=id:' + id + ';seq:' + seq
    });
    qcLayer = new ol.layer.Vector({
        source: qcSource,
        style: qcStyleFunction,
        visible: $("#layer2").is(":checked")
    });
    map.addLayer(qcLayer);
    // Define and add cluster layer
    map.removeLayer(clusterLayer);
    clusterSource = new ol.source.GeoJSON({
        url: geoserver +
'/geoserver/sde/ows?service=WFS&version=1.0.0&request=GetFeature&typeName=sde:cluster_neighbors&outputFormat=application/json'
        + '&viewparams=id:' + id + ';seq:' + seq
    });
    clusterLayer = new ol.layer.Vector({
        source: clusterSource,
        style: clusterStyleFunction,
        visible: $("#layer3").is(":checked")
    });
    map.addLayer(clusterLayer);
}
// Define and add photo layer
map.removeLayer(photoLayer);

```

```

// Set geometry condition to filter photos
var geomCondition = '%20AND%20WITHIN(geom,' + geom + ')';
// If adding do not use condition
if (mode === "add") {
  geomCondition = "";
}
photoSource = new ol.source.GeoJSON({
  url: geoserver +
'/geoserver/sde/ows?service=WFS&version=1.0.0&request=GetFeature&typeName=sde:photo_reduced&outputFormat=application/json'
  + '&cql_filter=tag_id=' + tag_id + '%20AND%20accuracy>=' +
min_acc + geomCondition
  + '%20AND%20date_taken%20DURING%20' + times[seq][0] + '/'
+ times[seq][1]
});
photoLayer = new ol.layer.Vector({
  source: photoSource,
  style: styleFunction,
  visible: ($("#layer4").is(":checked") || mode !== "explore")
});
map.addLayer(photoLayer);
// If adding and loading for the first time
if (mode === "add" && firstLoad) {
  // Define and add feature overlay for study area drawing
  featureOverlay = new ol.FeatureOverlay({
    style: new ol.style.Style({
      fill: new ol.style.Fill({
        color: 'rgba(255, 255, 255, 0.2)'
      }),
      stroke: new ol.style.Stroke({
        color: '#ffcc33',
        width: 2
      }),
      image: new ol.style.Circle({
        radius: 7,
        fill: new ol.style.Fill({
          color: '#ffcc33'
        })
      })
    })
  });
  featureOverlay.setMap(map);
  // Add modify interaction to edit the study area
  modify = new ol.interaction.Modify({
    features: featureOverlay.getFeatures()
  });
  map.addInteraction(modify);
  // Add draw interaction to add a study area
  draw = new ol.interaction.Draw({
    features: featureOverlay.getFeatures(),
    type: "Polygon"
  });
  map.addInteraction(draw);
  // When study area has been drawn
  draw.on('drawend', function() {
    // Remove draw interaction
    map.removeInteraction(draw);
    // Enable store button
    $("#btn_store").attr("disabled", false);
  });
}

```

```

    // If exploring bind layer visibility to layer list buttons
    if (mode === "explore") {
        map.getLayers().forEach(function(layer, i) {
            new ol.dom.Input($('#layer' + i)[0]).bindTo('checked',
layer, 'visible');
        });
    }
}

/*****
* Section: Interactivity
*
*****/

/**
* Switch layer visibility on layer list click
*/
$('#label.btn').on('click', function() {
    // Get selected layer
    var label = $(this);
    var checkbox = label.find('input');
    // Switch visibility
    checkbox.prop("checked", !checkbox.is(":checked"));
    // Fire visibility change
    var ev = document.createEvent('Event');
    ev.initEvent('change', true, false);
    checkbox[0].dispatchEvent(ev);
});

/**
* Set the background colours on the time slider
* @param {integer} seq Sequential of the time unit
*/
function setSliderColours(seq) {
    // Remove existing colours and add colour to present time unit
    $("#seq_area_0").parent().children().removeClass();
    $("#seq_area_" + seq).addClass("selected");
    $("#seq_count_0").parent().children().removeClass();
    $("#seq_count_" + seq).addClass("selected");
    // Send AJAX request to find out if there are overlapping clusters
on neighbouring time units
    $.post("action/ww_overlapping_clusters.php", {sa: id, seq:
seq}).done(function(data) {
        for (var i in data) {
            // Add colour class depending on time differential
            switch (data[i]) {
                case -2:
                    $("#seq_area_" +
seq).prev().prev().addClass("minus2");
                    $("#seq_count_" +
seq).prev().prev().addClass("minus2");
                    break;
                case -1:
                    $("#seq_area_" + seq).prev().addClass("minus1");
                    $("#seq_count_" + seq).prev().addClass("minus1");
                    break;
                case 1:
                    $("#seq_area_" + seq).next().addClass("plus1");

```

```

        $("#seq_count_" + seq).next().addClass("plus1");
        break;
    case 2:
        $("#seq_area_" +
seq).next().next().addClass("plus2");
        $("#seq_count_" +
seq).next().next().addClass("plus2");
        break;
    }
    }, "json");
}

/**
 * Set the photo count for a time unit
 * @param {integer} seq Sequential of the time unit
 */
function setPhotoCount(seq) {
    // Send AJAX request to find out the number of photos on a time
unit
    $.post("action/ww_photo_count.php", {sa: id, seq:
seq}).done(function(data) {
        $("#photo_count").text(data.count);
    }, "json");
}

/**
 * Set layers and display elements when the time unit changes
 */
$("#seq").change(function() {
    // Add layers
    addLayers(false);
    // Get current time unit
    var seq = $(this).val();
    // Display time boundaries of time unit
    $("#from").text(times[seq][0].substr(0,19).replace('T', ' '));
    $("#to").text(times[seq][1].substr(0,19).replace('T', ' '));
    // Change time slider colours
    setSliderColours(seq);
    // Set photo count
    setPhotoCount(seq);
});

/**
 * Resets the bar chart to standard colours
 */
function resetBars() {
    // Replace each modified bar for a standard bar
    $("#img.overlap").each(function(index){
        $(this).parent().append("<img src='img/bar.png' alt='bar'
class='bar' />");
        $(this).remove();
    });
}

/**
 * Changes the current time unit being displayed
 * @param {integer} seq The time unit to move to
 */
function move(seq) {
    // Change if different from current time unit

```

```

    if (parseInt($("#seq").val()) !== parseInt(seq)) {
        // Reset bar chart formatting
        resetBars();
        // Change value of current time unit
        $("#seq").val(seq);
        // Fire time unit change
        var el = document.getElementById('seq');
        var ev = document.createEvent('Event');
        ev.initEvent('change', true, false);
        el.dispatchEvent(ev);
    }
}

/**
 * Change the current time unit by a specified step
 * @param {integer} a_step Positive or negative number of steps to
move from the current time unit
 */
function step(a_step) {
    // Calculate new time unit
    var newIndex = parseInt($("#seq").val()) + a_step;
    var max = $("#seq").attr("max");
    // Check it falls within timeframe boundaries
    if (newIndex < 0) {
        newIndex = 0;
    } else if (newIndex > $("#seq").attr("max")) {
        newIndex = max;
    }
    // Move time unit to new time unit
    move(newIndex);
}

/**
 * Move to the next time unit
 */
function next() {
    step(1);
}

/**
 * Move to the previous time unit
 */
function prev() {
    step(-1);
}

/**
 * Move three time units ahead
 */
function forward() {
    step(3);
}

/**
 * Move three time units behind
 */
function rewind() {
    step(-3);
}

/**

```

```

    * Move to first time unit
    */
function first() {
    move(0);
}

/**
 * Move to last time unit
 */
function last() {
    move($("#seq").attr("max"));
}

/**
 * Save edits made to the study area by the user
 */
function saveEdits() {
    // Only save in edit mode
    if (mode === "edit") {
        // Get coordinates of drawn geometry
        var coords =
saSource.getFeatures()[0].getGeometry().getCoordinates()[0];
        // Build WKT
        geom = "POLYGON(";
        coords.forEach(function(val) {
            geom = geom + val[0] + " " + val[1] + ", ";
        });
        geom = geom.substr(0, geom.length - 2) + ")";
        // Set new minimum accuracy
        min_acc = $("#sa_min_accuracy").val();
        // Send AJAX request to update study area
        $.post("action/ww_study_area.php", {id: id, description:
$("#sa_description").val(),
            quadrat_size: $("#sa_quadrat_size").val(), min_accuracy:
min_acc,
            eps: $("#sa_eps").val(), min_pts: $("#sa_min_pts").val(),
from: $("#sa_timeframe_from").val() + " 00:00:00",
            time_unit: $("#sa_time_unit").val(), time_units:
$("#sa_time_units").val(), geom: geom, upd_geom: modified})
            // When done
            .done(function(data) {
                // Update layers
                addLayers(false);
            }, "json");
    }
}

/**
 * Clears the study area drawn by the user
 */
function clearDrawings() {
    // Clear feature drawn by user
    featureOverlay.getFeatures().clear();
    // Add back draw interaction
    map.addInteraction(draw);
    // Disable store button
    $("#btn_store").attr("disabled", true);
}

/**
 * Add a new study area

```

```

*/
function store() {
  // Only store in add mode
  if (mode === "add") {
    // Set filter for date
    var datePattern = /^2\d\d\d-(0?[1-9]|1[012])-(0?[1-9]|12)[0-9]|3[01])$/;
    // Check if all fields have been filled
    if ($("#sa_description").val().length > 0 &&
    $("#sa_time_unit").val().length > 0 &&
    datePattern.test($("#sa_timeframe_from").val())) {
      // Get coordinates of drawn geometry
      var coords =
featureOverlay.getFeatures().item(0).getGeometry().getCoordinates()[0]
;
      // Build WKT
      var geom = "POLYGON(";
      coords.forEach(function(val) {
        geom = geom + val[0] + " " + val[1] + ", ";
      });
      geom = geom.substr(0, geom.length - 2) + ")";
      // Set geometry field in form
      $("#sa_geom").val(geom);
      // Submit form
      $("#add_study_area").submit();
    } else {
      // Show message if fields have not been filled
      alert("Please fill all fields in the form.");
    }
  }
}

/**
 * Shows the quadrat count analysis dialogue
 */
function quadrat_counts() {
  // Send AJAX request to retrieve quadrat count analysis tables
  $.post("action/ww_quadrat_counts.php", {sa: id, seq:
parseInt($("#seq").val())})
  // When done
  .done(function(data) {
    // Set quadrat count analysis results on container
    document.getElementById("qc_container").innerHTML = data;
    // Show quadrat count analysis
    $("#quadrat_count").modal();
  }, "json");
}

/**
 * Popup container for photo preview
 * @type @exp;document@call;getElementById
 */
var container = document.getElementById('popup');

/**
 * Popup content for photo preview
 * @type @exp;document@call;getElementById
 */
var content = document.getElementById('popup-content');

/**

```

```
* Popup closer for photo preview
* @type @exp;document@call;getElementById
*/
var closer = document.getElementById('popup-closer');

/**
 * Hide photo preview container on closer click
 */
closer.onclick = function() {
  container.style.display = 'none';
};

/**
 * Overlay to display photo preview
 * @type ol.Overlay
 */
var overlay = new ol.Overlay({
  element: container
});

/**
 * Map element
 * @type ol.Map
 */
var map = new ol.Map({
  target: 'map',
  layers: [baseLayer],
  overlays: [overlay],
  view: view
});

/**
 * Scale line
 * @type ol.control.ScaleLine
 */
var scale_line = new ol.control.ScaleLine();

/**
 * Set the scale line on the map
 * @param {ol.Map} param
 */
scale_line.setMap(map);

/**
 * Fires interactions when the map is clicked
 * @param {type} pixel
 * @param {type} coordinate
 */
var mapClicked = function(pixel, coordinate) {
  var action = 'none';
  // Check for action and feature clicked
  var feature = map.forEachFeatureAtPixel(pixel, function(feature,
layer) {
    if (layer === photoLayer) {
      action = 'display_photo';
      return feature;
    } else if (layer === clusterLayer) {
      action = 'show_overlapping_clusters';
      // Only return cluster if exists in current time unit
      if (feature.get('seq_delta') === 0) {
        return feature;
      }
    }
  });
}
```



```

    }
  });
  // If a feature has been clicked
  if (feature) {
    // If clicked on a photo
    if (action === 'display_photo') {
      // Set photo preview overlay on the coordinates clicked
      overlay.setPosition(coordinate);
      // Set title of the photo overlay, Flickr id and user
      $('#popup-
content').prev().first('h3').find('span').text(feature.get('num'));
      $('#popup-
content').prev().first('h3').find('small').text(feature.get('usr'));
      // Load photo from Flickr
      content.innerHTML = "<img src='https://farm" +
feature.get('farm') + ".staticflickr.com/" +
      feature.get('server') + "/" + feature.get('num') +
      "_" + feature.get('secret') +
      "_m.jpg' alt='Preview' />";
      // Display photo preview container
      container.style.display = 'block';
      // Set date of the photo taken
      $('#popup-
content').next().find('small').text(feature.get('date_taken'));
      // If clicked on a cluster
    } else if (action === 'show_overlapping_clusters') {
      // Send AJAX request to retrieve other time units with
clusters that overlap this cluster
      $.post("action/ww_overlapping_clusters.php", {sa: id, id:
feature.get('id')})
      // When done
      .done(function(data) {
        // Clear styling on second row of time slider

$('#seq_count_0').parent().children().removeClass();
        // Reset bar chart
        resetBars();
        // Loop through each time unit with cluster that
overlap
        for (var a_id in data) {
          // Set cell background
          $("#seq_count_" +
data[a_id][0]).addClass("overlaps");
          // Replace standard bars
          for (var i = 0; i < data[a_id][1]; i++) {
            $("#seq_area_" + data[a_id][0] + "
img").first().remove();
            $("#seq_area_" +
data[a_id][0]).append("<img class='overlap' src='img/bar2.png'
alt='bar' />");
          }
        }
      }, "json");
    }
  }
};

/**
 * Set map to listen for clicks from the user
 */

```

```
map.on('click', function(evt) {
    mapClicked(evt.pixel, evt.coordinate);
});

/**
 * Execute once the DOM has finished loading
 */
$(window).load(function() {
    // Add layers
    addLayers(true);
    // If mode is explore set initial time slider colour
    if (mode === "explore") {
        setSliderColours(0);
    }
});
```

Appendix E – CSS code for the main web pages

index.css

```
/*
 * This program is free software: you can redistribute it and/or
 * modify
 * it under the terms of the GNU General Public License as published
 * by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see
 * <http://www.gnu.org/licenses/>.
 */

/*
 * Styling for index.php
 * Author      : Juan Luis Rodas Rivera
 * Copyright   : (c) 2014, Juan Luis Rodas Rivera and The University
 * of Edinburgh
 */

body {
    padding-top: 50px;
    position: relative;
}
```

viewer.css

```
/*
 * This program is free software: you can redistribute it and/or
 modify
 * it under the terms of the GNU General Public License as published
 by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see
 <http://www.gnu.org/licenses/>.
 */

/*
 * Styling for viewer.php
 * Author      : Juan Luis Rodas Rivera
 * Copyright   : (c) 2014, Juan Luis Rodas Rivera and The University
 of Edinburgh
 */

body {
    padding-top: 25px;
}

#navigator {
    border-collapse: collapse;
    width: 100%;
}

#navigator tr.bars {
    height: 41px;
}

#navigator td.minus2 {
    background-color: #d7191c;
    color: ghostwhite;
}

#navigator td.minus1 {
    background-color: #fdae61;
}

#navigator td.selected {
    background-color: #ffffbf;
    font-weight: bold;
    border-left: 1px solid black;
    border-right: 1px solid black;
    font-size: small;
}

#navigator td.overlaps {
    background-color: steelblue;
    color: ghostwhite;
    font-weight: bold;
}
```

```
}

#navigator td.plus1 {
  background-color: #a6d96a;
}

#navigator td.plus2 {
  background-color: #1a9641;
  color: ghostwhite;
}

#navigator td.year {
  border-left: 1px solid gainsboro;
  border-right: 1px solid gainsboro;
}

#navigator td.year.first {
  border-left: none;
}

#navigator td.year.last {
  border-right: none;
}

#navigator td {
  background-color: ghostwhite;
  font-size: smaller;
  vertical-align: bottom;
  text-align: center;
}

img.bar, img.overlap{
  display:block;
  margin-left: auto;
  margin-right: auto;
}

.map {
  height: 560px;
  width: 100%;
}

.ol-popup {
  display: none;
  position: absolute;
  background-color: white;
  -moz-box-shadow: 0 1px 4px rgba(0,0,0,0.2);
  -webkit-filter: drop-shadow(0 1px 4px rgba(0,0,0,0.2));
  filter: drop-shadow(0 1px 4px rgba(0,0,0,0.2));
  padding: 15px;
  border-radius: 10px;
  border: 1px solid #cccccc;
  bottom: 12px;
  left: -50px;
}

.ol-popup:after, .ol-popup:before {
  top: 100%;
  border: solid transparent;
  content: " ";
  height: 0;

```

```
    width: 0;
    position: absolute;
    pointer-events: none;
}

.ol-popup:after {
    border-top-color: white;
    border-width: 10px;
    left: 48px;
    margin-left: -10px;
}

.ol-popup:before {
    border-top-color: #cccccc;
    border-width: 11px;
    left: 48px;
    margin-left: -11px;
}
```

Appendix F – PHP code for the files in the action folder

ww_clusters.php

```
<?php

/*
 * This program is free software: you can redistribute it and/or
 modify
 * it under the terms of the GNU General Public License as published
 by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see
 <http://www.gnu.org/licenses/>.
 */

require_once '../include/study_area.php';

/**
 * Script in charge of calculating the clusters for a study area
 * @copyright (c) 2014, Juan Luis Rodas Rivera and the University of
 Edinburgh
 * @author Juan Luis Rodas Rivera
 */

// Calculate clusters for the study area specified by the user
$study_area = new study_area(htmlspecialchars($_POST["study_area"]));
$num = $study_area->calculate();

// Return the number of clusters calculated in JSON format
header('Content-Type: application/json');
echo json_encode(array("caller" => "#cl_" . $study_area->getId(),
"study_area_id" => $study_area->getId(), "num" => $num),
JSON_PRETTY_PRINT);
die();
```

ww_delete.php

```
<?php

/*
 * This program is free software: you can redistribute it and/or
modify
 * it under the terms of the GNU General Public License as published
by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see
<http://www.gnu.org/licenses/>.
 */

require_once '../include/vars.php';

/**
 * Script in charge of deleting a tag or a study area
 * @copyright (c) 2014, Juan Luis Rodas Rivera and the University of
Edinburgh
 * @author Juan Luis Rodas Rivera
 */

// Retrieve identifier sent by the user
$id = htmlspecialchars($_POST["id"]);
$conn = pg_connect(CONN_STRING);
switch (htmlspecialchars($_POST["type"])) {
    // Delete study area, rows dependent on this key from other tables
will cascade
    case "Study area":
        pg_delete($conn, "study_area", ["id" => $id]);
        break;
    // Delete tag, rows dependent on this key from other tables will
cascade
    case "Tag":
        pg_delete($conn, "tag", ["id" => $id]);
        // Also delete photos related to the tag
        $sql = "DELETE FROM photo WHERE id IN (SELECT id FROM photo
EXCEPT SELECT a.id FROM photo a INNER JOIN photo_tag b ON a.id =
b.photo_id)";
        pg_query($conn, $sql);
        break;
}
pg_close($conn);
die();
```


ww_overlapping_clusters.php

```

<?php

/*
 * This program is free software: you can redistribute it and/or
modify
 * it under the terms of the GNU General Public License as published
by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see
<http://www.gnu.org/licenses/>.
 */

require_once '../include/vars.php';

/**
 * Script in charge of getting the clusters which overlap either with
another cluster or with another time units
 * @copyright (c) 2014, Juan Luis Rodas Rivera and the University of
Edinburgh
 * @author Juan Luis Rodas Rivera
 */

// Retrieve study area identifier
$sa = htmlspecialchars($_POST['sa']);
$return = array();
$conn = pg_connect(CONN_STRING);
// Retrieve cluster which overlap with another cluster
if (array_key_exists('id', $_POST)) {
    // Retrieve cluster identifier
    $cluster = htmlspecialchars($_POST['id']);
    // Retrieve area of biggest cluster in this study area
    $result1 = pg_query($conn, "SELECT COALESCE(MAX(area), 0) FROM
cluster_summary WHERE study_area_id = {$sa};");
    $max_area = pg_fetch_result($result1, 0, 0);
    if ($max_area > 0) {
        // Retrieve clusters which overlap and their relative size
        $sql = "SELECT DISTINCT b.seq, 10 * ST_Area(b.geom) /
{$max_area} FROM cluster a, cluster b "
        . "WHERE ST_Intersects(a.geom, b.geom) AND a.study_area_id
= {$sa} AND a.id = {$cluster} "
        . "AND b.study_area_id = {$sa};";
        $result = pg_query($conn, $sql);
        // Place them in the results array
        while($row = pg_fetch_row($result)) {
            array_push($return, array($row[0], $row[1]));
        }
    }
}
// Retrieve cluster which overlap across time units
} elseif (array_key_exists('seq', $_POST)) {
    // Retrieve sequential
    $seq = htmlspecialchars($_POST['seq']);

```

```
// Loop from two time units before to time units past the
sequential
for ($i = -2; $i <= 2 ; $i++) {
    // Skip current time unit
    if ($i != 0) {
        // Retrieve overlapping clusters
        $sql = "SELECT * FROM clusters_overlap({$sa}, {$seq},
{$i});";
        $result = pg_query($conn, $sql);
        // Place them in the results array
        if (pg_fetch_result($result, 0, 0) == 1) {
            array_push($return, $i);
        }
    }
}
pg_close($conn);

// Return the clusters that overlap in JSON format
header('Content-Type: application/json');
echo json_encode($return, JSON_PRETTY_PRINT);
die();
```

ww_photo_count.php

```
<?php

/*
 * This program is free software: you can redistribute it and/or
modify
 * it under the terms of the GNU General Public License as published
by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see
<http://www.gnu.org/licenses/>.
 */

require_once '../include/vars.php';
require_once '../include/study_area.php';

/**
 * Script in charge of getting the number of photos in a study area
and time unit
 * @copyright (c) 2014, Juan Luis Rodas Rivera and the University of
Edinburgh
 * @author Juan Luis Rodas Rivera
 */

// Retrieve parameters study area and sequential
$sa = new study_area(htmlspecialchars($_POST['sa']));
$seq = htmlspecialchars($_POST['seq']);

// Return the number of photos in JSON format
header('Content-Type: application/json');
echo json_encode(["count" => $sa->getNumPhotos($seq)],
JSON_PRETTY_PRINT);
die();
```

ww_poll_tag_update.php

```
<?php

/*
 * This program is free software: you can redistribute it and/or
modify
 * it under the terms of the GNU General Public License as published
by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see
<http://www.gnu.org/licenses/>.
 */

require_once '../include/vars.php';

/**
 * Script in charge of sending updates on the progress of a tag update
process
 * @copyright (c) 2014, Juan Luis Rodas Rivera and the University of
Edinburgh
 * @author Juan Luis Rodas Rivera
 */

// Retrieve identifier of the tag
$id = htmlspecialchars($_POST['tag']);

// Retrieve update status from the database
$conn = pg_connect(CONN_STRING);
$result = pg_query($conn, "SELECT a.total, COUNT(b.photo_id) FROM
data_grab a LEFT OUTER JOIN grabbed_photo b ON a.id = b.data_grab_id "
. "WHERE a.tag_id = {$id} AND a.latest = 1 GROUP BY a.total");
$pct = 0;

// If update in progress
if (pg_num_rows($result) > 0) {
    // Calculate progress
    $total = pg_fetch_result($result, 0, 0);
    if ($total > 0) {
        $actual = pg_fetch_result($result, 0, 1);
        $pct = round(100 * $actual / $total, 2);
    }
}
pg_close($conn);

// Return the percentage of progress in JSON format
header('Content-Type: application/json');
echo json_encode(["pct" => $pct], JSON_PRETTY_PRINT);
die();
```

ww_quadrat_counts.php

```

<?php

/*
 * This program is free software: you can redistribute it and/or
modify
 * it under the terms of the GNU General Public License as published
by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see
<http://www.gnu.org/licenses/>.
 */

require_once '../include/vars.php';

/**
 * Script in charge of returning the results of quadrat analysis for a
study area and sequential
 * @copyright (c) 2014, Juan Luis Rodas Rivera and the University of
Edinburgh
 * @author Juan Luis Rodas Rivera
 */

// Retrieve parameters study area and sequential
$ssa = htmlspecialchars($_POST['sa']);
$seq = htmlspecialchars($_POST['seq']);

// Check if there are photos on this study area and time unit
$conn = pg_connect(CONN_STRING);
$result1 = pg_query($conn, "SELECT COUNT(*), SUM(count) FROM
quadrat_count({$ssa}, {$seq})");
$total = pg_fetch_result($result1, 0, 0);
$n = pg_fetch_result($result1, 0, 1);
// If there are points print analysis output in HTML format
if ($n > 0) {
    // Lookup table for chi-square test
    // Source: http://www.medcalc.org/manual/chi-square-table.php
    $lookup = [1 => [0.0000393, 0.000982, 1.642, 2.706, 3.841, 5.024,
5.412, 6.635, 7.879, 9.550, 10.828],
                2 => [0.0100, 0.0506, 3.219, 4.605, 5.991, 7.378,
7.824, 9.210, 10.597, 12.429, 13.816],
                3 => [0.0717, 0.216, 4.642, 6.251, 7.815, 9.348,
9.837, 11.345, 12.838, 14.796, 16.266],
                4 => [0.207, 0.484, 5.989, 7.779, 9.488, 11.143,
11.668, 13.277, 14.860, 16.924, 18.467],
                5 => [0.412, 0.831, 7.289, 9.236, 11.070, 12.833,
13.388, 15.086, 16.750, 18.907, 20.515],
                .
                .
                .
                .
                . << Whole array ommited for simplicity, view source file on disk >>

```

```

1000 => [888.564, 914.257, 1037.431, 1057.724,
1074.679, 1089.531, 1093.977, 1106.969, 1118.948, 1133.579,
1143.917]];
?>
<div class="panel panel-heading">
  <h4 class="panel-title">Frequency distribution of quadrat
counts</h4>
</div>
<table class='table table-hover'>
  <thead>
    <tr><th></th><th class='text-right'>No. photos,
<var>k</var></th><th class='text-right'>No. quadrats,
<var>x</var></th><th class='text-right'>Proportion</th><th
class='text-right'><var>k</var>-<var>&micro;</var></th><th
class='text-right'>(<var>k</var>-
<var>&micro;</var>)<sup>2</sup></th><th class='text-
right'><var>x</var>(<var>k</var>-
<var>&micro;</var>)<sup>2</sup></th></tr>
  </thead>
  <tbody>
    <?php
      // Mean quadrat count
      $mu = $n/$total;
      $sum = 0;
      // Get values for the quadrat count distribution
      $result2 = pg_query($conn, "SELECT count as photos, COUNT(*) AS
quadrat FROM quadrat_count({$sa}, {$seq}) GROUP BY photos ORDER BY
photos;");
      while($row = pg_fetch_row($result2)) {
        // Number of events minus mean
        $k_minus_mu = $row[0] - $mu;
        // Number of events minus mean squared
        $k_minus_mu2 = pow($k_minus_mu, 2);
        // Number of events minus mean squared times number of
quadrats
        $x_times_k_minus_mu2 = $row[1] * $k_minus_mu2;
        $sum += $x_times_k_minus_mu2;
        // Round values for displaying
        $td1 = round($row[0], 4); $td2 = round($row[1], 4); $td3 =
round($row[1] / $total, 4); $td4 = round($k_minus_mu, 4); $td5 =
round($k_minus_mu2, 4); $td6 = round($x_times_k_minus_mu2, 4);
        echo "<tr><td></td><td class='text-right'>{$td1}</td><td
class='text-right'>{$td2}</td><td class='text-right'>{$td3}</td><td
class='text-right'>{$td4}</td><td class='text-right'>{$td5}</td><td
class='text-right'>{$td6}</td></tr>";
      }
      pg_close($conn);
      // Round values for displaying
      $td7 = round($sum, 4);
      echo "<tr><td class='text-right'><strong>TOTAL</strong></td><td
class='text-right'><strong>{$n}</strong></td><td class='text-
right'><strong>{$total}</strong></td><td colspan='3'></td><td
class='text-right'><strong>{$td7}</strong></td></tr>";
      // Observed variance
      $s2 = $sum / $total;
      // Variance/mean ration
      $VMR = $s2 / $mu;
      // Chi-square
      $chi2 = $sum / $mu;
    </tbody>
  </tbody>

```

```

</table>
<span class="label label-primary">Statistics</span>
<table class='table'>
  <tbody>
    <tr>
      <?php
        // Round values for displaying
        $td8 = round($mu, 4); $td9 = round($s2, 4); $td10 =
round($VMR, 4); $td11 = round($chi2, 4);
        echo "<td class='text-center'><var>&micro;</var> =
{$td8}</td>";
        echo "<td class='text-center'><var>s</var><sup>2</sup> =
{$td9}</td>";
        echo "<td class='text-center'><var>VMR</var> =
{$td10}</td>";
        echo "<td class='text-center'><var>x</var><sup>2</sup> =
{$td11}</td>";
      <?>
    </tr>
  </tbody>
</table>
<?php
  // If there are values on the lookup table for this number of
degrees of freedom then display them
  if (array_key_exists($total - 1, $lookup)) {
<?>
<span class="label label-primary">Critical values</span>
<table class='table'>
  <thead>
    <tr><th class='text-right'>0.995</th><th class='text-
right'>0.975</th><th class='text-right'>0.20</th><th class='text-
right'>0.10</th><th class='text-right'>0.05</th><th class='text-
right'>0.025</th><th class='text-right'>0.02</th><th class='text-
right'>0.01</th><th class='text-right'>0.005</th><th class='text-
right'>0.002</th><th class='text-right'>0.001</th></tr>
  </thead>
  <tbody>
    <tr>
      <?php
        for($i = 0; $i < 11; $i++) {
          echo "<td class='text-right'>{$lookup[$total -
1][\$i]}</td>";
        }
      <?>
    </tr>
  </tbody>
</table>
<?php
  }
  // If there are no photos on this study area and time unit display a
message accordingly
} else {
<?>
<div class='panel-body'>
  <h4><span class="label label-warning">There are no photos on this
time unit.</span></h4>
</div>
<?php
  }
  die();
}

```

ww_reduce_data.php

```
<?php

/*
 * This program is free software: you can redistribute it and/or
modify
 * it under the terms of the GNU General Public License as published
by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see
<http://www.gnu.org/licenses/>.
 */

require_once '../include/vars.php';

/**
 * Script in charge of performing data reduction for a set of photos
 * @copyright (c) 2014, Juan Luis Rodas Rivera and the University of
Edinburgh
 * @author Juan Luis Rodas Rivera
 */

// Retrieve the identifier of the tag for which photos will be reduced
$tag_id = htmlspecialchars($_POST["tag"]);

// Call the photo reduction function in the database
$conn = pg_connect(CONN_STRING);
$result = pg_query($conn, "SELECT reduce_photos({$tag_id});");
// Retrieve the reduced number of photos
$num = pg_fetch_result($result, 0, 0);

// Return the reduced number of photos in JSON format
header('Content-Type: application/json');
echo json_encode(array("caller" => "#rd_" . $tag_id, "num" => $num),
JSON_PRETTY_PRINT);
die();
```


ww_study_area.php

```
<?php

/*
 * This program is free software: you can redistribute it and/or
modify
 * it under the terms of the GNU General Public License as published
by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see
<http://www.gnu.org/licenses/>.
 */

require_once '../include/study_area.php';

/**
 * Script in charge of performing data insertion and update for study
areas
 * @copyright (c) 2014, Juan Luis Rodas Rivera and the University of
Edinburgh
 * @author Juan Luis Rodas Rivera
 */

// If the id of the study are was not sent on POST then create a new
study area
if (empty(htmlspecialchars($_POST["id"]))) {
    require_once '../include/tag.php';
    // Get the tag
    $tag = new tag(htmlspecialchars($_POST["tag"]));
    // Create the study area object with the parameters sent by the
user
    $sa = new study_area(0, $tag,
htmlspecialchars($_POST["description"]),
htmlspecialchars($_POST["quadrat_size"]),
htmlspecialchars($_POST["min_accuracy"]),
htmlspecialchars($_POST["eps"]), htmlspecialchars($_POST["min_pts"]),
htmlspecialchars($_POST["time_unit"]),
htmlspecialchars($_POST["time_units"]),
htmlspecialchars($_POST["timeframe_from"]),
htmlspecialchars($_POST["geom"]));
    // Store the new study area in the database
    $sa->store();
    // Redirect back to the main page
    header("Location: index.php");
    die();
} else {
    $sa = new study_area(htmlspecialchars($_POST["id"]));
    // Set the new parameters from as specifies by the user
    $sa->setDescription(htmlspecialchars($_POST["description"]));
    $sa->setEps(htmlspecialchars($_POST["eps"]));
    $sa->setFrom(htmlspecialchars($_POST["from"]));
}
```

```
// Check if geometry was modified
if (htmlspecialchars($_POST["upd_geom"]) > 0) {
    $sa->setGeom(htmlspecialchars($_POST["geom"]));
}
$sa->setMinAccuracy(htmlspecialchars($_POST["min_accuracy"]));
$sa->setMinPts(htmlspecialchars($_POST["min_pts"]));
$sa->setQuadratSize(htmlspecialchars($_POST["quadrat_size"]));
$sa->setTimeUnit(htmlspecialchars($_POST["time_unit"]));
$sa->setTimeUnits(htmlspecialchars($_POST["time_units"]));
// Update the study the area
$sa->update();
```

ww_tag.php

```
<?php

/*
 * This program is free software: you can redistribute it and/or
modify
 * it under the terms of the GNU General Public License as published
by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see
<http://www.gnu.org/licenses/>.
 */

require_once '../include/tag.php';

/**
 * Script in charge of performing data insertion and update for tags
 * @copyright (c) 2014, Juan Luis Rodas Rivera and the University of
Edinburgh
 * @author Juan Luis Rodas Rivera
 */

// If tag name sent then insert new tag
if (array_key_exists("tag", $_POST)) {
    // Retrieve the name of the tag to insert and insert it
    $tag = new tag(0, htmlspecialchars($_POST["tag"]));
    // Redirect to main page
    header("Location: index.php");
} // If tag identifier sent then perform update of photos for this tag
elseif (array_key_exists("id", $_POST)) {
    require_once '../include/data_grab.php';
    // Retrieve the identifier of the tag to update
    $tag = new tag(htmlspecialchars($_POST["id"]));
    // Create a new data grab object
    $dg = new data_grab($tag);
    // Retrieve new photos from Flickr
    $dg->grab();
    // Return data on the photos downloaded in JSON format
    header('Content-Type: application/json');
    echo json_encode(array('caller' => "#pu_".$dg->getTag()->getId(),
'tag_id' => $tag->getId(), 'count' => $dg->getNPhotos(), 'updated' =>
$dg->getEnded()), JSON_PRETTY_PRINT);
}
die();
```

Appendix G – PHP code for the files in the include folder

cluster.php

```
<?php

/*
 * This program is free software: you can redistribute it and/or
 modify
 * it under the terms of the GNU General Public License as published
 by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see
 <http://www.gnu.org/licenses/>.
 */

require_once dirname(__FILE__) . '/vars.php';

/**
 * Class that represents a cluster of photos calculated by the DBSCAN
 algorithm
 * @copyright (c) 2014, Juan Luis Rodas Rivera and the University of
 Edinburgh
 * @author Juan Luis Rodas Rivera
 */
class cluster {

    /**
     * The unique identifier of the cluster
     * @var integer
     */
    private $id;

    /**
     * The list of photos within the main extent of the cluster
     * @var photo[]
     */
    private $points;

    /**
     * The list of photos that are density-reachable from the cluster
     * @var photo[]
     */
    private $density_reachable;

    /**
     * Returns the number of photos in the cluster
     * @return integer
     */
    function getNumPoints() {
        return count($this->points) + count($this->density_reachable);
    }
}

```

```

    }

    /**
     * Returns the list of photos in the cluster
     * @return photo[]
     */
    function getPoints() {
        return array_merge($this->points, $this->density_reachable);
    }

    /**
     * Class constructor
     */
    function __construct() {
        $this->points = array();
        $this->density_reachable = array();
    }

    /**
     * Adds a photo to the cluster
     * @param photo $point The point to add
     * @param boolean $density_reachable Whether the point is density
    reachable or not
     */
    function add($point, $density_reachable = FALSE) {
        if ($density_reachable) {
            array_push($this->density_reachable, $point);
        } else {
            array_push($this->points, $point);
        }
    }

    /**
     * Stores the cluster on the database
     * @param integer $id Id of the study area to which the cluster
    belongs
     * @param integer $seq Sequential number of the cluster in the
    timeframe
     * @param real $seps Distance used by the buffer function to create
    the cluster
     */
    function store($id, $seq, $seps) {
        // Retrieve all the id's of the photos which are part of the
    cluster
        $point_ids = "";
        foreach($this->points as $point) {
            $point_ids = $point_ids . $point->getId() . ',';
        }
        $point_ids = rtrim($point_ids, ',');
        $conn = pg_connect(CONN_STRING);
        // Use union together with buffer to create the extent of the
    cluster
        $sql = "INSERT INTO cluster (study_area_id, seq, geom) SELECT
    {$id}, {$seq}, "
        . "ST_Union(ST_Buffer(geom, {$seps})) FROM photo WHERE id IN
    ({$point_ids});";
        pg_query($conn, $sql);
        // Retrieve the id of the cluster
        $this->id = pg_fetch_result(pg_query($conn, "SELECT
    currval('cluster_id_seq)'), 0, 0);
        // Mark each individual photo as part of the cluster
    
```

```
        foreach($this->getPoints() as $point) {
            pg_insert($conn, 'cluster_photo', array('cluster_id' =>
$this->id, 'photo_id' => $point->getId()));
        }
        pg_close($conn);
    }
```

data_grab.php

```

<?php

/*
 * This program is free software: you can redistribute it and/or
modify
 * it under the terms of the GNU General Public License as published
by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see
<http://www.gnu.org/licenses/>.
 */

require_once 'Phlickr/Api.php';
require_once dirname(__FILE__) . '/photo.php';
require_once dirname(__FILE__) . '/tag.php';
require_once dirname(__FILE__) . '/vars.php';

/**
 * Class that represents a process of data downloading from Flickr
 * @copyright (c) 2014, Juan Luis Rodas Rivera and the University of
Edinburgh
 * @author Juan Luis Rodas Rivera
 */
class data_grab {

    /**
     * The unique identifier of the data grab
     * @var integer
     */
    private $id;

    /**
     * The tag for which the data is being downloaded
     * @var tag
     */
    private $tag;

    /**
     * The total number of photos as reported by Flickr
     * @var integer
     */
    private $total;

    /**
     * The number of the page which is being downloaded
     * @var integer
     */
    private $page;

    /**
     * The total number of pages for this data grab

```

```
* @var integer
*/
private $pages;

/**
 * The time at which the download process started
 * @var DateTime
 */
private $started;

/**
 * The time at which the download process ended
 * @var DateTime
 */
private $ended;

/**
 * The total number of photos actually downloaded
 * @var integer
 */
private $n_photos;

/**
 * The point of access to Flickr's REST endpoint
 * @var Phlickr_Api
 */
private $api;

/**
 * The parameters sent to the flickr.photos.search method
 * @var mixed[]
 */
private $parms;

/**
 * Returns the unique identifier of the data grab
 * @return integer
 */
public function getId() {
    return $this->id;
}

/**
 * Returns the tag for which the data is being downloaded
 * @return tag
 */
public function getTag() {
    return $this->tag;
}

/**
 * Returns the total number of photos as reported by Flickr
 * @return integer
 */
public function getTotal() {
    return $this->total;
}

/**
 * Returns the time at which the download process started
 * @return string
```



```

    */
    public function getStarted() {
        return $this->started->format('Y-m-d H:i:s');
    }

    /**
     * Returns the time at which the download process ended
     * @return string
     */
    public function getEnded() {
        return $this->ended->format('Y-m-d H:i:s');
    }

    /**
     * Returns the total number of photos actually downloaded
     * @return integer
     */
    public function getNPhotos() {
        return $this->n_photos;
    }

    /**
     * Class constructor
     * @param tag $tag Tag for which the data is being downloaded
     */
    function __construct($tag) {
        // Define Flickr API parameters
        $this->api = new Phlickr_Api(Flickr_API_KEY,
Flickr_API_SECRET);
        // Assign initial values
        $this->tag = $tag;
        $this->page = 1;
        $this->started = new DateTime();
        // These are the default parameters used when searching for
photos on Flickr
        $this->parms = array('tags' => $this->tag->getName(),
'min_upload_date' => $this->getMinUploadDate(),
        'max_upload_date' => $this->getStarted(), 'sort' => 'date-
posted-asc', 'has_geo' => 1,
        'extras' => 'date_upload,date_taken,geo', 'per_page' =>
250, 'page' => $this->page,
        'min_taken_date' => '2000-01-01 00:00:00');
        // Get data on the search from Flickr
        $rsp = $this->api->ExecuteMethod('flickr.photos.search',
$this->parms);
        // Store information retrieved from Flickr
        $this->total = intval($rsp->xml->photos['total']);
        $this->pages = intval($rsp->xml->photos['pages']);
        $conn = pg_connect(CONN_STRING);
        // Mark all previous data grabs for this tag as not being the
latest
        pg_update($conn, "data_grab", array('latest' => 0),
array('tag_id' => $this->tag->getId(), 'latest' => 1));
        // Store the new data grab on the database
        $sql = "INSERT INTO data_grab (tag_id, total, started) VALUES
({$this->tag->getId()}, {$this->total}, '{$this->getStarted()}')";
        pg_query($conn, $sql);
        // Retrieve the unique identifier
        $this->id = pg_fetch_result(pg_query($conn, "SELECT
currval('data_grab_id_seq')"), 0, 0);
        pg_close($conn);
    }

```

```

    }

    /**
     * Returns the date to be used as minimum upload date for the data
download
     * @return string
     */
    private function getMinUploadDate() {
        // Define a default date to return
        $return = "2000-01-01 00:00:00";
        // Retrieve the last time this tag was updated
        $conn = pg_connect(CONN_STRING);
        $sql = "SELECT MAX(started) FROM data_grab WHERE tag_id = " .
$this->tag->getId();
        $result = pg_query($conn, $sql);
        // If the tag has been updated before return that date
        if (!pg_field_is_null($result, 0, 0)) {
            $return = pg_fetch_result($result, 0, 0);
        }
        pg_close();
        return $return;
    }

    /**
     * Recalculates parameters when total of pictures exceeds 4000
     * @param string $lt Latest timestamp
     */
    private function recalculateParams($lt) {
        // Set new parameters if more than 4000 photos (16 pages x 250
photos)
        if ($this->page > 16) {
            $this->page = 1;
            $this->parms['min_upload_date'] = $lt;
            $rsp = $this->api->ExecuteMethod('flickr.photos.search',
$this->parms);
            $this->pages = intval($rsp->xml->photos['pages']);
        }
        $this->parms['page'] = $this->page;
    }

    /**
     * Retrieves photos from Flickr
     */
    function grab() {
        // If there are no photos to download then skip calculations
and return
        if ($this->total == 0) {
            $this->done();
            return;
        }
        $lastTimestamp = 0;
        // Loop though all the pages with photos
        do {
            // Calculate parameters
            $this->recalculateParams($lastTimestamp);
            // Get photos from Flickr
            $rsp = $this->api->ExecuteMethod('flickr.photos.search',
$this->parms);
            // Loop through each photo on this page
            foreach ($rsp->xml->photos->photo as $photo) {
                // Get data on the photo

```

```

        $pht = new photo($photo['id'], $photo['dateupload'],
$photo['datetaken'], $photo['accuracy'], $photo['owner'],
        $photo['latitude'], $photo['longitude'],
$photo['farm'], $photo['server'], $photo['secret']);
        // Store the photo on the database
        $pht->store($this->tag, $this);
        $lastTimestamp = intval($photo['dateupload']);
    }
    $this->page++;
} while ($this->page <= $this->pages);
// Store information when finished
$this->done();
}

/**
 * Stores information on the database relating the data download
once it has finished
 */
private function done() {
    // Get time of ending
    $this->ended = new DateTime();
    // In cases where there are no photos to download the start
and end time can be the same
    if ($this->ended == $this->started) {
        // Add a second to end time to make it different from
start time
        $this->ended->add(DateInterval::createFromDateString("1
second"));
    }
    $conn = pg_connect(CONN_STRING);
    // Store time of ending
    pg_query($conn, "UPDATE data_grab SET ended = '{$this-
>getEnded()}' WHERE id = {$this->id}");
    // Retrieve number of photos downloaded
    $this->n_photos = pg_fetch_result(pg_query($conn, "SELECT
COUNT(*) FROM photo_tag WHERE tag_id = {$this->tag->getId()}"), 0, 0);
    pg_close($conn);
}

```

dbscan_algorithm.php

```
<?php

/*
 * This program is free software: you can redistribute it and/or
modify
 * it under the terms of the GNU General Public License as published
by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see
<http://www.gnu.org/licenses/>.
 */

require_once dirname(__FILE__) . '/vars.php';
require_once dirname(__FILE__) . '/photo.php';
require_once dirname(__FILE__) . '/cluster.php';

/**
 * Implementation of the DBSCAN clustering algorithm
 * @copyright (c) 2014, Juan Luis Rodas Rivera and the University of
Edinburgh
 * @author Juan Luis Rodas Rivera
 */
class dbscan_algorithm {

    /**
     * Tag for which the clusters are being calculated
     * @var tag
     */
    private $tag;

    /**
     * Minimum accuracy of the points on the calculation
     * @var integer
     */
    private $min_accuracy;

    /**
     * WKT geometry of the study area
     * @var string
     */
    private $geom;

    /**
     * Search distance
     * @var real
     */
    private $eps;

    /**
     * Lower boundary of the timeframe
     * @var string
     */

```

```
    */
    private $from;

    /**
     * Upper boundary of the timeframe
     * @var string
     */
    private $to;

    /**
     * Minimum number of points to define a cluster
     * @var integer
     */
    private $min_pts;

    /**
     * List of photos that fall within the search criteria
     * @var photo[]
     */
    private $points;

    /**
     * List of clusters found
     * @var cluster[]
     */
    private $clusters;

    /**
     * List with that contains data on which photos have been visited
     * @var integer[]
     */
    private $visited;

    /**
     * Returns the search distance
     * @return real
     */
    public function getEps() {
        return $this->eps;
    }

    /**
     * Returns the minimum number of points to define a cluster
     * @return integer
     */
    public function getMinPoints() {
        return $this->min_pts;
    }

    /**
     * Returns the list of clusters found
     * @return cluster[]
     */
    public function getClusters() {
        return $this->clusters;
    }

    /**
     * Returns the number of clusters found
     * @return integer
     */
    */
```

```

public function getNumClusters() {
    return count($this->clusters);
}

/**
 * Class constructor
 * @param tag $tag Tag for which the clusters are being calculated
 * @param integer $min_accuracy Minimum accuracy of the points on
the calculation
 * @param string $geom WKT geometry of the study area
 * @param string $from Lower boundary of the timeframe
 * @param string $to Upper boundary of the timeframe
 * @param real $eps Search distance
 * @param integer $min_pts Minimum number of points to define a
cluster
 */
function __construct($tag, $min_accuracy, $geom, $from, $to, $eps,
$min_pts) {
    // Assign initial values
    $this->tag = $tag;
    $this->min_accuracy = $min_accuracy;
    $this->geom = $geom;
    $this->from = $from;
    $this->to = $to;
    $this->eps = $eps;
    $this->min_pts = $min_pts;
    $this->points = array();
    $this->clusters = array();
    $this->visited = array();
    // Retrieve the list of photos that meet the search criteria
    $sql = "SELECT * FROM photo_reduced WHERE tag_id = {$tag-
>getId()} AND accuracy >= {$this->min_accuracy} AND "
    . "date_taken BETWEEN '{$this->from}' AND '{$this->to}' AND
ST_Within(geom, {$this->geom});";
    $conn = pg_connect(CONN_STRING);
    $result = pg_query($conn, $sql);
    // Add them to the photo list
    while ($row = pg_fetch_row($result)) {
        array_push($this->points, new photo($row[1], $row[2],
$row[3], $row[4], $row[5], $row[6], $row[7], $row[8], $row[9],
$row[10], $row[0]));
    }
    pg_close($conn);
}
/**
 * Search for clusters in the list of photos
 */
function find_clusters() {
    // If there are less photos than the minimum then return
    if (count($this->points) < $this->min_pts) {
        return;
    }
    // Loop through the list of photos
    for ($i = 0; $i < count($this->points); $i++) {
        // Retrieve current photo
        $point = $this->points[$i];
        // If this photo has been visited already then skip to
next photo
        if (array_key_exists($point->getId(), $this->visited)) {
            continue;
        }
    }
}

```

```

        // Get the neighbour of this photo
        $neighbours = $point->getNeighbors($this->eps, $this->tag,
$this->min_accuracy, $this->geom, $this->from, $this->to);
        // If the number of neighbours is greater or equal to the
minimum number of photos then create a new cluster and expand it
        if (count($neighbours) >= $this->min_pts) {
            array_push($this->clusters, $this->expand_cluster(new
cluster(), $point, $neighbours));
        } else {
            // Else, mark photo as visited and noise
            $this->visited[$point->getId()] = "NOISE";
        }
    }
}

/**
 * Expands a cluster starting from the specified photo and its
neighbours
 * @param cluster $cluster Cluster to expand
 * @param photo $point First photo in the cluster
 * @param photo[] $neighbours Neighbours of the first photo
 * @return cluster
 */
private function expand_cluster(cluster $cluster, photo $point,
array $neighbours) {
    // Add photo to cluster
    $cluster->add($point);
    // Mark it as visited and part of a cluster
    $this->visited[$point->getId()] = "IN_CLUSTER";
    $i = 0;
    // Loop through the list of neighbours
    while ($i < count($neighbours)) {
        // By default mark photo as density-reachable
        $density_reachable = TRUE;
        // Get current neighbour
        $current = $neighbours[$i];
        // Check if photo has been visited
        $status = array_key_exists($current->getId(), $this-
>visited);
        // If it has not been visited
        if (!$status) {
            // Retrieve the neighbours of the current photo
            $current_neighbours = $current->getNeighbors($this-
>eps, $this->tag, $this->min_accuracy, $this->geom, $this->from,
$this->to);
            // If the number of neighbours is greater or equal to
the minimum number of photos
            if (count($current_neighbours) >= $this->min_pts) {
                // Add its neighbours to the list of neighbours
                $neighbours = $this->merge($neighbours,
$current_neighbours);
                // Mark it as core point
                $density_reachable = FALSE;
            }
        } else {
            // Else, retrieve its visited status
            $status = $this->visited[$current->getId()];
        }
        // If the point has not been visited and is not in a
cluster add it to this cluster and mark it as in a cluster
        if ($status != "IN_CLUSTER") {

```

```

        $this->visited[$current->getId()] = "IN_CLUSTER";
        $cluster->add($current, $density_reachable);
    }
    $i++;
}
return $cluster;
}

/**
 * Merges to list of photos together
 * @param photo[] $seeds List of photos to which to add
 * @param photo[] $neighbors List of photos to add
 * @return photo[]
 */
private function merge($seeds, $neighbors) {
    // Loop through the list of photos to add
    foreach($neighbors as $neighbor) {
        // If a photo is not already in the list, add it
        if (!in_array($neighbor, $seeds)) {
            array_push($seeds, $neighbor);
        }
    }
    return $seeds;
}

/**
 * Stores the clusters in the database
 * @param integer $id The identifier of the study area
 * @param integer $seq The sequential to which the cluster belongs
 */
function store($id, $seq) {
    // Loop through each cluster and store it
    foreach($this->clusters as $cluster) {
        $cluster->store($id, $seq, $this->eps);
    }
}

```


footer.php

```
<?php

/*
 * This program is free software: you can redistribute it and/or
modify
 * it under the terms of the GNU General Public License as published
by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see
<http://www.gnu.org/licenses/>.
 */

/**
 * Footer used on the application web pages
 * @copyright (c) 2014, Juan Luis Rodas Rivera and the University of
Edinburgh
 * @author Juan Luis Rodas Rivera
 */
?>
<footer>
  <div class="container">
    <div class="well well-sm text-center">
      
      <h6>School of Geosciences</h6>
      <h6>Dissertation for the degree of</h6>
      <h6><strong>MSc in Geographical Information
Science</strong></h6>
      <h6><strong>Juan Luis Rodas Rivera</strong></h6>
      <h6>August 2014</h6>
    </div>
  </div>
</footer>
```

photo.php

```
<?php

/*
 * This program is free software: you can redistribute it and/or
modify
 * it under the terms of the GNU General Public License as published
by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see
<http://www.gnu.org/licenses/>.
 */

require_once dirname(__FILE__) . '/vars.php';

/**
 * Class that represents a photo from Flickr
 * @copyright (c) 2014, Juan Luis Rodas Rivera and the University of
Edinburgh
 * @author Juan Luis Rodas Rivera
 */
class photo {

    /**
     * The unique identifier of the photo
     * @var integer
     */
    private $id;

    /**
     * The unique identifier of the photo used by Flickr
     * @var string
     */
    private $num;

    /**
     * Date the photo was added
     * @var string
     */
    private $dateAdded;

    /**
     * Date the photo was taken
     * @var string
     */
    private $dateTaken;

    /**
     * Declared accuracy of the photo
     * @var integer
     */
    private $accuracy;
```

```
/**
 * Identifier of the user that uploaded the photo
 * @var string
 */
private $user;

/**
 * Latitude of the location the photo was taken
 * @var real
 */
private $lat;

/**
 * Longitude of the location the photo was taken
 * @var real
 */
private $lon;

/**
 * Server farm where the photo is hosted
 * @var integer
 */
private $farm;

/**
 * Server where the photo is hosted
 * @var integer
 */
private $server;

/**
 * Secret of the photo
 * @var string
 */
private $secret;

/**
 * Returns the unique identifier of the photo
 * @return integer
 */
public function getId() {
    return $this->id;
}

/**
 * Returns the unique identifier of the photo used by Flickr
 * @return string
 */
public function getNum() {
    return $this->num;
}

/**
 * Returns the date the photo was added
 * @return string
 */
public function getDateAdded() {
    return $this->dateAdded;
}
```

```

/**
 * Returns the date the photo was taken
 * @return string
 */
public function getDateTaken() {
    return $this->dateTaken;
}

/**
 * Returns the declared accuracy of the photo
 * @return integer
 */
public function getAccuracy() {
    return $this->accuracy;
}

/**
 * Returns the identifier of the user that uploaded the photo
 * @return string
 */
public function getUser() {
    return $this->user;
}

/**
 * Returns the server farm where the photo is hosted
 * @return integer
 */
public function getFarm() {
    return $this->farm;
}

/**
 * Returns the server where the photo is hosted
 * @return integer
 */
public function getServer() {
    return $this->server;
}

/**
 * Returns the secret of the photo
 * @return string
 */
public function getSecret() {
    return $this->secret;
}

/**
 * Returns the geometry of the photo as expected to be inserted in
the database
 * @return string
 */
public function getGeom() {
    return "ST_Transform(ST_GeomFromText('{ $this-
>getGeomAsWKT()}', 4326), " . EPSG . ")";
}

/**
 * Returns the WKT representation of the location of the photo
 * @return string

```

```

    */
    public function getGeomAsWKT() {
        return "POINT({$this->lon} {$this->lat})";
    }

    /**
     * Class constructor
     * @param string $num The unique identifier of the photo used by
Flickr
     * @param string|integer $dAdded Date the photo was added
     * @param string $dTaken Date the photo was taken
     * @param integer $accuracy Declared accuracy of the photo
     * @param string $user Identifier of the user that uploaded the
photo
     * @param real $lat Latitude of the location the photo was taken
     * @param real $lon Longitude of the location the photo was taken
     * @param integer $farm Server farm where the photo is hosted
     * @param integer $server Server where the photo is hosted
     * @param string $secret Secret of the photo
     * @param integer $id The unique identifier of the photo
    */
    function __construct($num, $dAdded, $dTaken, $accuracy, $user,
$lat, $lon, $farm, $server, $secret, $id = 0) {
        // If the identifier was specified then assign it
        if ($id > 0) {
            $this->id = $id;
        }
        // Assign default values
        $this->num = strval($num);
        // Date added can be specified either as a Unix timestamp or a
string
        $ct1 = DateTime::createFromFormat('U', $dAdded);
        if (!$ct1) {
            $this->dateAdded = $dAdded;
        } else {
            $this->dateAdded = $ct1->format('Y-m-d H:i:s');
        }
        $ct2 = new DateTime($dTaken);
        $this->dateTaken = $ct2->format('Y-m-d H:i:s');
        $this->accuracy = (intval($accuracy) > 0) ? intval($accuracy)
: 1;
        $this->user = (string)$user;
        // Mercator cannot project coordinates at the poles, so reduce
tolerance of point
        switch (floatval($lat)) {
            case 90.0:
                $this->lat = 89.999999;
                break;
            case -90.0:
                $this->lat = -89.999999;
                break;
            default:
                $this->lat = floatval($lat);
        }
        $this->lon = floatval($lon);
        $this->farm = intval($farm);
        $this->server = intval($server);
        $this->secret = (string)$secret;
    }

    /**

```

```

* Store the photo on the database
* @param tag $tag Tag to assign the photo to
* @param data_grab $dg Data grab used to retrieve the photo
*/
function store($tag, $dg) {
    $conn = pg_connect(CONN_STRING);
    // Check if photo already exists
    $sel = pg_select($conn, "photo", array('num' => $this->num));
    if ($sel == 0) {
        // If not then store it on the database
        $sql = "INSERT INTO photo (num, date_added, date_taken,
accuracy, usr, lat, lon, farm, server, secret, geom) "
            . "VALUES ({ $this->num }, '{ $this->dateAdded }',
'{$this->dateTaken}', { $this->accuracy }, '{ $this->user }', "
            . "{ $this->lat }, { $this->lon }, { $this->farm },
{ $this->server }, '{ $this->secret }', { $this->getGeom() });";
        pg_query($conn, $sql);
        // Retrieve the unique identifier
        $this->id = pg_fetch_result(pg_query($conn, "SELECT
currval('photo_id_seq')"), 0, 0);
        // Associate the photo with the tag and with the data grab
        pg_insert($conn, "photo_tag", array('photo_id' => $this-
>id, 'tag_id' => $tag->getId()));
        pg_insert($conn, "grabbed_photo", array('photo_id' =>
$this->id, 'data_grab_id' => $dg->getId()));
    } else {
        // Else, retrieve its unique identifier
        $this->id = $sel[0]["id"];
        // Associate the photo with the tag if not already in
database
        $sell = pg_select($conn, "photo_tag", array('photo_id' =>
$this->id, 'tag_id' => $tag->getId()));
        if ($sell == 0) {
            pg_insert($conn, "photo_tag", array('photo_id' =>
$this->id, 'tag_id' => $tag->getId()));
        }
        // Associate the photo with the data grab if not already
in database
        $sel2 = pg_select($conn, "grabbed_photo", array('photo_id'
=> $this->id, 'data_grab_id' => $dg->getId()));
        if ($sel2 == 0) {
            pg_insert($conn, "grabbed_photo", array('photo_id' =>
$this->id, 'data_grab_id' => $dg->getId()));
        }
    }
    pg_close($conn);
}

/**
* Retrieves a list of photos that are within a specified distance
and search criteria from this photo
* @param real $seps Search radius
* @param tag $tag Tag for which the neighbours are being searched
* @param integer $min_accuracy Minimum accuracy of the neighbours
* @param string $geom Geometry to search within
* @param string $from Lower boundary of the timeframe
* @param string $sto Upper boundary of the timeframe
* @return photo[]
*/
function getNeighbors($seps, $tag, $min_accuracy, $geom, $from,
$sto) {

```

```
// Retrieve all neighbours that meet the search criteria
$return = array();
$sql = "SELECT a.* FROM photo_reduced a, photo b "
      . "WHERE ST_DWithin(a.geom, b.geom, {$seps}) AND b.id = "
      . "{$this->id} AND a.tag_id = {$tag->getId()} AND "
      . "a.accuracy >= {$min_accuracy} AND a.date_taken "
      . "BETWEEN '{$from}' AND '{$to}' AND "
      . "ST_Within(a.geom, {$geom});";
$conn = pg_connect(CONN_STRING);
$result = pg_query($conn, $sql);
// Add them to the returning array
while ($row = pg_fetch_row($result)) {
    array_push($return, new photo($row[1], $row[2], $row[3],
    $row[4], $row[5], $row[6], $row[7], $row[8], $row[9], $row[10],
    $row[0]));
}
pg_close($conn);
return $return;
```

study_area.php

```

<?php

/*
 * This program is free software: you can redistribute it and/or
modify
 * it under the terms of the GNU General Public License as published
by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see
<http://www.gnu.org/licenses/>.
 */

require_once dirname(__FILE__) . '/vars.php';
require_once dirname(__FILE__) . '/tag.php';
require_once dirname(__FILE__) . '/dbscan_algorithm.php';
require_once dirname(__FILE__) . '/geoPHP/geoPHP.inc';

/**
 * Class that represents a study area defined by the user
 * @copyright (c) 2014, Juan Luis Rodas Rivera and the University of
Edinburgh
 * @author Juan Luis Rodas Rivera
 */
class study_area {

    /**
     * The unique identifier of the study area
     * @var integer
     */
    private $id;

    /**
     * The tag asociated to the study area
     * @var tag
     */
    private $tag;

    /**
     * The textual description of the study area
     * @var string
     */
    private $description;

    /**
     * Whether the description needs update
     * @var boolean
     */
    private $upd_description = false;

    /**
     * The quadrat size used in the study area

```



```

    * @var real
    */
    private $quadrat_size;

    /**
     * Whether the quadrat size needs update
     * @var boolean
     */
    private $upd_quadrat_size = false;

    /**
     * The minimal accuracy for photos in the study area
     * @var integer
     */
    private $min_accuracy;

    /**
     * Whether the minimal accuracy needs update
     * @var boolean
     */
    private $upd_min_accuracy = false;

    /**
     * The search radius for neighbours on the DBSCAN algorithm
     * @var real
     */
    private $eps;

    /**
     * Whether the eps parameter needs update
     * @var boolean
     */
    private $upd_eps = false;

    /**
     * The minimum number of points to create a cluster on the DBSCAN
algorithm
     * @var integer
     */
    private $min_pts;

    /**
     * Whether the min_pts parameter needs update
     * @var boolean
     */
    private $upd_min_pts = false;

    /**
     * The time unit used on the study area
     * @var string
     */
    private $time_unit;

    /**
     * Whether the time unit needs update
     * @var boolean
     */
    private $upd_time_unit = false;

    /**
     * The number of time units in the study area

```

```
* @var integer
*/
private $time_units;

/**
 * Whether the number of time units needs update
 * @var boolean
 */
private $upd_time_units = false;

/**
 * The starting date of the timeframe
 * @var string
 */
private $from;

/**
 * Whether the starting date of the timefrme needs update
 * @var boolean
 */
private $upd_from = false;

/**
 * The ending date of the timefram
 * @var string
 */
private $to;

/**
 * Whether clusters in the study area have been calculated
 * @var integer
 */
private $calculated;

/**
 * The date of the last update made to the study area
 * @var string
 */
private $updated;

/**
 * The geometry of the study area
 * @var string
 */
private $geom;

/**
 * Whether the geometry needs update
 * @var boolean
 */
private $upd_geom = false;

/**
 * Returns the unique identifier of the study area
 * @return integer
 */
public function getId() {
    return $this->id;
}

/**
```

```

    * Returns the tag asociated to the study area
    * @return tag
    */
public function getTag() {
    return $this->tag;
}

/**
 * Returns the textual description of the study area
 * @return string
 */
public function getDescription() {
    return $this->description;
}

/**
 * Sets the textual description of the study area
 * @param string $description The new textual description
 */
public function setDescription($description) {
    if ($this->description != $description) {
        $this->description = $description;
        $this->upd_description = true;
    }
}

/**
 * Returns the quadrat size used in the study area
 * @return real
 */
public function getQuadratSize() {
    return $this->quadrat_size;
}

/**
 * Sets the quadrat size used in the study area
 * @param real $quadrat_size The new quadrat size
 */
public function setQuadratSize($quadrat_size) {
    if ($this->quadrat_size != $quadrat_size) {
        $this->quadrat_size = $quadrat_size;
        $this->upd_quadrat_size = true;
    }
}

/**
 * Returns the minimal accuracy for photos in the study area
 * @return integer
 */
public function getMinAccuracy() {
    return $this->min_accuracy;
}

/**
 * Sets the minimal accuracy for photos in the study area
 * @param integer $min_accuracy The new minimal accuracy
 */
public function setMinAccuracy($min_accuracy) {
    if ($this->min_accuracy != $min_accuracy) {
        $this->min_accuracy = $min_accuracy;
        $this->upd_min_accuracy = true;
    }
}

```

```

    }
}

/**
 * Returns the search radius for neighbours on the DBSCAN
algorithm
 * @return real
 */
public function getEps() {
    return $this->eps;
}

/**
 * Sets the search radius for neighbours on the DBSCAN algorithm
 * @param real $eps The new eps
 */
public function setEps($eps) {
    if ($this->eps != $eps) {
        $this->eps = $eps;
        $this->upd_eps = true;
    }
}

/**
 * Returns the minimum number of points to create a cluster on the
DBSCAN algorithm
 * @return integer
 */
public function getMinPts() {
    return $this->min_pts;
}

/**
 * Sets the minimum number of points to create a cluster on the
DBSCAN algorithm
 * @param integer $min_pts The new min_pts
 */
public function setMinPts($min_pts) {
    if ($this->min_pts != $min_pts) {
        $this->min_pts = $min_pts;
        $this->upd_min_pts = true;
    }
}

/**
 * Returns the time unit used on the study area as an interval
 * @return DateInterval
 */
private function getInterval() {
    return DateInterval::createFromDateString($this->time_unit);
}

/**
 * Returns the time unit used on the study area
 * @return string
 */
public function getTimeUnit() {
    return $this->time_unit;
}

/**

```

```

    * Sets the time unit used on the study area
    * @param string $time_unit The new time unit
    */
    public function setTimeUnit($time_unit) {
        if ($this->time_unit != $time_unit) {
            $this->time_unit = $time_unit;
            $this->setTo();
            $this->upd_time_unit = true;
        }
    }

    /**
     * Returns the number of time units in the study area
     * @return integer
     */
    public function getTimeUnits() {
        return $this->time_units;
    }

    /**
     * Sets the number of time units in the study area
     * @param integer $time_units The new number of time units
     */
    public function setTimeUnits($time_units) {
        if ($this->time_units != $time_units) {
            $this->time_units = $time_units;
            $this->setTo();
            $this->upd_time_units = true;
        }
    }

    /**
     * Returns the starting date of the timeframe
     * @param integer $index If specified returns the starting date of
this time unit
     * @return DateTime
     */
    public function getFrom($index = 0) {
        $from = new DateTime($this->from);
        for($i = 0; $i < $index; $i++) {
            $from->add($this->getInterval());
        }
        return $from;
    }

    /**
     * Sets the starting date of the timeframe
     * @param string $from The new starting date of the timeframe
     */
    public function setFrom($from) {
        if ($this->from != $from) {
            $this->from = $from;
            $this->setTo();
            $this->upd_from = true;
        }
    }

    /**
     * Returns the ending date of the timeframe
     * @param integer $index If specified returns the ending date of
this time unit

```

```

    * @return DateTime
    */
    public function getTo($index = NULL) {
        if (is_null($index) xor $index == $this->time_units) {
            return new DateTime($this->to);
        } else {
            return $this->getFrom($index + 1)-
>sub(DateInterval::createFromDateString("1 second"));
        }
    }

    /**
     * Calculates and sets the ending date of the timeframe
     */
    private function setTo() {
        $to = clone $this->getFrom();
        for ($i = 0; $i < $this->time_units; $i++) {
            $to->add($this->getInterval());
        }
        $to->sub(DateInterval::createFromDateString("1 second"));
        $this->to = $to->format('Y-m-d H:i:s');
    }

    /**
     * Returns whether clusters in the study area have been calculated
     * @return integer
     */
    public function getCalculated() {
        return $this->calculated;
    }

    /**
     * Returns the date of the last update made to the study area
     * @return string
     */
    public function getUpdated() {
        return $this->updated;
    }

    /**
     * Returns the geometry of the study area
     * @return string
     */
    public function getGeom() {
        return "ST_GeomFromText('{ $this->geom}', " . EPSG . ")";
    }

    /**
     * Sets the geometry of the study area
     * @param string $geom The new geometry
     */
    public function setGeom($geom) {
        $this->geom = $geom;
        $this->upd_geom = true;
    }

    /**
     * Returns the geometry of the study area as WKT
     * @return string
     */
    public function getGeomAsWKT() {

```

```

        return $this->geom;
    }

    /**
     * Returns the geometry of the quadrats covering the study area
     * @return string
     */
    public function getQuadratGeom() {
        $conn = pg_connect(CONN_STRING);
        // Retrieve beehive grid and merge all polygons
        $result = pg_query($conn, "SELECT
ST_AsText(ST_Union(ST_SnapToGrid(geom, 0.0001))) AS geom FROM
study_area_as_hex_quadrat({$this->id});");
        $quadGeom = "ST_GeomFromText('" . pg_fetch_result($result, 0,
0) . "', " . EPSG . ")";
        pg_close($conn);
        return $quadGeom;
    }

    /**
     * Returns the number of photos present on the study area
     * @param integer $seq If specified returns the number of photos
in this time unit
     * @return integer
     */
    public function getNumPhotos($seq = -1) {
        $conn = pg_connect(CONN_STRING);
        if ($seq < 0) {
            $result = pg_query($conn, "SELECT COUNT(*) FROM
photos_on_study_area WHERE study_area_id = {$this->id};");
        } else {
            $from = $this->getFrom($seq)->format("Y-m-d H:i:s");
            $to = $this->getTo($seq)->format("Y-m-d H:i:s");
            $result = pg_query($conn, "SELECT COUNT(*) FROM
photos_on_study_area WHERE study_area_id = {$this->id} AND date_taken
BETWEEN '{$from}' AND '{$to}';");
        }
        pg_close($conn);
        if ($result != 0) {
            return pg_fetch_result($result, 0, 0);
        } else {
            return 0;
        }
    }

    /**
     * Returns whether the study area needs updating
     * @return boolean
     */
    private function needsUpdate() {
        return $this->upd_description or $this->upd_quadrat_size or
$this->needsClusterDeletion();
    }

    /**
     * Returns whether the study area needs to delete cluster
calculated
     * @return boolean
     */
    private function needsClusterDeletion() {

```

```

        return $this->upd_eps or $this->upd_from or $this->upd_geom or
        $this->upd_min_accuracy or $this->upd_min_pts or $this->upd_time_unit
        or $this->upd_time_units;
    }

    /**
     * Class constructor
     * @param integer $id The unique identifier of the study area
     * @param tag $tag The tag asociated to the study area
     * @param string $description The textual description of the study
area
     * @param real $quadrat_size The quadrat size used in the study
area
     * @param integer $min_accuracy The minimal accuracy for photos in
the study area
     * @param real $eps The search radius for neighbours on the DBSCAN
algorithm
     * @param integer $min_pts The minumum number of points to create
a cluster on the DBSCAN algorithm
     * @param string $time_unit The time unit used on the study area
     * @param integer $time_units The number of time units in the
study area
     * @param string $from The staring date of the timeframe
     * @param string $geom The geometry of the study area
     */
    function __construct($id, $tag = null, $description = null,
        $quadrat_size = null, $min_accuracy = null, $eps = null, $min_pts =
        null, $time_unit = null, $time_units = null, $from = null, $geom =
        null) {
        // If the identifier was specified then assign it
        if ($id > 0) {
            $this->id = $id;
            $conn = pg_connect(CONN_STRING);
            // Retrieve values for this study area
            $result = pg_select($conn, "study_area", array("id" =>
        $this->id));
            pg_close($conn);
            if (count($result) == 1) {
                $this->tag = new tag($result[0]["tag_id"]);
                $this->description = $result[0]["description"];
                $this->quadrat_size = $result[0]["quadrat_size"];
                $this->min_accuracy = $result[0]["min_accuracy"];
                $this->eps = $result[0]["eps"];
                $this->min_pts = $result[0]["min_pts"];
                $this->time_unit = str_replace("mon", "month",
        $result[0]["time_unit"]);
                $this->time_units = $result[0]["time_units"];
                $this->from = $result[0]["timeframe_from"];
                $this->to = $result[0]["timeframe_to"];
                $this->calculated = $result[0]["calculated"];
                $this->updated = $result[0]["updated"];
                // Use geoPHP to read WKB geometry
                $wkb = new WKB();
                $geometry = $wkb->read($result[0]["geom"], TRUE);
                $this->geom = $geometry->asText();
            }
        } else {
            // Assign values from the user
            $this->tag = $tag;
            $this->description = $description;
            $this->quadrat_size = $quadrat_size;
        }
    }

```



```

        $this->min_accuracy = $min_accuracy;
        $this->eps = $eps;
        $this->min_pts = $min_pts;
        $this->time_unit = $time_unit;
        $this->time_units = $time_units;
        $this->from = $from;
        $this->setTo();
        $this->geom = $geom;
    }
}

/**
 * Store the study area on the database
 */
function store() {
    $conn = pg_connect(CONN_STRING);
    // Insert study area
    $sql = "INSERT INTO study_area (tag_id, description,
quadrat_size, min_accuracy, eps, min_pts, time_unit, "
        . "time_units, timeframe_from, timeframe_to, geom)
VALUES ({ $this->tag->getId() }, '{ $this->description}', "
        . "{ $this->quadrat_size }, { $this->min_accuracy },
{ $this->eps }, { $this->min_pts }, '{ $this->time_unit}', "
        . "{ $this->time_units }, '{ $this->from}', '{ $this-
>to}', { $this->getGeom() });";
    pg_query($conn, $sql);
    // Retrieve unique identifier
    $this->id = pg_fetch_result(pg_query($conn, "SELECT
currval('study_area_id_seq')"), 0, 0);
    pg_close($conn);
}

/**
 * Updates the study area with changes made by the user
 */
function update() {
    // Execute only if update is necessary
    if ($this->needsUpdate()) {
        // Build SQL update sentence
        $sql = "UPDATE study_area SET ";
        if ($this->upd_description) {
            $sql = $sql . "description = '{ $this->description}',
";

        }
        if ($this->upd_eps) {
            $sql = $sql . "eps = { $this->eps }, ";
        }
        if ($this->upd_geom) {
            $sql = $sql . "geom = { $this->getGeom() }, ";
        }
        if ($this->upd_min_accuracy) {
            $sql = $sql . "min_accuracy = { $this->min_accuracy},
";

        }
        if ($this->upd_min_pts) {
            $sql = $sql . "min_pts = { $this->min_pts }, ";
        }
        if ($this->upd_quadrat_size) {
            $sql = $sql . "quadrat_size = { $this->quadrat_size},
";

        }
    }
}

```

```

        if ($this->upd_time_unit) {
            $sql = $sql . "time_unit = '{$this->time_unit}', ";
        }
        if ($this->upd_time_units) {
            $sql = $sql . "time_units = {$this->time_units}, ";
        }
        if ($this->upd_from) {
            $sql = $sql . "timeframe_from = '{$this->from}', ";
        }
        if ($this->upd_from or $this->upd_time_unit or $this->
>upd_time_units) {
            $sql = $sql . "timeframe_to = '{$this->to}', ";
        }
        $conn = pg_connect(CONN_STRING);
        // Check if clusters need to be recalculated
        if ($this->needsClusterDeletion()) {
            pg_delete($conn, "cluster", ["study_area_id" => $this->
>id]);
            $sql = $sql . "updated = now(), calculated = 0 WHERE
id = {$this->id}";
        } else {
            $sql = $sql . "updated = now() WHERE id = {$this->
>id}";
        }
        // Perform the update
        pg_query($conn, $sql);
        pg_close($conn);
    }
}

/**
 * Perform cluster calculation on the study area and return the
number of clusters found
 * @return integer
 */
function calculate() {
    // Retrieve timeframe parameters
    $from = $this->getFrom();
    $interval = $this->getInterval();
    // Retrieve study area hexagonal grid boundary
    $geom = $this->getQuadratGeom();
    $n = 0;
    // Loop through each time unit
    for($i = 0; $i < $this->time_units; $i++) {
        // Define upper boundary of the time unit
        $to = clone $from;
        $to->add($interval);
        $to->sub(DateInterval::createFromDateString("1 second"));
        // Create DBSCAN object
        $dbscan = new dbscan_algorithm($this->tag, $this->
>min_accuracy, $geom, $from->format("Y-m-d H:i:s"), $to->format("Y-m-d
H:i:s"), $this->eps, $this->min_pts);
        // Perform cluster calculation
        $dbscan->find_clusters();
        // Add the number of cluster found
        $n = $n + $dbscan->getNumClusters();
        // Store clusters found in the database
        $dbscan->store($this->id, $i);
        // Define lower boundary of the next time unit
        $from->add($interval);
    }
}

```

```
$conn = pg_connect(CONN_STRING);
// Update study area
pg_update($conn, "study_area", array("calculated" => 1,
'updated' => 'now()'), array("id" => $this->id));
pg_close($conn);
return $n;
}

/**
 * Returns a copy of this study area for a new tag
 * @param tag $new_tag The tag to be used in the new study area
 * @return study_area
 */
public function copy($new_tag) {
    return new study_area(0, $new_tag, '', $this->quadrat_size,
    $this->min_accuracy, $this->eps, $this->min_pts, $this->time_unit,
    $this->time_units, $this->from, $this->geom);
}
```

tag.php

```
<?php

/*
 * This program is free software: you can redistribute it and/or
modify
 * it under the terms of the GNU General Public License as published
by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see
<http://www.gnu.org/licenses/>.
 */

require_once dirname(__FILE__) . '/vars.php';

/**
 * Class that represents a of tag
 * @copyright (c) 2014, Juan Luis Rodas Rivera and the University of
Edinburgh
 * @author Juan Luis Rodas Rivera
 */
class tag {

    /**
     * The unique identifier of the tag
     * @var integer
     */
    private $id;

    /**
     * The name or description of the tag
     * @var string
     */
    private $name;

    /**
     * Returns the unique identifier of the tag
     * @return integer
     */
    public function getId() {
        return $this->id;
    }

    /**
     * Returns the name or description of the tag
     * @return string
     */
    public function getName() {
        return $this->name;
    }

    /**
```

```

* Class constructor
* @param integer The unique identifier of the tag
* @param string The name or description of the tag
*/
function __construct($id, $name = null) {
    $conn = pg_connect(CONN_STRING);
    // If the identifier is specified then retrieve the name from
the database
    if ($id > 0) {
        $this->id = $id;
        $result = pg_select($conn, 'tag', array('id' => $this-
>id));
        $this->name = $result[0]['name'];
    } else {
        // Convert name to lower case
        $this->name = strtolower($name);
        // Retrieve unique identifier from database
        $result = pg_select($conn, 'tag', array('name' => $this-
>name));
        if ($result == 0) {
            // If the tag does not exist in the database insert it
            pg_insert($conn, 'tag', array('name' => $this->name));
            $this->id = pg_fetch_result(pg_query($conn, "SELECT
currval('tag_id_seq)'), 0, 0);
        } else {
            // Assign the unique identifier to the tag
            $this->id = $result[0]['id'];
        }
    }
    pg_close($conn);
}

```

vars.php

```
<?php

/*
 * This program is free software: you can redistribute it and/or
modify
 * it under the terms of the GNU General Public License as published
by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see
<http://www.gnu.org/licenses/>.
 */

/**
 * File which contains configuration parameters for the application
 * @copyright (c) 2014, Juan Luis Rodas Rivera and the University of
Edinburgh
 * @author Juan Luis Rodas Rivera
 */

/**
 * Connection string to connect to the database
 */
define('CONN_STRING', 'host=localhost dbname=flickr user=postgres
password=flickr');

/**
 * Flickr API key
 */
define('FLICKR_API_KEY', '875c7452aaea518daab4907211dc1f89');

/**
 * Flickr API secret
 */
define('FLICKR_API_SECRET', 'e293d50b4c392c40');

/**
 * Coordinate system used for the geometries on the database
 */
define('EPSG', '3857');

/**
 * Address of host where geoserver is running
 */
define('GEOSERVER', 'http://localhost');
```