CROWD-CURATED GEOGRAPHICAL AREAS ON A SCALE-CAPABLE CLOUD ARCHITECTURE

by

JUAN P ALEMÁN-REYES

A project submitted in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING
in
COMPUTER ENGINEERING

UNIVERSITY OF PUERTO RICO
MAYAGÜEZ CAMPUS
2012

Approved by:

_____	_____
Iván Baigés-Valentín, PhD Member, Graduate Committee	Date
_____	_____
Manuel Rodríguez-Martínez, PhD Member, Graduate Committee	Date
_____	_____
Jaime Seguel, PhD President, Graduate Committee	Date
_____	_____
Isabel Ríos, MBA Representative of Graduate Studies	Date
_____	_____
Pedro Rivera-Vega, PhD Chairperson of the Department	Date

ABSTRACT

This project presents a new way of managing geographical areas through a web service called AREAS. Using the geohash technique, this platform allows the crowd to define what is the area of any place in a democratic way. This is possible using a model that meets two requirements, being easy for anyone to understand, and at the same time being efficient enough to exist on a global scale. This report presents the theoretical development of what the geohash is and how it can be used to define areas. It also presents all the technological development, using technologies that have the capability to scale efficiently in a cloud infrastructure. Specifically, a RESTful web framework is used to allow any device to interact with the system. And also a document-oriented NoSQL database is used to manage relations in the data in a more efficient way.

RESUMEN

Este proyecto presenta una nueva manera de manejar áreas geográficas a través de un servicio web llamado AREAS. Utilizando la técnica del "geohash", esta plataforma permite que la multitud pueda definir cuanta área ocupa cualquier lugar de una manera democrática. Esto es posible gracias a un modelo que cumple con dos requisitos, es sencillo para cualquier persona entender, y a la misma vez, es suficientemente eficiente para que exista a una escala mundial. En este reporte se presenta el desarrollo teórico sobre el "geohash" y cómo se puede utilizar para definir áreas. También se presenta el desarrollo tecnológico, donde las tecnologías utilizadas tienen la capacidad de escalar eficientemente en infraestructuras tipo nube. Específicamente, se usa un "RESTful web framework" que permite que cualquier dispositivo pueda comunicarse con el sistema. Y también se utiliza una base de datos NoSQL orientado a documentos, la cual permite manejar relaciones en los datos de manera mas eficiente.

ACKNOWLEDGEMENTS

This project is dedicated to everyone that has helped me grow as a person.

To Dr. Jaime Seguel: I consider you more than an advisor; you are a friend to whom I am proud of having and sharing values with. Thanks for always believing in me, even while I was still maturing about what and how to do my Masters. You always gave me great advice, I won't forget.

To my professors: I want to thank the professors that gave me opportunities to get involved in experiences that helped me develop professionally. But also, I want to thank those who have taught me further from the academics. Getting involved with interdisciplinary matters that help me discover the most important problems I want to dedicate my life in solving.

To my family: For making me feel that I have received the most complete and unconditional support in the world. And letting me live many experiences that have helped me find myself. I will always be grateful, and I will make you proud.

Thank you all.

TABLE OF CONTENTS

FIGURE LIST

Figures **Page**

1 INTRODUCTION

The data from most of the top websites is generated and curated by its own users. From giving a “like” to a video, to even sharing its link on social networks, all these actions contribute in giving more relevancy to the data.

The power people have to give value to anything on the web must be recognized. So based on this, could something similar be done with things or events in the real world? Some of this can already be seen, when people add their location when sharing a status or photo on social networks, but there has to be more possibilities.

Emerging technologies like augmented reality could bring a new generation of applications that allow interaction with our real life 3-dimensional spaces. Therefore, there is an opportunity to do an intermediary between an ambitious 3-dimensional space and the simple points that are actually used to represent places. And this opportunity is to start considering places with the geographical area they cover.

This project presents AREAS, which is a platform that allows geographical areas to be defined by the crowd in an easy and democratic way. At the same time, the platform is built with a universal interface that allows communication between different devices and is deployed with technologies that have the capability to scale efficiently in a cloud infrastructure.

1.1 Problem Statement

There are many location data APIs (Application Programming Interface) available. All these web services are based on what is referred as “points of interest” (POI). Web services like Google Places API [10] or foursquare Venues Platform [7] work in the following way, shown in Figure 1.1.

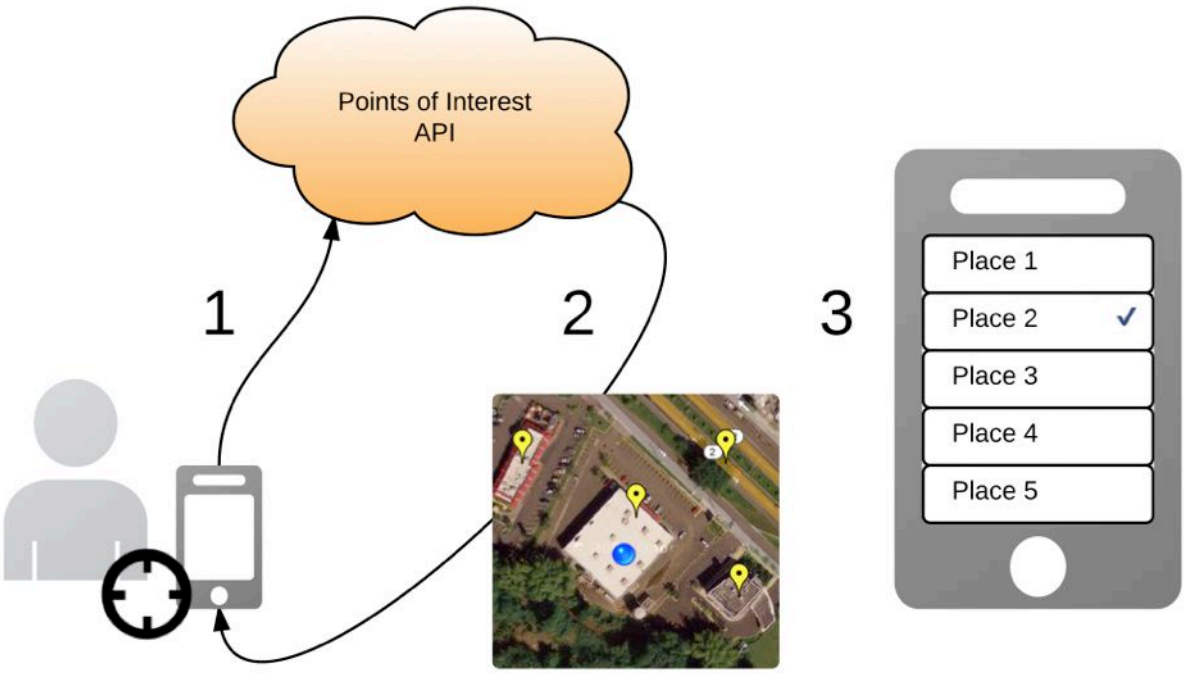

Figure 1.1 Associating data to a location by choosing a near point

If someone uses these APIs through a mobile app, and wants to add his/her location to anything in the app, the first thing that happens is that a query is sent with the detected location from the device. Then, the web service returns a list with all the POIs near the sent coordinates. Finally, the user selects the location he/she is standing at.

Even though this approach works, it could be better. If these locations where represented by the geographical area they cover, as shown in Figure 1.2, there would be no necessity for the user to choose where he/she is standing. If the sent coordinates are inside the area, the place could be automatically chosen. But if there is any kind of conflict, like an inaccurate GPS (Global Positioning System) reading or more than one place for a certain area (e.g. a multiple floor building), then the system could still "fallback" and allow the user to choose the place.

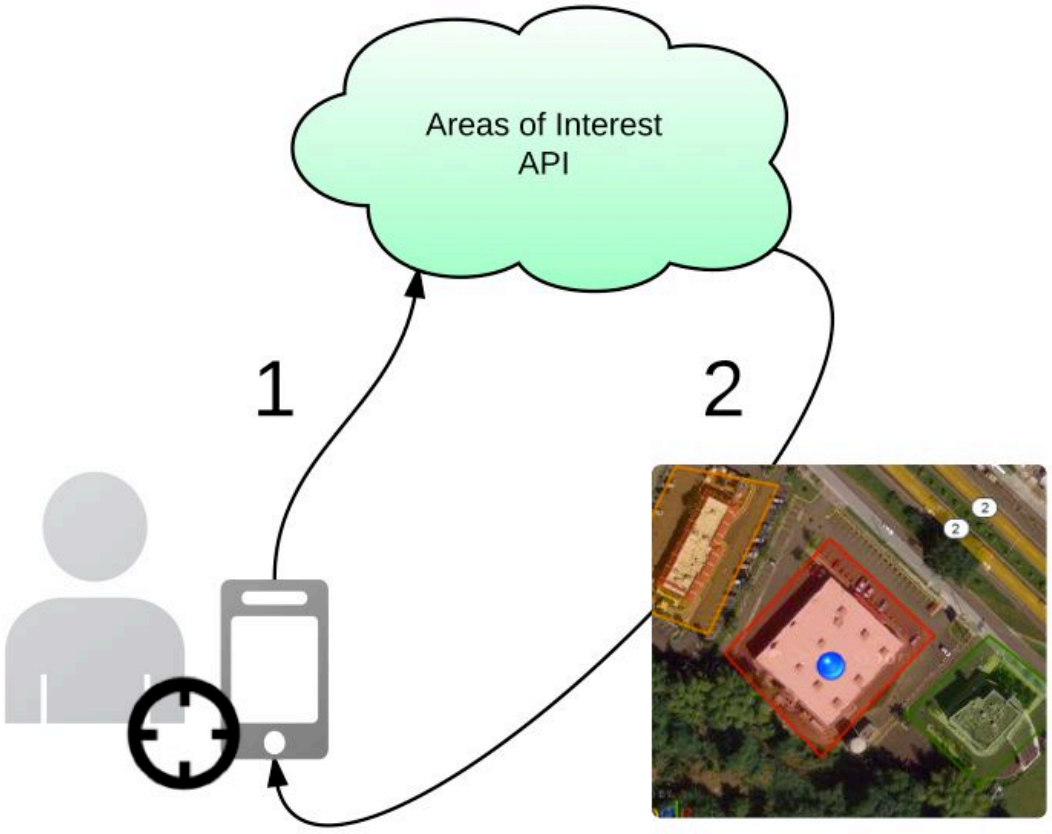

Figure 1.2 Associating data to a location by being inside its area

But using geographical areas can be more valuable than just improving the geo-awareness of location-based data. The area of places is one of the first steps to improve local management of natural resources. The ecological footprint calculation is completely based on area, where

the human demands are compared to what capacity (biocapacity) the planet has to balance them out. This biocapacity is calculated based on the different land types that exist and the area they cover [5].

But these areas can also have social and economic benefits. Like sending alerts to parents from their children when they move out from a determined area. Or receiving notifications from offers at stores nearby. [4]

So there is an opportunity for a crowdsourced area creation platform that is easy for anyone to understand and use. This is the problem that wants to be solved in this project.

The only source available for geographical areas is at big scale, from the boundaries of continents to the boundaries of states or provinces. Smaller areas, like city boundaries, are limited. And neighborhood areas, is even more limited. [17] But even if they were easily available, these data is not manageable at a customer level. They are formatted in complicated formats that require GIS (Geographic Information System) knowledge to manage.

1.2 Proposed Solution: AREAS

AREAS is an online platform to create and manage geographical areas, designed from the ground up to be easy for anyone to understand and contribute. The focus is in hyperlocal areas, that is, as local as possible. For example, from the area covered by a school to the area covered by it's baseball park.

But before going further, there is one thing that has to be specified. AREAS is not about cartography. Figure 1.3 shows OpenStreetMap [16], which allows anyone to become a cartographer for the world.

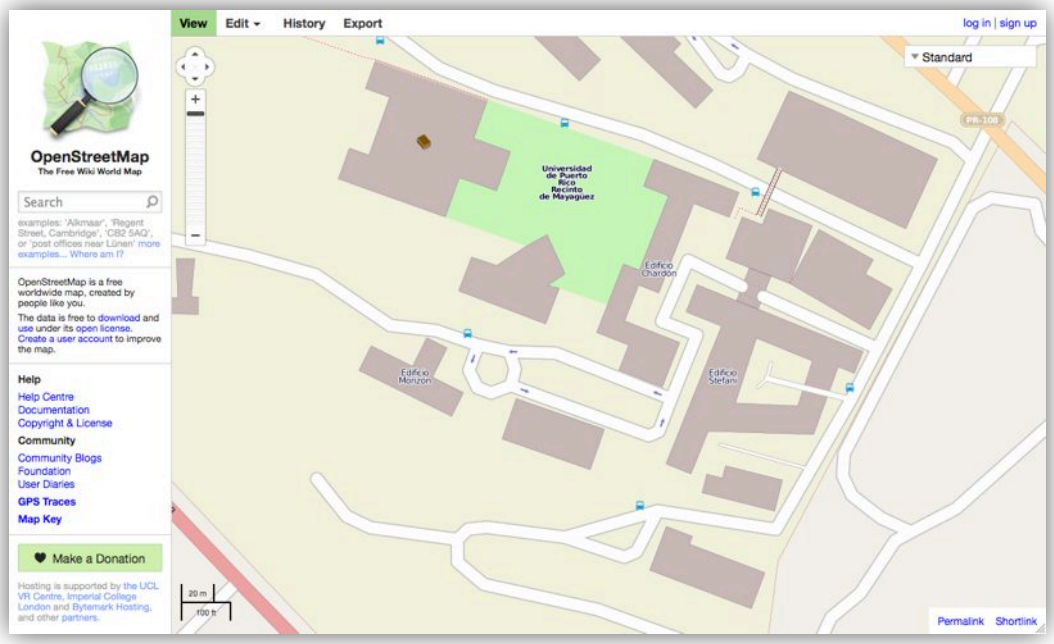

Figure 1.3 Main screen of OpenStreetMap website [16]

The focus of AREAS is about expanding the concept of "point of interest" to what could be called "area of interest". Then, these areas can be used for anything, like improving the geo-awareness of mobile devices, like was shown in Figure 1.2.

For example, imagine a mobile app that allows people to take pictures about the places they are in. If people use the app inside a place that has an area defined, like a park, then these pictures could be automatically associated to the park without any user intervention. Then, when other people access these pictures, they can be certain that the pictures are actually from the park.

So, because the focus is not on map-making, there is no need for the areas to be exact representations of the real world. This allows the usage of a new way to model areas that has the benefit that anyone can easily contribute to the platform. And this model is based on the geohash [14].

The geohash is an alphanumeric code calculated from the latitude and longitude coordinates. This will be explained in detail in section 2.1, but its main characteristic is that geohashes define areas, and these areas can have different sizes based on the resolution of the geohash.

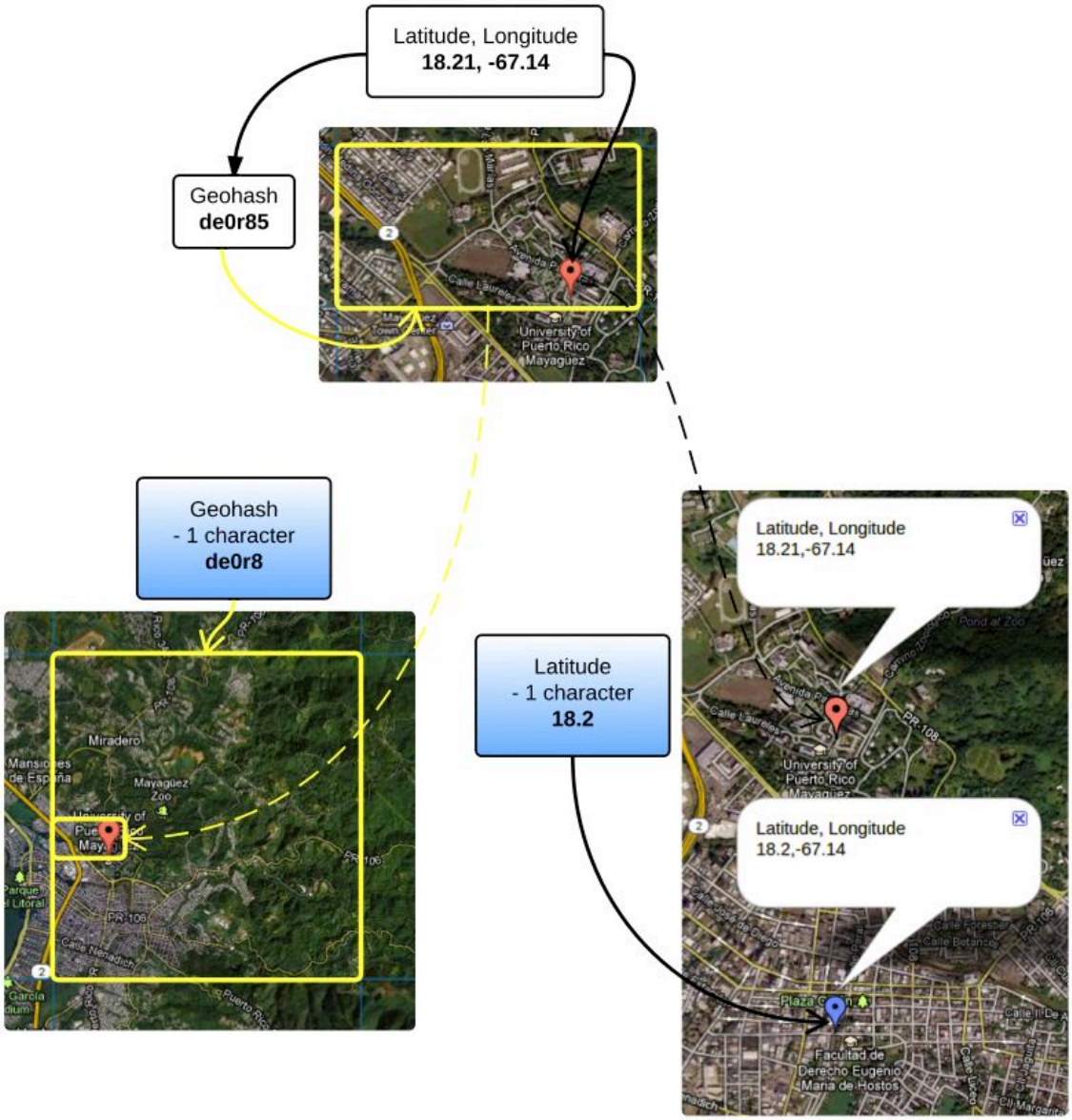

Figure 1.4 Comparing coordinates with the geohash

In Figure 1.4, the top map shows the geohash from a pair of latitude and longitude coordinates. These coordinates are always going to be inside the area defined by its geohash. If the last character of this geohash is removed, it will change its resolution. As seen in the bottom left map, the result is that a bigger area is covered, but it will still contain the original geohash inside of it. On the other hand, lets look what happens if a character is removed from

a coordinate. As shown in the bottom right map, having fewer characters does not change the resolution, it changes the location completely. This helps understand the difference between the geohash and coordinates, one is area based and the other is point based.

For the non-expert crowd, the only unit for location is latitude and longitude coordinates. So, when dealing with geographical areas, what is actually used is also based on coordinates.

For areas with coordinates, a term used is “geo-fencing”, which can be done in two ways. The first way is by simply adding a radius of a certain length to any point, like shown in the left map in Figure 1.5. The other way is more precise, which is by creating a polygon from an array of coordinates, as shown in the right map in Figure 1.5.

Figure 1.5 Geo-fencing alternatives done for a baseball park

But the precision of the polygon comes at a price. AREAS wants to become a crowdsourced area creation platform, and dealing with coordinates is too ambiguous. This can be proven with an example. Imagine if three people are asked to define the area of a place by drawing a polygon over a map, obtaining the results from Figure 1.6.

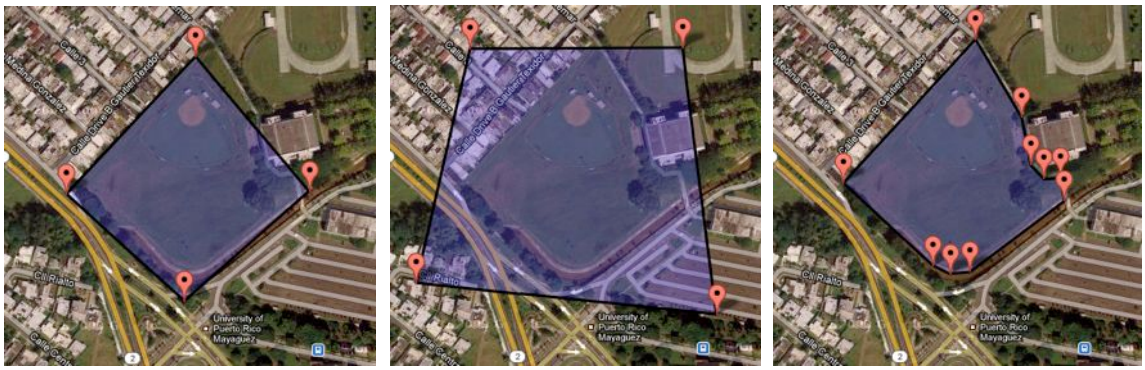

Figure 1.6 Different polygons by different users

Choosing the correct one is not related to the number of people trying to draw the area, is based on who made the "best" polygon, which is ambiguous.

If areas are done with geohashes, there is no ambiguity. For each resolution, there is only one geohash for any area.

The geographic area of a place could be defined by how many geohashes it contains. Then, if done by many people, users will simply decide if an area represented by one geohash should be considered or not as part of the total area of a place. Figure 1.7 shows three users defining the area of a place with geohashes.

Figure 1.7 Different geohash-defined areas by different users

Each yellow rectangle represents a different geohash. These defined areas are ideal to be chosen by the crowd. For example, some sort of "rating" could be assigned to each geohash based on how many people supported or not that specific geohash as being part of the area of the place. This rating is directly proportional to number of people contributing, proving that is a better model to do areas with the crowd. This is shown in Figure 1.8.

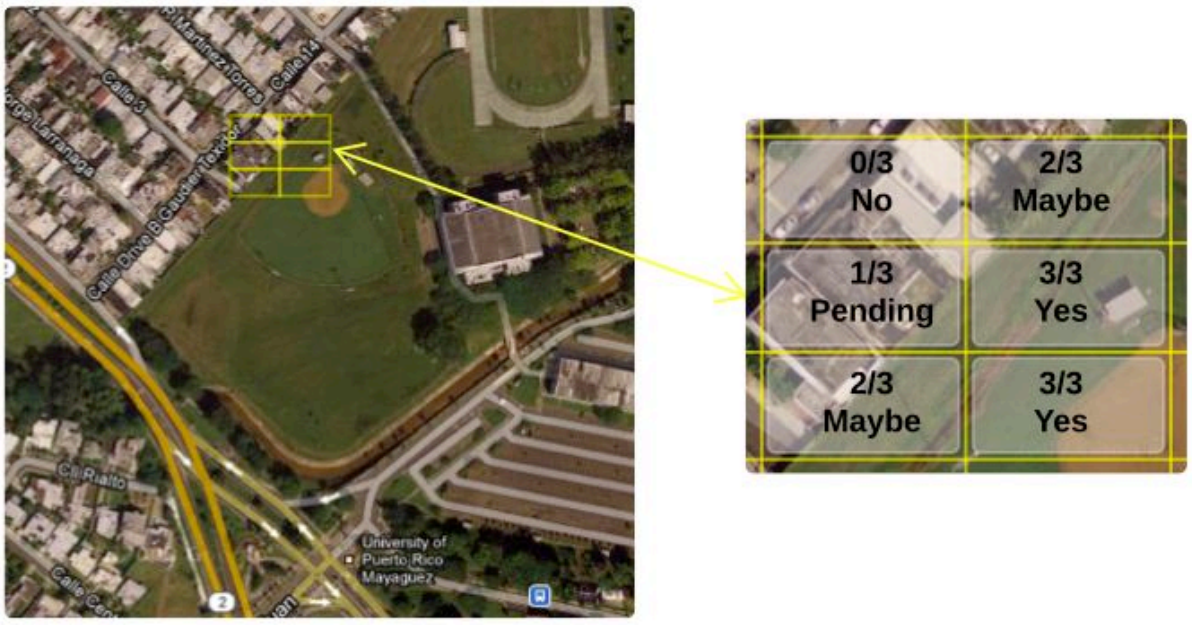

Figure 1.8 Each geohash is curated by the crowd (based on how many users chose it)

And making a crowdsourced platform for hyperlocal data is not only about the ease of user interaction, is also a feasibility problem. AREAS is being designed with technologies that can eventually take advantage of the scaling capabilities of a cloud computing architecture, specifically, allowing horizontal scalability.

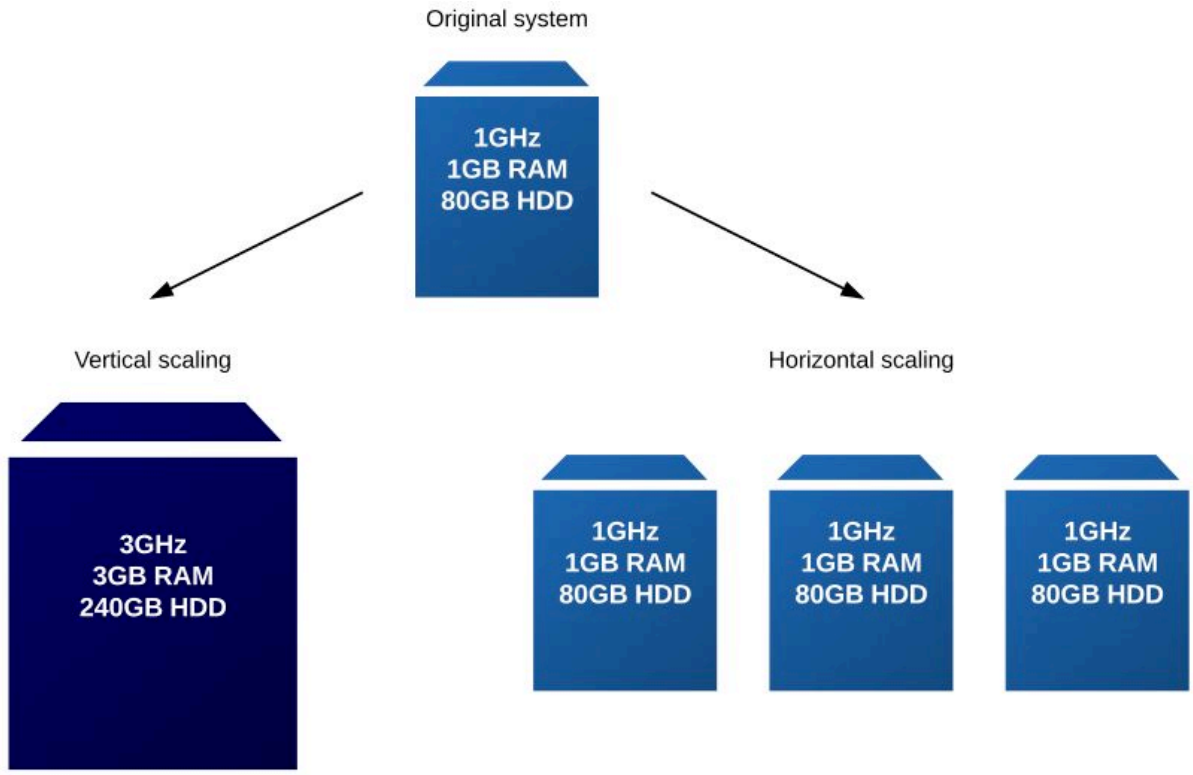

Figure 1.9 Options for scalability

If a web site or web service has lots of users, the original single computer and/or database may not keep up with demand. As shown in Figure 1.9, instead of increasing capabilities of the single machine (vertical scaling), one could add more machines (horizontal scaling) made with commodity hardware specs to keep up with the growth demands. This is more economical but also reduces risk of interrupting service if a machine fails.

In terms of data capacity, allowing horizontal scalability can be simplified by new paradigms in database technology. These new types of databases are commonly referred as NoSQL, because instead of focusing on data normalization, these are designed to handle lots of data efficiently.

For AREAS, the goal is to eventually become an API for areas of interest. This will allow any kind of device to take advantage of its data. Because of this, everything is being developed over a RESTful web framework, meaning that it complies with the REST (REpresentational State Transfer) design principles. This allows universal communication between a variety of devices and is also compatible with the horizontal scalability requirement.

The development of AREAS will work as a proof of technology for this web service to exist atop a scale-capable cloud architecture.

1.3 Contributions

This project has the following contributions:

1. Demonstrates an intuitive new way to manage geographical areas.
2. Develops a model that allows areas to be curated by the crowd.
3. Implements technologies that allow horizontal scalability in the cloud.
4. Develops a RESTful web service that is compatible with multiple devices.
5. Manages data relations in an efficient way using a document-oriented database.

1.4 Report Organization

The previous sections introduced the concept of the geohash and how it can be used to determine the geographical area of places. Section 2 will start by showing more details about the geohash concept, and then present the technologies used to create AREAS. Then, section 3 will go in depth within the application.

Section 3.1 will be a walkthrough of the web service, first showing the RESTful interface and then focusing on the different experiments done until getting to the finalized area defining platform. Section 3.2 will explain its architecture, and section 3.3 will go in detail with the data models used for defining areas in a crowd-curated way. Finally, section 4 will present conclusions and future work.

2 BACKGROUND

AREAS is a project with strong basis on the geohash concept, so a theoretical background of the geohash will be the first focus of this section. Then the technological background will be presented, starting with the front-end stack. Specifically the technologies used to present maps in the web browser. Finally the back-end stack will be presented, which is focused on technologies that allow the creation of a cloud-based web service.

2.1 Theoretical Background: Geohash

2.1.1 What is the geohash?

The geohash is a geocode system designed and put into public domain by Gustavo Niemeyer [14]. It uses the same origin as the latitude and longitude coordinates, but instead of telling locations based on the angles, it tells them based on the subdivisions done to the area of the Earth. The following figures show this concept. [20]

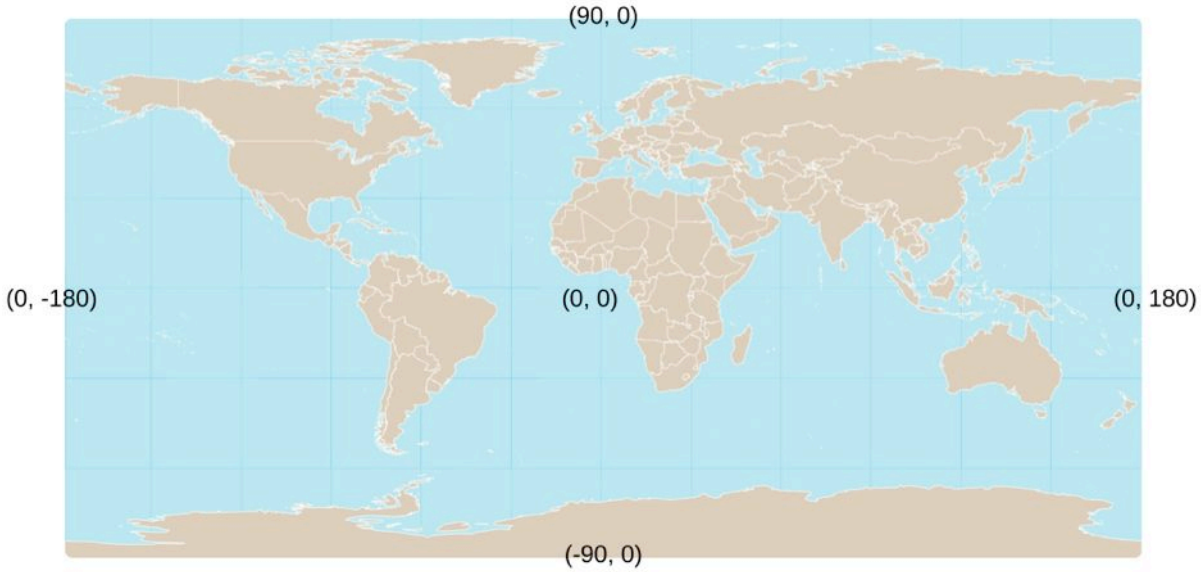

Figure 2.1 Map based on quadrangles created by latitude and longitude coordinates

The map in Figure 2.1 shows a flat projection of the Earth based on quadrangles created by the latitude and longitude coordinates. Let's say that the user is interested in the geohash of a place in Puerto Rico, like the University of Puerto Rico Mayagüez Campus (UPRM). So instead of using the coordinates, the first thing to do is to divide the map in two halves with a vertical line. Then, name each half with a binary value, which puts Puerto Rico in the 0 half. All this is shown Figure 2.2. [20]

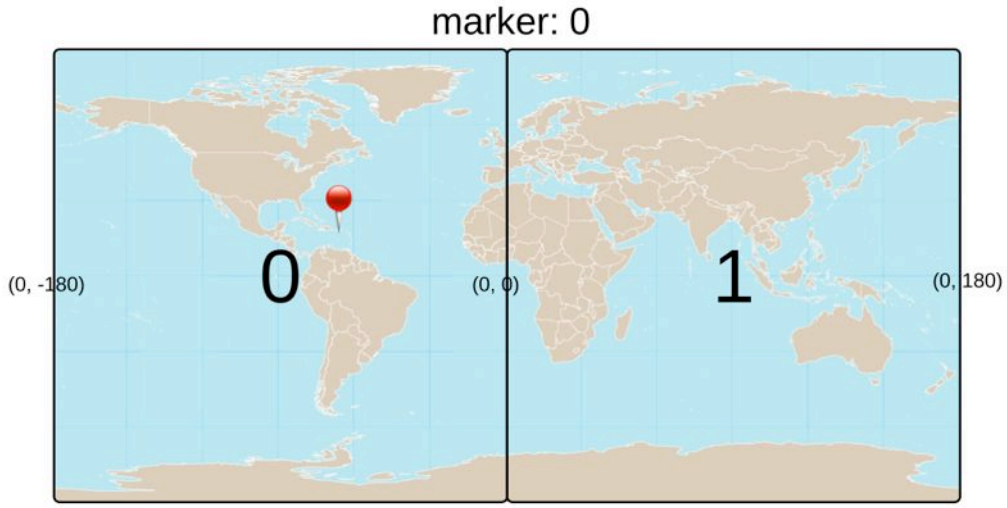

Figure 2.2 The first division for the geohash is vertical. PR's marker is in the "0" half.

Then another division is done inside the previous chosen side, but now horizontally. Puerto Rico is in the 1 half now, as shown in Figure 2.3.

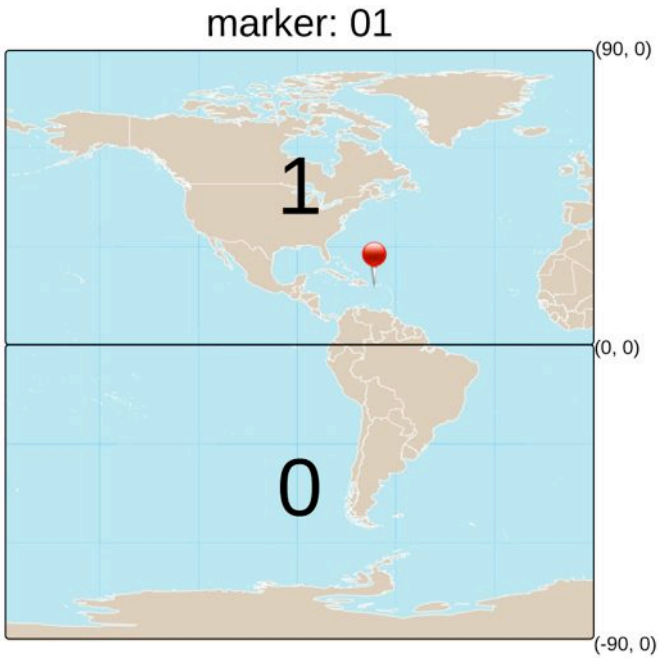

Figure 2.3 The second division is horizontal, inside the previous chosen side

Then another division is done, but vertically again. Puerto Rico is in the 1 half again, as shown in Figure 2.4.

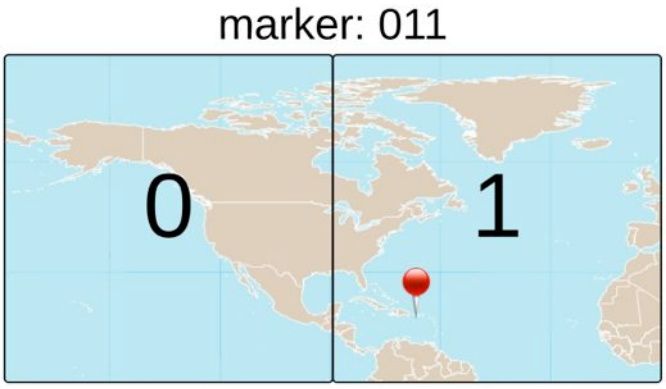

Figure 2.4 The third division is vertical again

This same process is repeated, each time creating alternating subdivisions that determine smaller areas. Eventually getting to a long binary number that represents a hyperlocal area inside UPRM, like where it's iconic "Pórtico" is located, as shown in Figure 2.5. [20]

Figure 2.5 Pórtico area inside UPRM after 40 binary divisions

Then, this long binary number can be encoded in 32 bits to make it much more manageable. This can be done with the character map shown in Figure 2.6. [19]

Binary	00000	00001	00010	00011	00100	00101	00110	00111
Decimal	0	1	2	3	4	5	6	7
Base 32	0	1	2	3	4	5	6	7
Binary	01000	01001	01010	01011	01100	01101	01110	01111
Decimal	8	9	10	11	12	13	14	15
Base 32	8	9	b	c	d	e	f	g
Binary	10000	10001	10010	10011	10100	10101	10110	10111
Decimal	16	17	18	19	20	21	22	23
Base 32	h	j	k	m	n	p	q	r
Binary	11000	11001	11010	11011	11100	11101	11110	11111
Decimal	24	25	26	27	28	29	30	31
Base 32	s	t	u	v	w	x	y	z

Figure 2.6 Geohash character map

Making the conversion results in the geohash **de0r84v9**, which represents the area where the Pórtico at UPRM is located.

2.1.2 How is it useful?

The size of the area covered by a geohash depends on its resolution, which is represented in the number of characters of the geohash. A short geohash has a small resolution and this means that it covers more area. A longer geohash has more resolution and covers a smaller area.

So depending on the resolution, it can group coordinates that exist inside that area. This can be useful when storing data in a database. An index could be done on this geohash, which helps in finding nearby data by taking entries with the same characters at the beginning of their geohash. [20]

Figure 2.7 Proximity search based on similar geohash

The markers shown in Figure 2.7 are indexed by their respective geohashes, at a resolution of 8 characters. The red markers are indexed with geohash **de0r84v9**, and the blue markers with **de0r84vc**. If these are searched in the database by the first seven characters of their geohash, which are **de0r84v**, then both groups of markers will be returned. This proves that the geohash helps in proximity searches.

But this is not perfect, as two points nearby could have very different geohashes. This happens near the locations where binary divisions for big areas are made, as shown in Figure 2.8. [20]

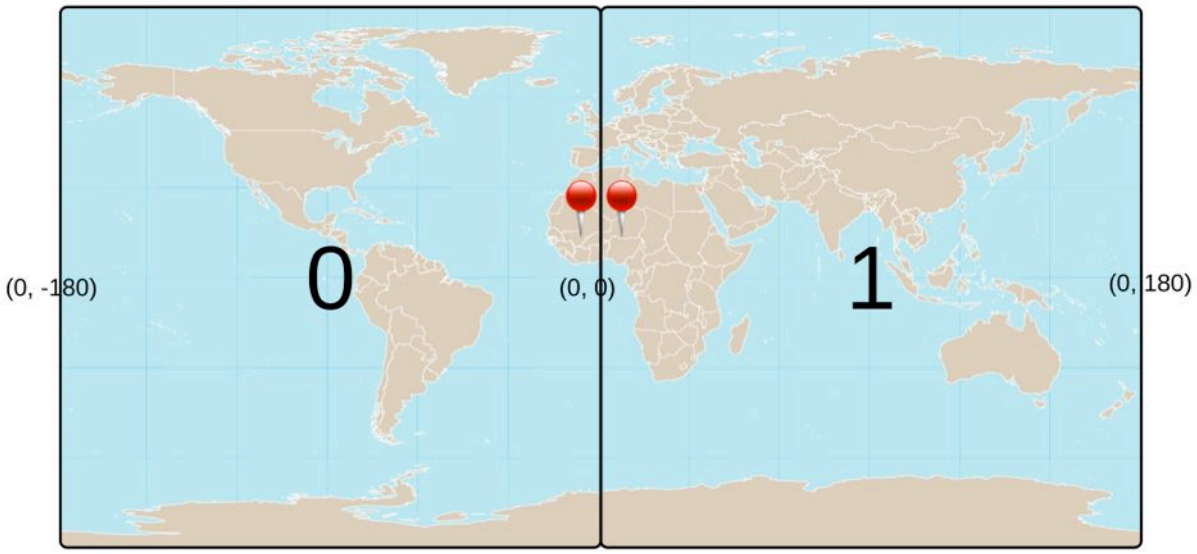

Figure 2.8 Not all near points will have similar geohashes

But this has already been solved, because the geohash is based on the same origins of the latitude and longitude coordinates. This means that with algorithms there can be any kind of conversion of coordinates to geohash, and vice versa. And there are also algorithms to find the surrounding geohashes around any geohash. With this, there is always a way to find points in proximity with a geohash based indexing, as shown in the next section. [20]

2.2 Technological Background

2.2.1 Geohash Javascript Demonstrator

A set of geohash routines for JavaScript was created by David Troy in 2008 and released under the MIT License [19]. He used the version 2 of Google Maps to demonstrate all these algorithms working in one application.

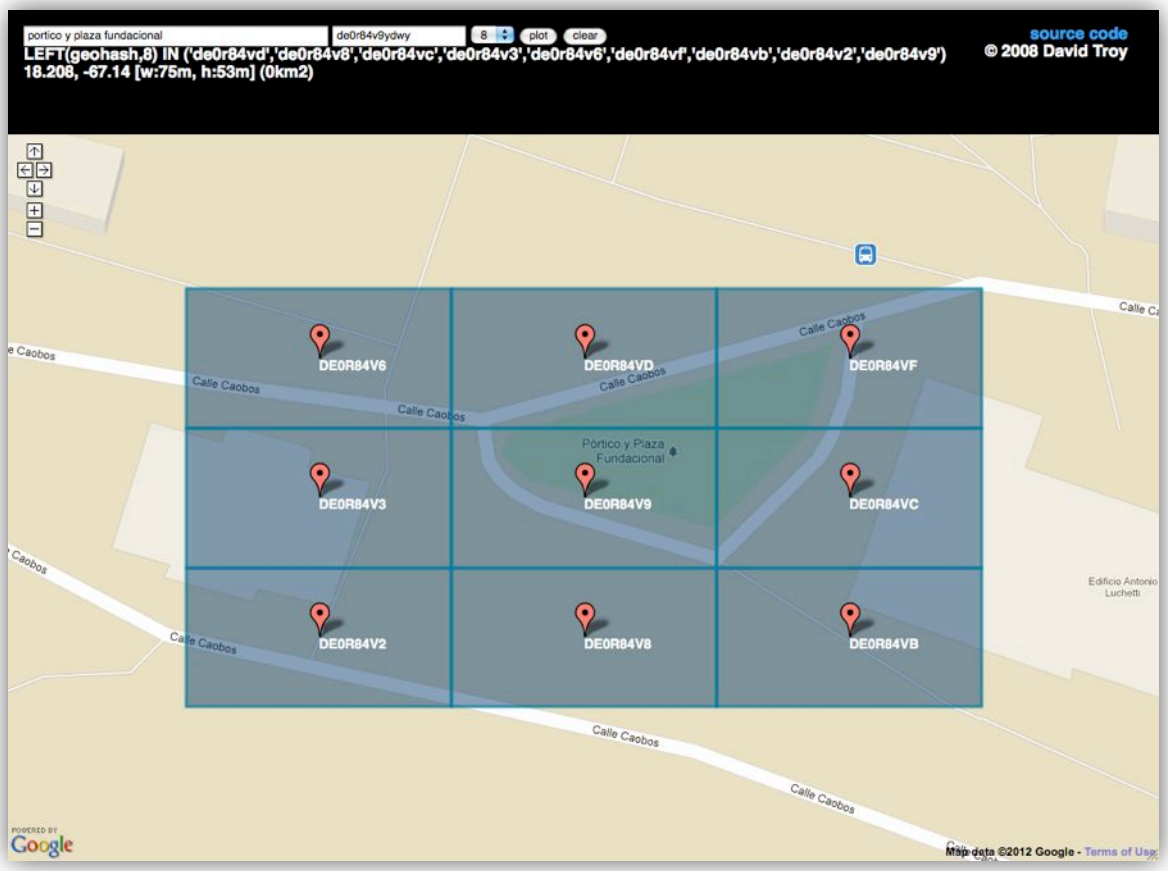

Figure 2.9 Geohash Javascript Demonstrator web application [19]

The goal of this demo is to create the SQL (Structured Query Language) query that would allow a geohash-indexed database to find data around a given point of interest. This POI is found using the Google Geocoding service, which returns the coordinates from a given address.

With these coordinates the application converts them to a geohash and, based on a chosen resolution, also calculates its 8 surrounding geohashes. All these are presented either as a SQL query or visually in a Google Map.

The complete demo is found at [19]. Its source code is also available to use and modify thanks to its MIT License.

2.2.2 Google Maps API

Currently in its third version, the Google Maps JavaScript API allows to embed maps in any web page. This latest version is much more efficient, which makes the platform useful for mobile devices.

These maps allow a great variety of functions. Any location from the world can be seen in map or satellite view. Places can be searched, calculate directions between two points, and even view them at street level.

But what has allowed this platform to grow in adoption and use has been the openness of its API, which allows virtually anyone to experiment with their own functionalities over the maps. From associating information to any point, or drawing anything in the map. These allow making ways, areas o even interactive overlays.

All these are possible because of an excellent library that does not require GIS knowledge. Everything is based on JavaScript, which makes it compatible with even more libraries like jQuery for many interaction possibilities.

It is the leader in its space, and because of the many years in the market, there is endless documentation on how to create any kind of application with their maps.

2.2.3 MongoDB

MongoDB is a database designed to have excellent scaling capabilities while keeping ease of use and manageability. Their basic data unit is the document. This is equivalent to a JSON (JavaScript Object Notation) and can have a dynamic schema. At the same time, MongoDB maintains great depth of functionality when compared to a traditional RDBMS (relational database management system). Figure 2.10 shows their terminology comparison. [1]

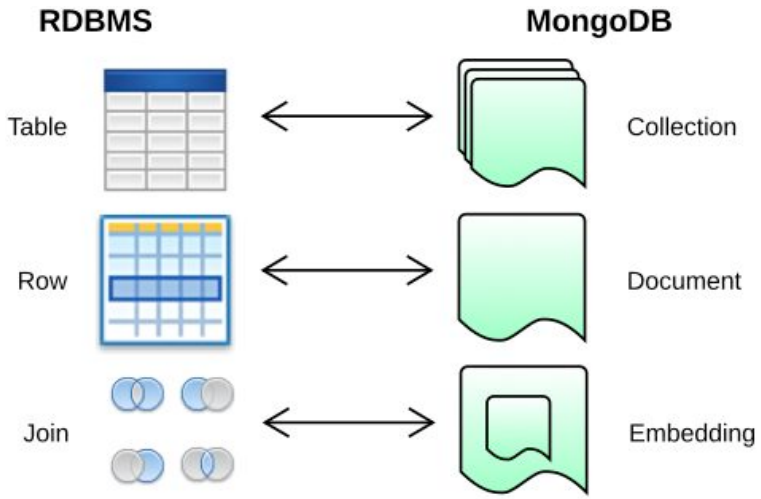

Figure 2.10 Comparison of MongoDB with traditional RDBMS

Documents have a dictionary-like structure in which its attributes can vary. These can also include non-atomic attributes and even other documents as attributes. These allow rich data objects without the need to do expensive multi-table joins like in a RDBMS. [3]

These documents are organized in collections and this is where the queries are made. These queries can be very powerful thanks to many built-in functions and the capability to include custom JavaScript code. These are also done efficiently because collections can have different types of indexes.

Some of the features that have made MongoDB the one of the most popular [18] NoSQL databases are: [1]

- Allows indexes (and secondary indexes) in any attribute of the document
- Allows dynamic queries (key/value stores do not)
- Native geospatial indexing (geohash powered)
- Data modeling agility because of flexible JSON-like schemas
- Atomic writes (on a per-document level) and fully-consistent reads
- Built-in support for horizontal scalability (auto range-based partitioning)
- Guaranteed data integrity through journaling and replication
- Flexible aggregation and Map/Reduce capabilities
- File storage of any size

But users have to be aware that MongoDB does not allow joins or multi-object transactions. Also, it is not compatible with SQL and doesn't provide ACID (atomicity, consistency,

isolation, and durability) guarantees over a series of operations [3]. So, if these are required features for the data, MongoDB is not recommended.

Finally, it must be noted that MongoDB is easy to install and manage. Making backups, importing, and exporting data is very simple. It also has utilities that allow interaction with the data with JavaScript, which is very useful for data migrations or any other manipulation or use of the data.

2.2.4 Restlet

Restlet is a REST web framework for Java. It faithfully models the elements that define REST, the architectural style of the web. These elements are mainly: resources, representations, components, and connectors. These principles allow network-based systems to be more scalable, efficient, and independent, because of loose coupling between clients and servers. [11] [12]

REST was created by Roy T. Fielding, considered one of the masterminds behind the Internet. He is one of the primary architects of HTTP 1.1, the backbone protocol of the Web. REST was a product of the realizations that he had while contributing to specifications of web standards. [11] [12]

Because Restlet was designed to take full advantage of the Internet, it is very easy to expose and consume web APIs. Everything is based on resources that are available for others through identifiers (like URLs). These resources can be interacted with using a set of standard HTTP methods like GET, PUT, DELETE, and POST. These methods serve the

basic needs of interaction with data, like create, read, update and delete (CRUD) [9]. And these resources are exposed in different types of representations (like HTML vs. JSON). All this is shown in Figure 2.11. [11] [12]

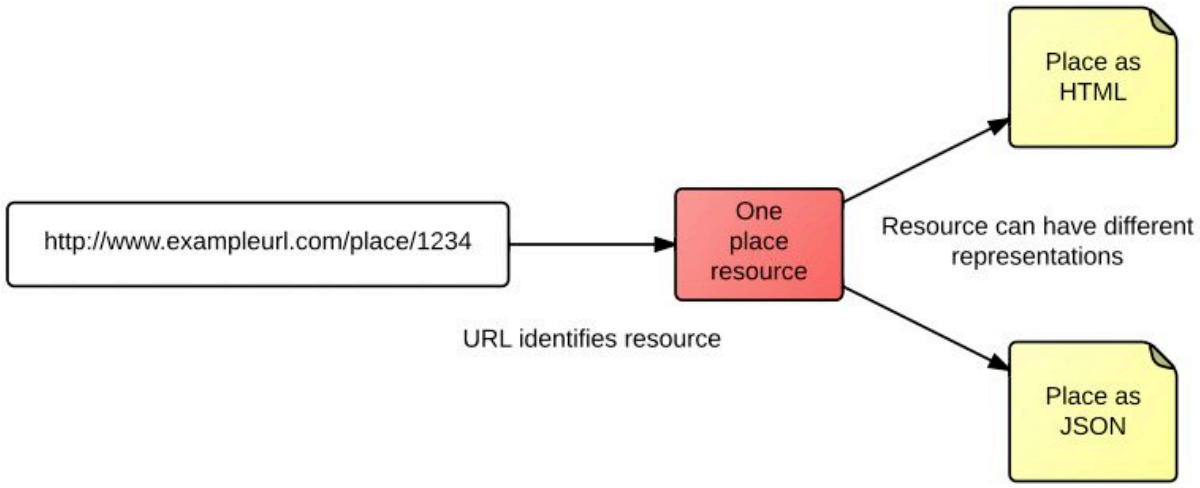

Figure 2.11 Basic interaction between identifiers, resources and representations

In Restlet, these resources are accessed through applications. In the applications is where most of the custom code lives, where the data models that represent the resources is manipulated.

Then, components contain different applications. The separation between applications and components allows loose coupling between applications and code reuse between components. So, the program at the server becomes a component that is connected to a client component (like a browser) through connectors (like HTTP). A simplified overview of this can be seen in Figure 2.12. [11] [12]

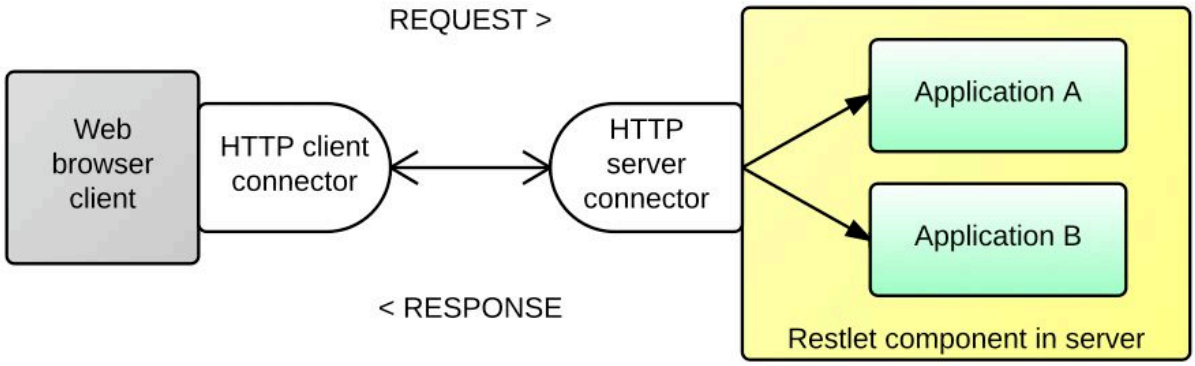

Figure 2.12 Basic overview of Restlet

With these elements Restlet provides a uniform interface between clients and servers, a standard way to interact with web resources. But another great feature is that it is designed to be compatible with scalable environments. For example there are no user sessions, which complies with the stateless constraint of REST, designed to keep up with scaling needs [6]. [11] [12]

2.2.5 Amazon EC2

The term "cloud computing" is used to define a variety of services, but there are 3 fundamental models: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). [13]

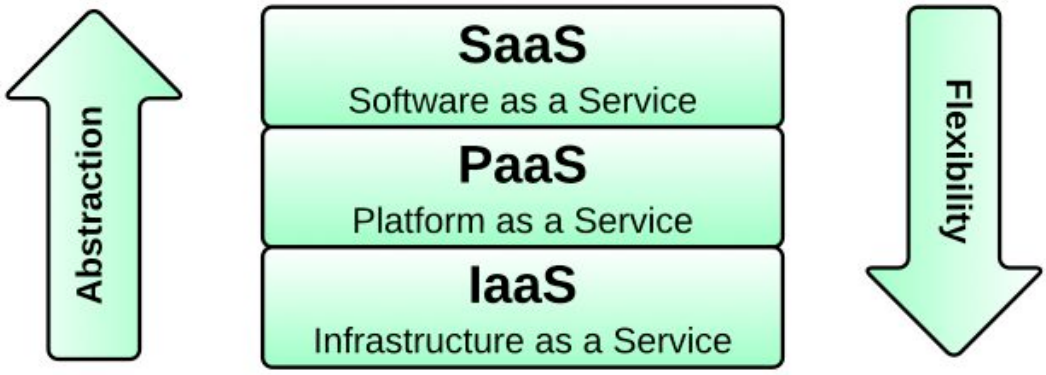

Figure 2.13 Three cloud computing fundamental models

As shown in Figure 2.13, as the level of abstraction is increased, these services become more specialized, thus offering less flexibility. Amazon EC2 provides services as IaaS, which basically allows complete flexibility of practically any computing resource that can be offered through a network.

These resources are implemented virtually, meaning that the different components can be modified through software and independent from each other. This provides an ideal environment to make any kind of computing requirement accessible, which allows users to make experiments or providing scalable web services. With this model, customers can only pay for the computing capacity they actually use. [2]

Amazon EC2 allows to:

- Create virtual computers from practically any capacity, called instances
    - From a selection of pre-configured images
    - Or from basic supported operating systems
- Manage their storage with Elastic Block Store, which allows to:
    - Create volumes of up to terabytes of size

    - Create snapshots of these volumes
    - Associate to any instance
- Configure security and network access to the instance
- Manage IP addresses dynamically to any instance

Finally, all these features are very easy to manage through an intuitive web interface and are compatible with many other cloud computing services from AWS (Amazon Web Services).

3 AREAS

The problem for creating areas with the crowd was presented, and the approach for solving it with the geohash was introduced. Also, the technologies that are going to be used to solve this problem have been presented. So, the focus of this section is to go in depth in the AREAS application.

The first part is a walkthrough of application features, first showing its RESTful interface that allows consuming and producing resources from different devices. Then, the different experiments done to manage places, from points to areas with geohash, will be presented. Then, the walkthrough ends with a brief demonstration of the final application by having one user add a place and its geographical area.

After the walkthrough, the architecture that makes AREAS possible is presented. All the technologies used are designed for allowing horizontal scalability, but they are not implemented for this yet. The usage patterns and data models must be proven before implementing a scalable architecture in an effective way.

Because of this, the last section presents more details of how the data models are actually managed in the document-oriented database. The process that happens when each user contributes to the area of a place is shown. And this will demonstrate the efficiency benefits for using this type of de-normalized database.

3.1 Walkthrough

The main functionality that AREAS wants to demonstrate is a new way to manage geographical areas. These could be used for anything a software developer wants. The use case that AREAS present is for users that take photos from their mobile device at places.

The first part of this walkthrough will focus on showing the RESTful API that allows consuming and producing resources. Because of capabilities for different representations, these resources will be compatible with web browser and native mobile apps at the same time. Then, the progress taken to produce a new way to manage areas with the geohash will be shown. Which allows anyone to easily contribute to the area of any place.

3.1.1 RESTful interface

The walkthrough starts by showing the places available in AREAS. There are two ways to access these places. The first way is to make a GET request to the following address, which can be called a URI (uniform resource identifier) because it identifies a resource.

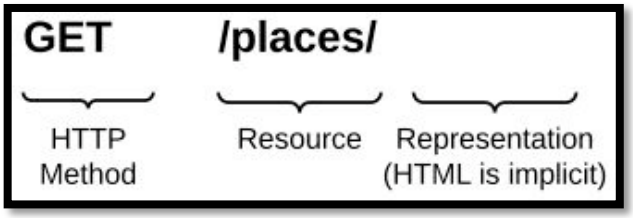

Figure 3.1 Parts of a RESTful HTTP request

By making the request shown in Figure 3.1, it is implicit that the client is a web browser and the resource will be responded as a HTML. The web page returned is in Figure 3.2, which includes a map and a list of all the places in the system without any constraints in location.

GET /places/

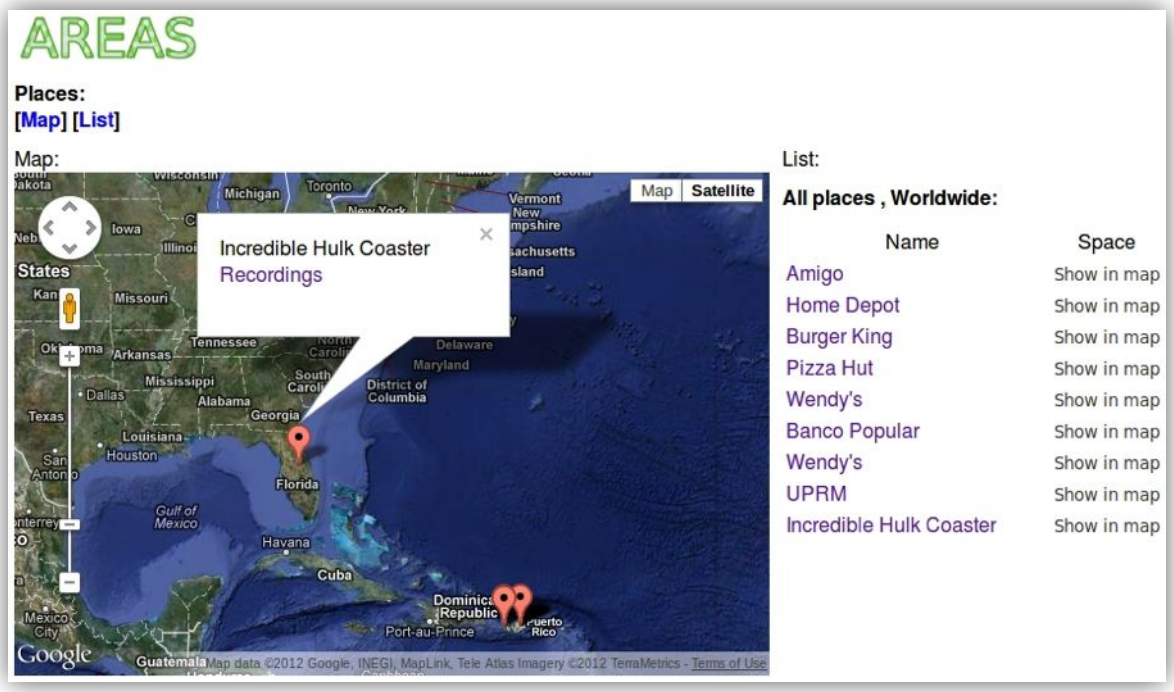

Figure 3.2 All places in the system requested as an HTML

This is just an example, showing one place in Florida, USA and the rest in Puerto Rico. But making this request is not a real use case if the system is full of places. In reality this resource

should be requested with a location reference, and this is the second way of accessing this resource.

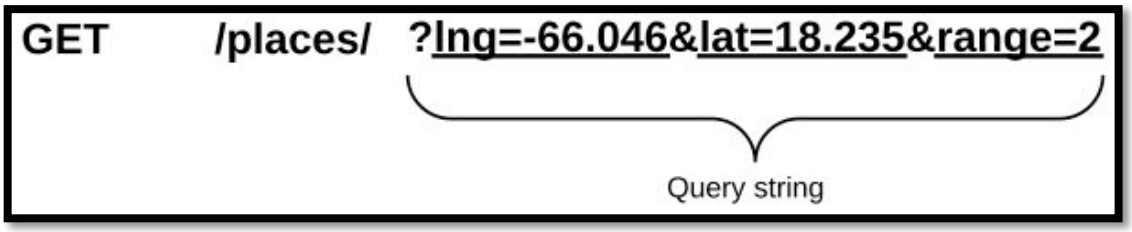

Figure 3.3 Accessing the places resource with a reference specified in a query string

A query string can be added to the request that will modify the resource and/or representation, as shown in Figure 3.3. In this, the mentioned location reference can be included, as the coordinates of a specific location and a radius to specify the range of area to include in the search.

GET /places/?lng=-66.046&lat=18.235&range=2

Figure 3.4 Attributes (underlined) in the query string changed the contents of the response

As shown in Figure 3.4, the query string contains the location and range attributes that modify the requested resource. This web page is different from the web page at Figure 3.2 because now the only places included are the ones inside the specified area.

These previous resources received as HTML representations are useful for a web browser client, but not for a native mobile application, like an iPhone app. But because AREAS provides a RESTful interface, these same resources can be requested specifying a representation that the iPhone app can understand.

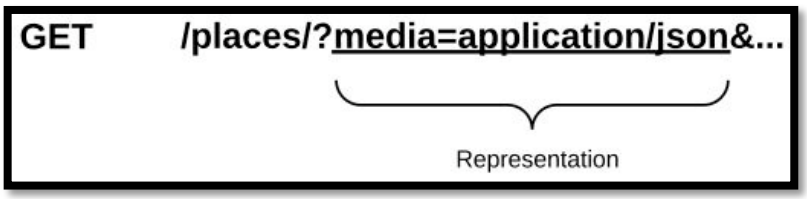

Figure 3.5 Specifying a representation as an attribute of the query string

As shown in Figure 3.5, this can be done specifying a media in the query string. In this case it will be JSON.

GET /places/?media=application%2Fjson&lng=-66.046&lat=18.235&range=2

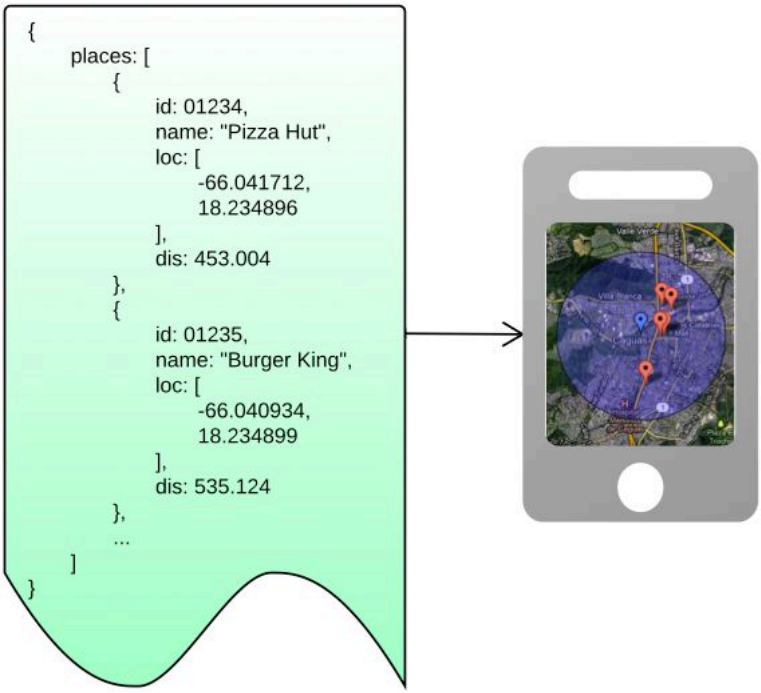

Figure 3.6 The same resource can be requested as a different representation. In this case, the JSON representation is compatible with an iPhone app client.

The JSON response can be seen in Figure 3.6, which shows a document with data from different places inside an array. For each place there is information like its name and coordinates, but it also includes the id.

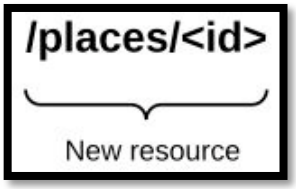

Figure 3.7 URI structure for one specific place

This id can be used to access a resource from one specific place, using the URI structure shown in Figure 3.7.

GET /places/506b5995e4b0dd560014826f

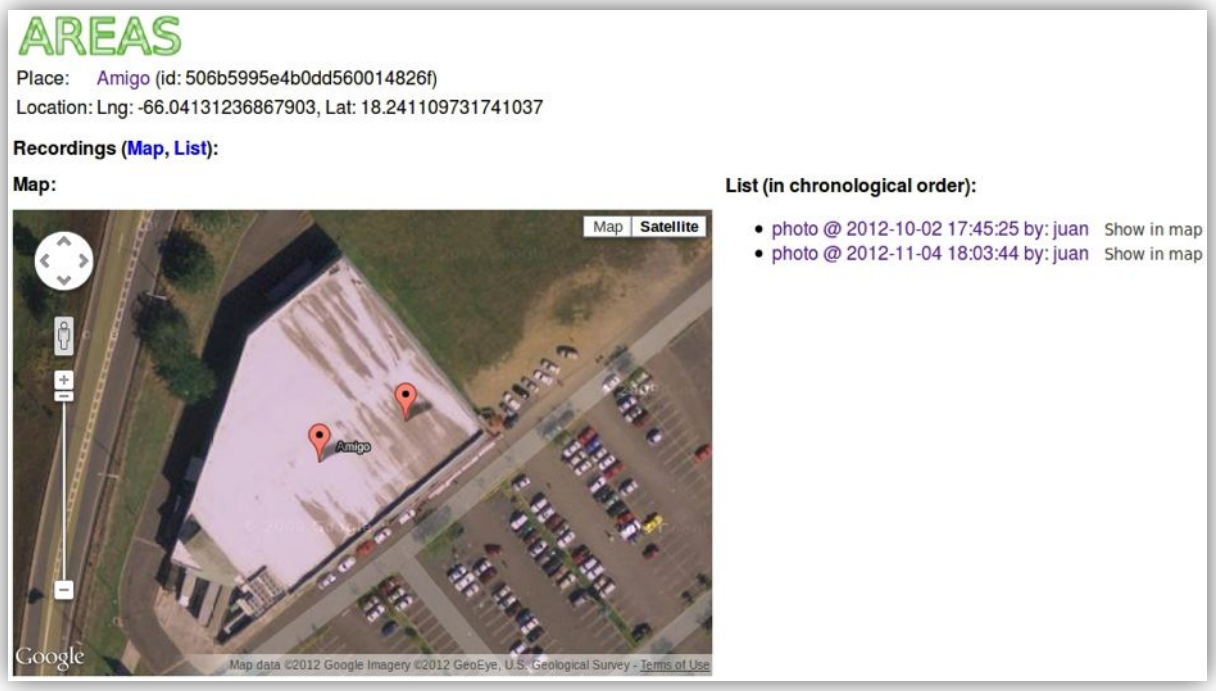

Figure 3.8 HTML representation of the resource for one place

Figure 3.8 shows the HTML representation of a response when an id is specified in the URI. This web page shows the information from one place, like name and coordinates. But it also includes something more, the links to the photos taken at the place.

These links have the URI structure shown in Figure 3.9.

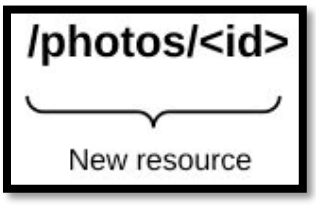

Figure 3.9 URI structure from one specific photo

These links show a new root URI (/photos/) to manage photo resources. By clicking one of these links, the web page from Figure 3.10 is shown.

GET /photos/5096e6a6e4b0bfe210b8dab5

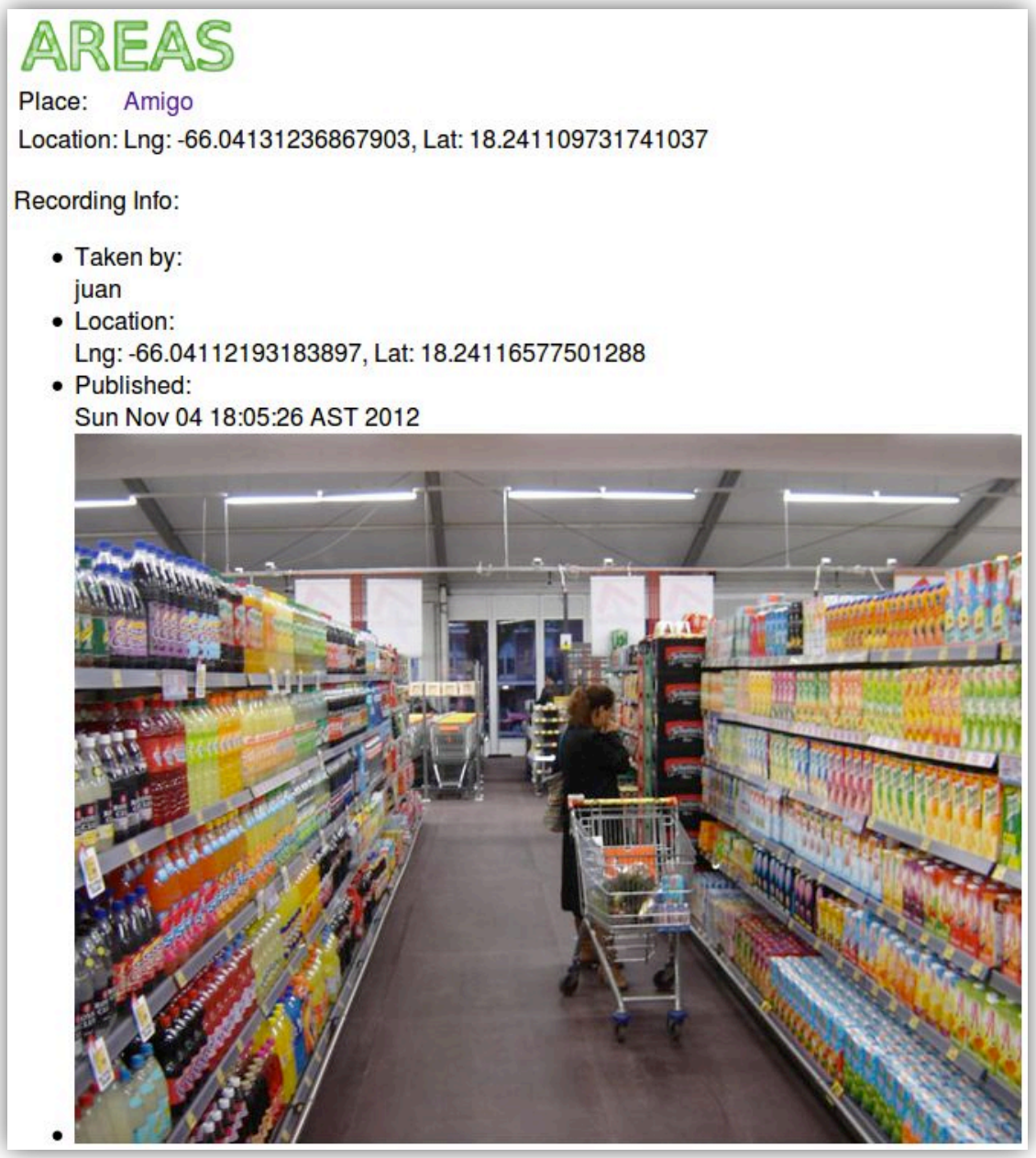

Figure 3.10 HTML representation of the resource for one photo

This page shows the different attributes of a photo resource like: the place it was taken, the specific coordinates, and also the picture file.

Finally, a RESTful web service also allows contributing resources. To upload a photo, a URI similar to the one from Figure 3.9 is used, but without an id. Also, the request method must be a POST, not a GET. The following figure shows a simplification of how this request would look coming from a client, like an iPhone.

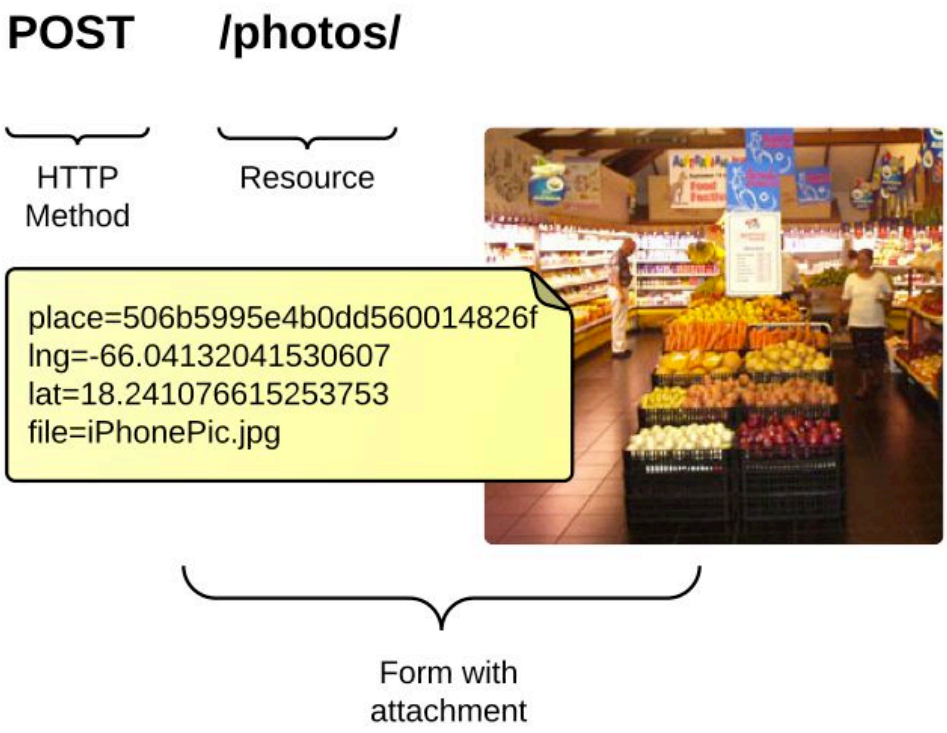

Figure 3.11 Simplification of a HTTP POST form with a photo attachment

Figure 3.11 shows that the request includes a form. The form includes all the information needed to add a new photo resource, like the place to associate the photo, the coordinates from the iPhone, and the picture as a file.

This first part of the walkthrough proves that AREAS is a RESTful web service.

3.1.2 Geographical areas

By having places include their area, the "POSTing" of a photo from a mobile device may not need to specify the place. But this is just a simple use case, in reality, these areas could be useful for many other things.

The main feature of the AREAS project is the new way of managing geographical areas. The experiments made to get to the final product will be the focus of this second part of the walkthrough.

3.1.2.1 Experiment 1: "Seed" concept

The simplest way to identify a place in a map is with a point, which is nowadays represented by latitude and longitude coordinates. Then, the simplest model for a place would be to add a name to the coordinates attribute, as shown in Figure 3.12.

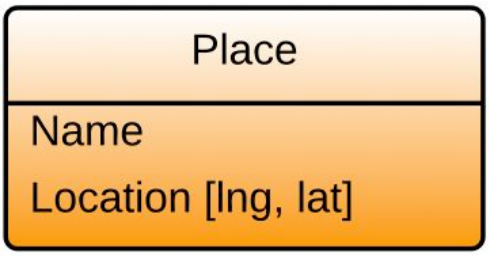

Figure 3.12 The simplest model for a place entity

Then, if there is interest in adding something complex, like an area, this could be a separate entity. This entity could be related to the place as shown in Figure 3.13. In MongoDB, this

relation is done by linking the area document to a place document with a reference, as shown in Figure 3.14.

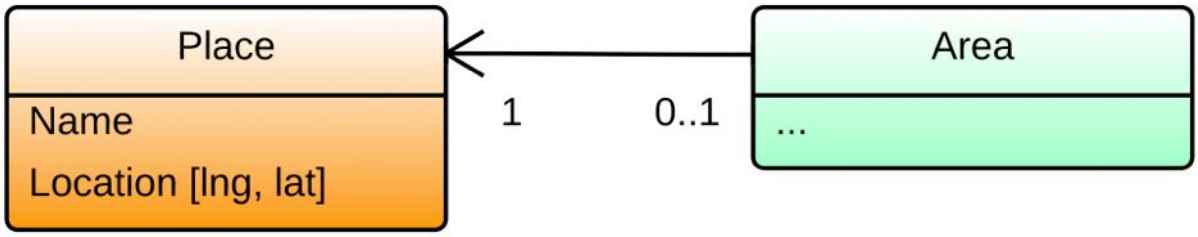

Figure 3.13 Relation between a place and its area

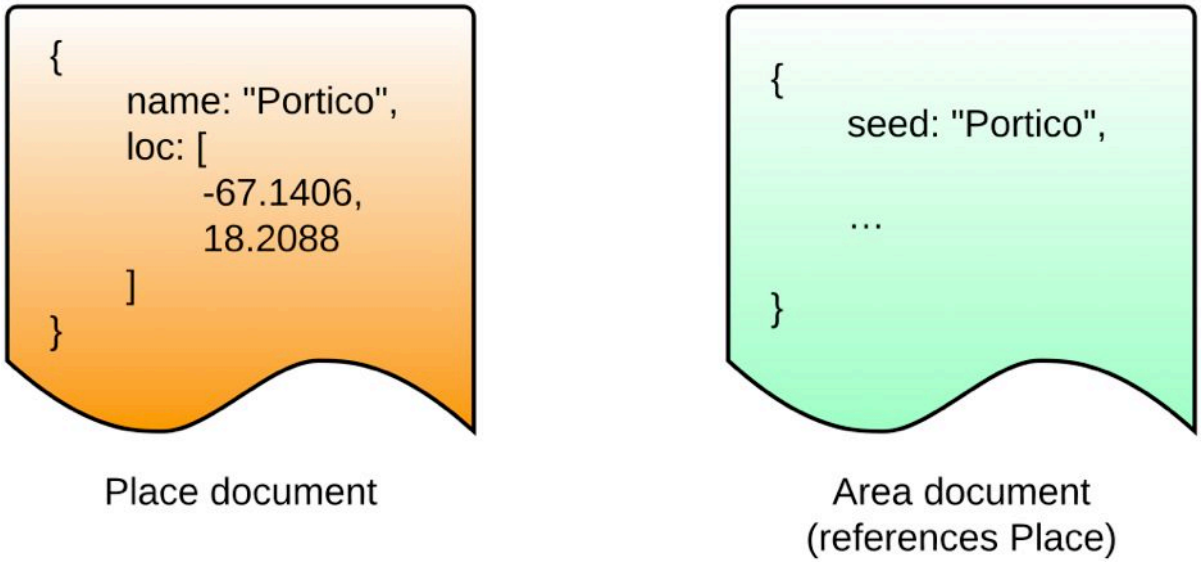

Figure 3.14 In MongoDB, the area document references the place document. The place document is considered the "seed".

This approach for dealing with areas is being referred as a "seed" approach. This means that when dealing with the area of a place, there will always exist a relation to a basic "seed" place.

Creating seeds for any place is a very simple process, but the real value of this approach is being able to utilize POI data that is widely available. This data could be accessed using an

API as mentioned in section 1.1, but there are also services that allow users to download their data to use without constraints. Because of this, the experiment of downloading publicly available information from OpenStreetMap [16] and GeoNames [8] was done. The data set was limited to the island of Puerto Rico, to make the data manageable.

Importing OpenStreetMap data to MongoDB required tools that are out of the scope of this report [15], but importing the GeoNames POI data set was very simple thanks to different MongoDB utilities. These allowed importing from a file and then modifying its schema with JavaScript, all directly from the Linux terminal.

Figure 3.15 Open source POI data imported from OpenStreetMap and GeoNames

These imported data sets are shown in Figure 3.15. As mentioned, these POI data could be used as the seed for relating areas to them. But this functionality will be integrated to AREAS in the future. For now, the users create the seed model.

3.1.2.2 Experiment 2: Polygon

Another experiment that was done was to create polygons with Google Maps, like the ones shown in section 1.2. When 3 or more markers are connected in a path, these can draw an overlay over the map to create a polygon. This process is shown in Figure 3.16.

Figure 3.16 Polygons created with a path of coordinates

This polygon could represent the area covered by a place. The coordinates of these markers are stored as an array of coordinates.

3.1.2.3 Experiment 3: Geohash

The final experiment was making areas with the geohash. This was developed modifying the excellent *Geohash Javascript Demonstrator* application mentioned in section 2.2.1. The application was also updated for version 3 of Google Maps.

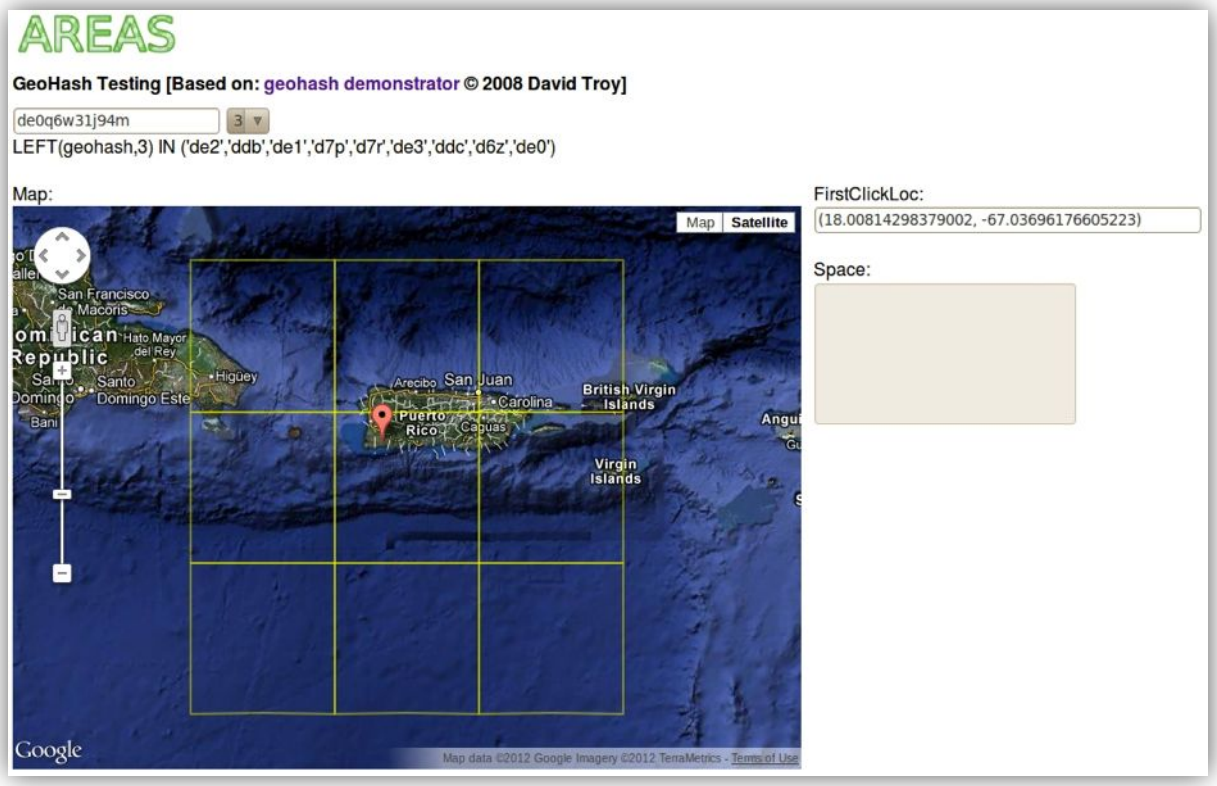

Figure 3.17 *GeoHash Testing* experiment, based on [19]

This modification is called *GeoHash Testing* and its main screen is shown in Figure 3.17. Using the capacity of drawing the 8 surrounding geohashes of any geohash, this experiment is based on the concept of creating some sort of "growing grid" that allows choosing geohashes. These chosen geohashes can be used to define the area of a place.

The first step is to click in any part of the map. This will create what can be called the "seed grid", as was shown in Figure 3.17. The center of the grid is the geohash for the coordinates from the location where the map was clicked. This is similar to the *Geohash Javascript Demonstrator* [19] application, but the similarities end here.

This grid is now interactive, designed to choose how many geohashes a place covers. These geohashes are selected by clicking inside the quadrangles of the grid. For example, if the center geohash is selected, it will create an overlay in the map and add this geohash to the "Space" form beside the map, as shown in Figure 3.18.

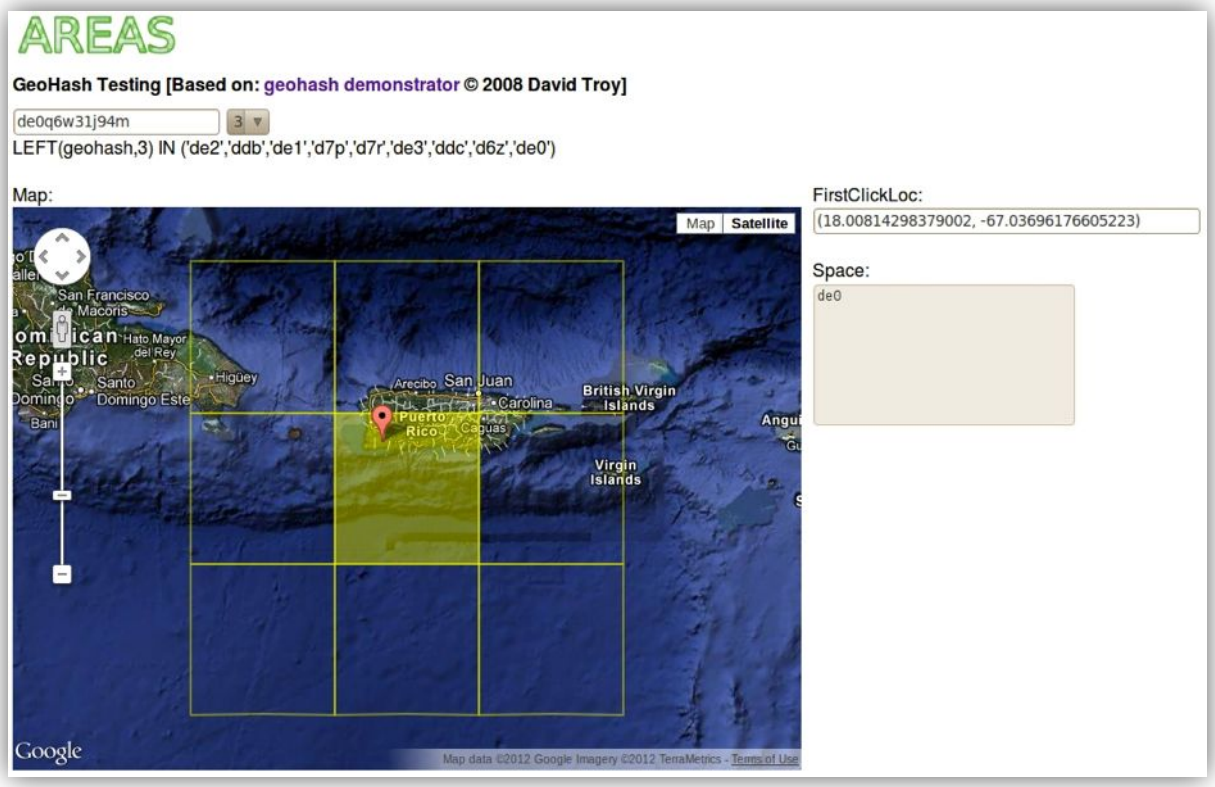

Figure 3.18 Center geohash chosen covers the southwest part of Puerto Rico

But if the user clicks in any of the surrounding geohashes, aside from selecting this geohash, the grid will also grow. A clicked geohash will always activate its surrounding geohashes for possibility of being chosen. This is shown in Figure 3.19.

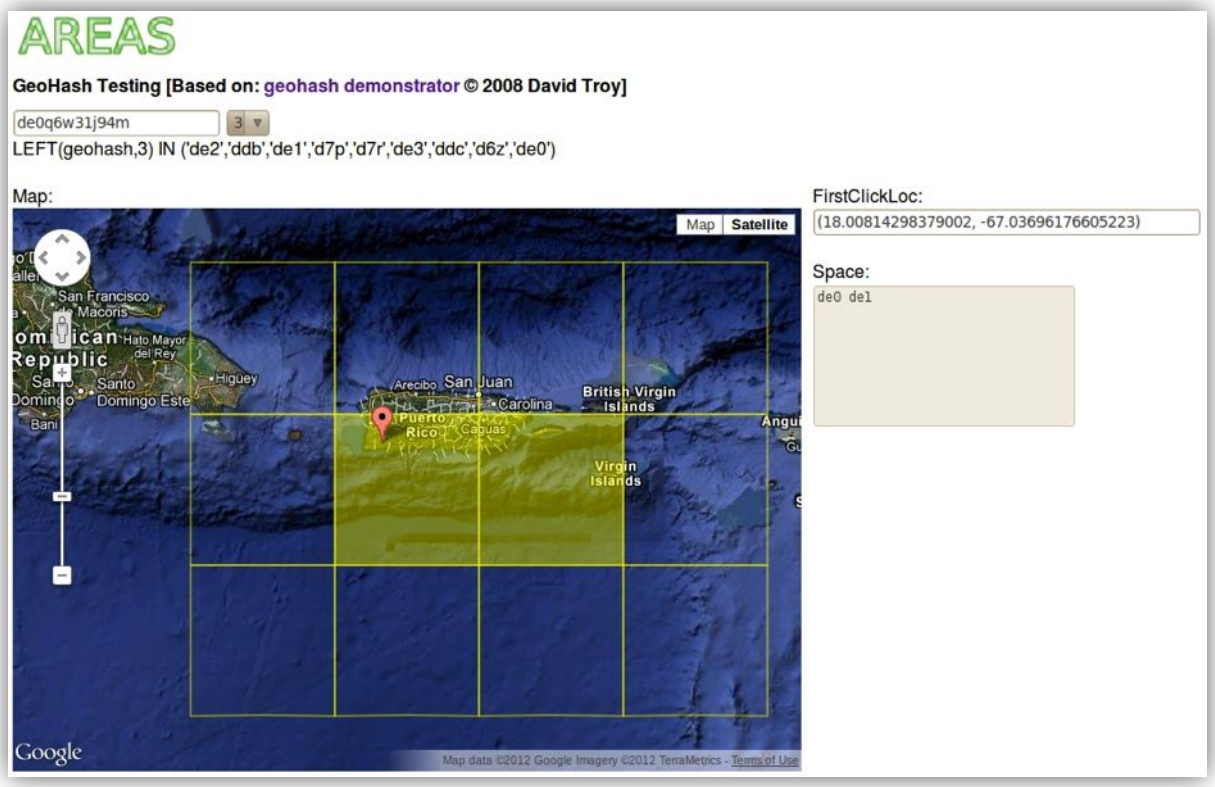

Figure 3.19 The next geohash to the right covers the southeast part of PR and its Vieques island. Choosing this also expands the grid.

This functionality allows the choosing of continuous geohashes that can define the area of anything. Up to this point, the figures show geohashes with small resolution, which makes them cover big areas. But this application also allows users to increase the resolution.

By changing the resolution, there will be less “extra” area, but more geohashes have to be chosen. Figure 3.20 shows how Vieques needs 17 geohashes to be covered at resolution 5, while this same island was covered with only one geohash of resolution 3 in Figure 3.19.

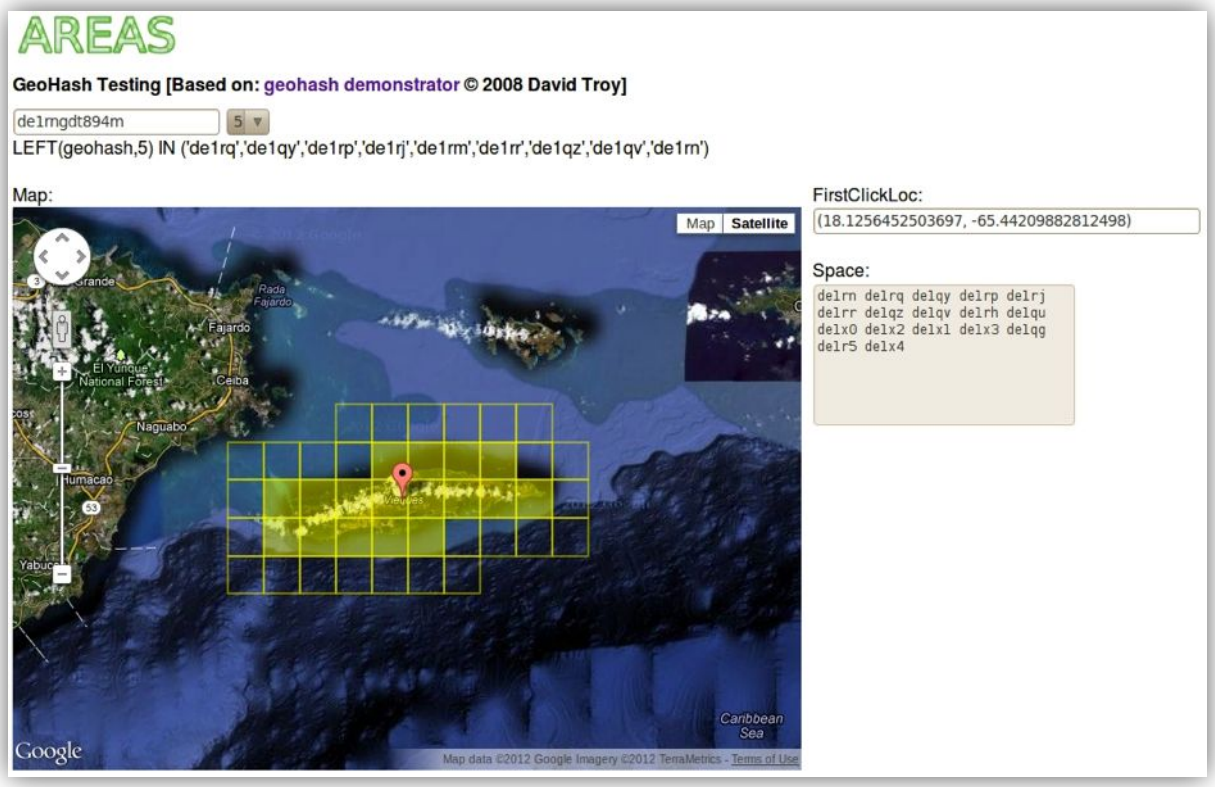

Figure 3.20 Vieques covered with 17 geohashes of resolution 5

3.1.2.4 Final product

The final part of this walkthrough will show how to add a place to AREAS. The first step is to simply add a "seed place" by clicking at a location in the map, adding a name and clicking the 'Save' button. This is shown in Figure 3.21.

Figure 3.21 A "seed place" added with a location and a name

After having the “seed place”, an area can be associated to it. As shown in Figure 3.22, this new screen is a simplification of the *GeoHash Testing* experiment from section 3.1.2.3.

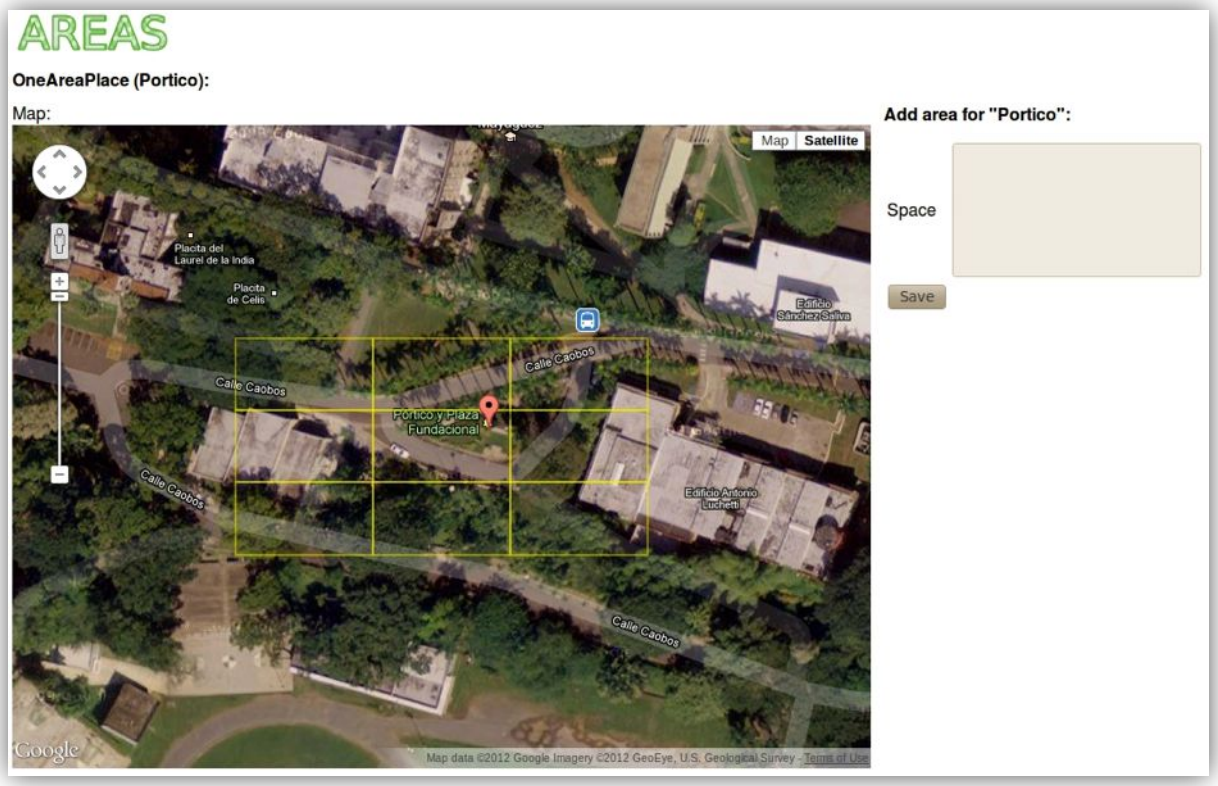

Figure 3.22 Seed grid with only 8-character resolution geohashes

This simplification is based on the goal of adding hyperlocal places. For this reason, only one default resolution is available. The 8-character resolution was chosen because having more resolution would need too many geohashes to be chosen per (hyperlocal) place.

In this screen the geohashes can be chosen in a similar way to the experiment. The grid will keep growing until the geohashes that cover the area of interest are chosen.

Finally, the user can click ‘Save’ to contribute his/her geohashes to the place. Figure 3.23 shows all this.

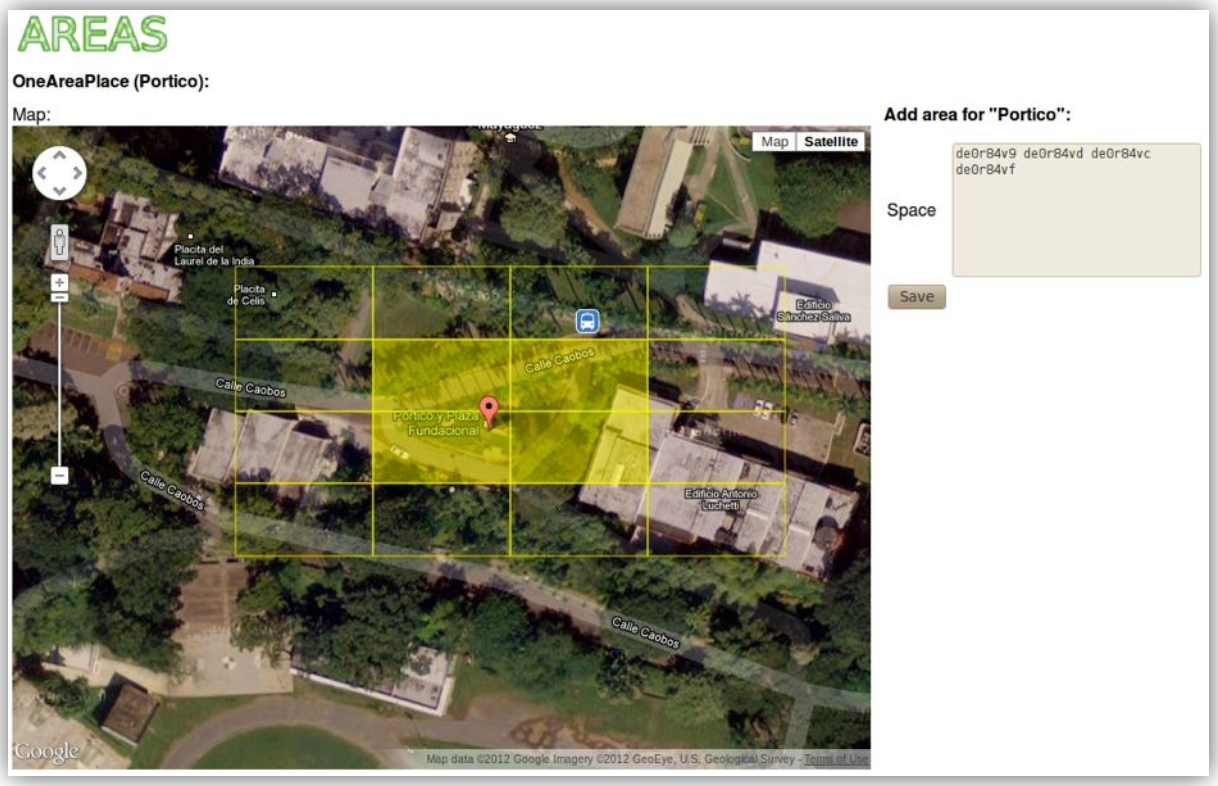

Figure 3.23 The user chooses the geohashes and saves them

Section 3.3 will show more details of how data is managed when different users contribute to the area of a place.

3.2 Architecture

The underlying architecture of AREAS is based on technologies that can effectively scale horizontally. But as has been mentioned, these are not yet specifically implemented to do so.

In order to scale, there needs to be some prior knowledge of the underlying technologies. And also, there needs to be an understanding of the application usage and data models in order to do an effective scalable architecture. This first version of AREAS is necessary in order to learn these things.

So the focus of this section is to explain the technologies being used and how they are implemented. Starting by showing an overview of the cloud architecture, then the underlying web framework, and finally the database technology.

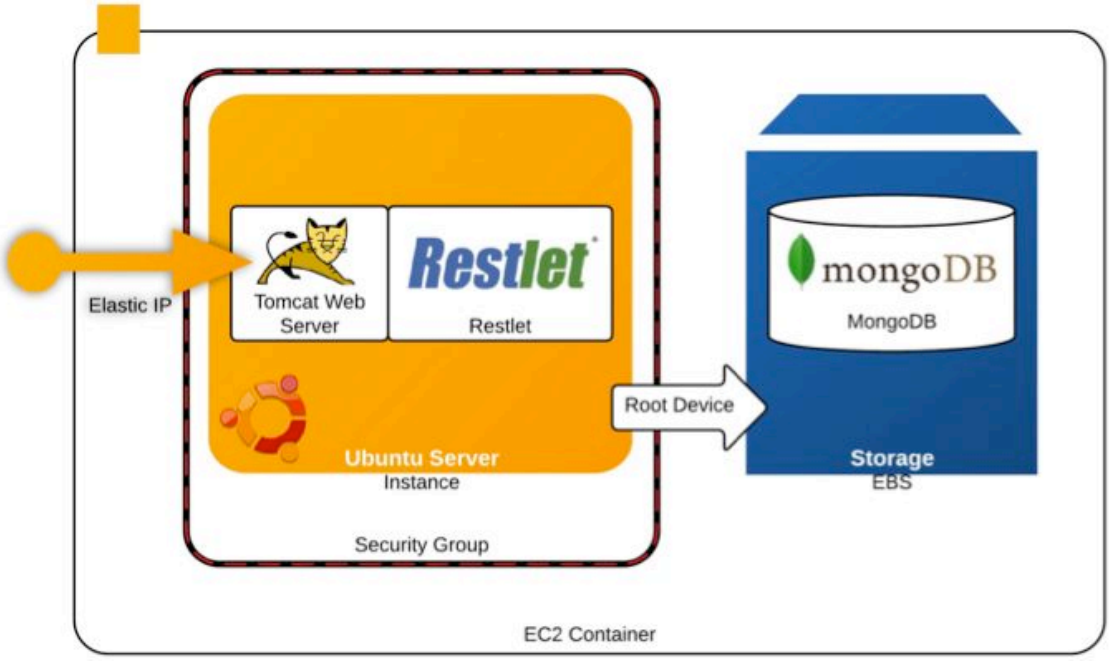

Figure 3.24 AREAS cloud architecture

3.2.1 Amazon EC2

The way to serve web sites or web services in the past was by buying or renting a physical server. This server is a computer with its basic components like processor, RAM and storage drive. In Amazon EC2 is another paradigm, were computing resources work virtually.

For AREAS, the backend architecture was done completely with Amazon EC2. Figure 3.24 shows a diagram of it and this section will explain each part.

3.2.1.1 Instance

Lets start with the concept of an instance, which in principle could be considered a complete virtual computer. Interestingly, because of the variety of hardware that underlies the virtual environment that Amazon creates, these instances are not classified by processor speed. Instead they use the ECU (EC2 Compute Unit), which is equivalent to the CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor. [2]

So, these instances can be of different capabilities that vary in number of ECUs, CPU cores and RAM. These instances also include a storage drive that allows them to be managed through an operating system, but this default drive is not persistent (this will be explained in section 3.2.1.2).

Because AREAS is still in development, there is no need for much computing capacity. For this reason the instance used for AREAS has 1 ECU, 1 core and 613 MB of RAM, which is the most basic instance offered by EC2. The operating system for this instance is Ubuntu Server.

3.2.1.2 EBS

EBS (Elastic Block Storage) storage is used to preserve data from an instance. Similar to the computing capabilities, the user can determine the size wanted for these volumes.

Like mentioned earlier in section 3.2.1.1, the default storage drive (root device) that the instances have is not persistent. This means that if the instance is shut down all the data will

be lost. Because of this, the instance that serves AREAS has an EBS volume as its root device.

This separation allows much flexibility. For example, it allows users to make snapshots of the volumes. These could work as backups of the data, to create new images, or creating a new volume with more capacity.

3.2.1.3 Security Group and Elastic IP

It can be seen in the figure that the instance is "guarded" by a Security Group. This is what manages the network connections of the instances. These determine which ports are open or closed in the instance. AREAS has the SSH port open to access it, and the HTTP ports open to be available in the Internet.

The Elastic IP is what manages the Internet address of the instances. When an instance is created, EC2 automatically provides an address in order to be able to communicate with it. This address is unique and non modifiable. So, in an environment where instances can be constantly created or destroyed, is convenient to separate this functionality. The Elastic IP allows AREAS to have a static IP that can be assigned to any instance.

3.2.2 Restlet

It can be seen in Figure 3.24 that Restlet is accessed through a traditional web server, which is Tomcat. The Restlet framework allows the RESTful interface of AREAS, and an overview of what happens inside is shown in Figure 3.25.

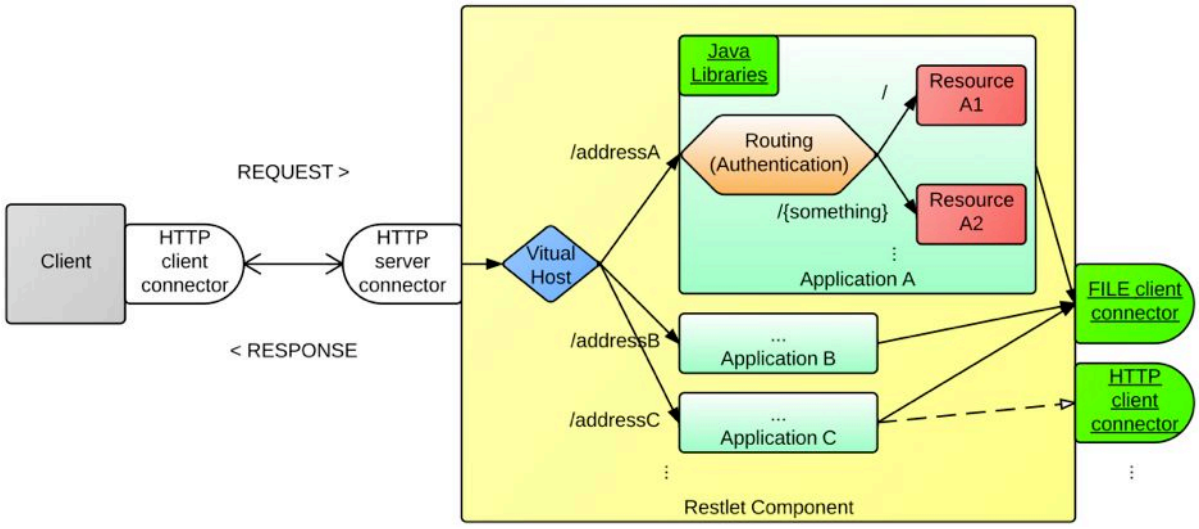

Figure 3.25 Overview of the AREAS architecture in the Restlet web framework [12]

Restlet is based on the resource orientation paradigm [11], in which a resource is requested and then a representation of this resource is responded. This communication is possible through connectors that the client and server have. And both can understand each other because they use the same HTTP protocol.

In Restlet, the AREAS web service is packaged in what is called a component. When a request enters the component, the first thing that happens is that it passes through a virtual host. This virtual host can determine which application manages the resource based on the first level of the URL.

Then, inside an application, the first thing that is determined is which resource is wanted. This is also identified by the URL, but now by its second level. In this level, if the resource does not require user authentication, the process finishes by responding with a representation of the resource.

If the resource does need authentication, this is verified from the application. It is important to note that authentication is verified in each request because Restlet does not manage sessions, complying with REST specifications.

This authentication needs access to the database. This brings an important point, which is, that from the applications is where the custom programming lives inside the Restlet framework. From these applications other resources are accessed, from the ones accessible through Java libraries, to anything that can be accessed through Restlet connectors.

In AREAS, one of the Java libraries used is the *MongoDB Java Driver*, which allows accesses to the database. Other libraries include the *BCrypt* library to do password hashing, and *jGeohash*, which includes many functions from the geohash JavaScript utility used in the front-end.

The Restlet connectors mentioned earlier abstract the way of accessing resources to make it in a RESTful way. In AREAS, many applications access the file system of the server, so this is done through a FILE client connector. The figure also shows another HTTP client connector, which is included to show that the applications can access external APIs.

For example, in AREAS it is beneficial to have access to the time zone of any place. GeoNames offers this data as a download or a web service [8]. The HTTP connector could

have been used to access the GeoNames web service. But as mentioned in section 3.1.2.1, the data was downloaded and imported to the database.

3.2.3 MongoDB

As mentioned, the AREAS MongoDB database is accessed through a Java library. This library allows many interactions with the database like the creation of collections, their indexes, making queries, and saving data.

Because AREAS is still in development, the architecture of the database is simple. There is only one database with various collections that include all the data from the documents, as shown in Figure 3.26.

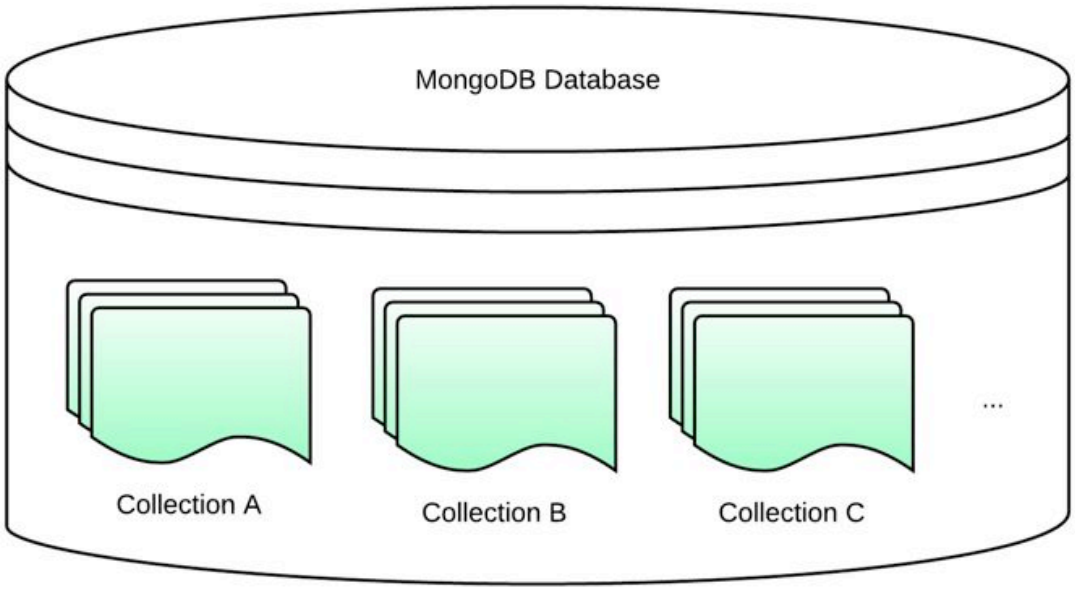

Figure 3.26 Basic AREAS database architecture with MongoDB

There are many collections in AREAS, including the Users collection and the GeoNames POI collection. For collections that include location data in latitude and longitude coordinates, MongoDB's geospatial indexing is used. This helps making location-based queries more efficient. It also has a spherical model option that takes in consideration the planet Earth's curvature to calculate distances more accurately. [1]

The interesting part of the database is the way the places with areas are modeled. This will be the focus of the next section.

3.3 AREAS Place Model

The model used to represent the area of a place is based on allowing the crowd to contribute. This requires a model that can relate all user interactions in an efficient way. This will be shown in this section, which is what happens in the back-end when users do the actions presented in section 3.1.2.4.

3.3.1 Basics

AREAS uses the same "seed" concept explained in section 3.1.2.1, where the area of a place references a basic model of a place.

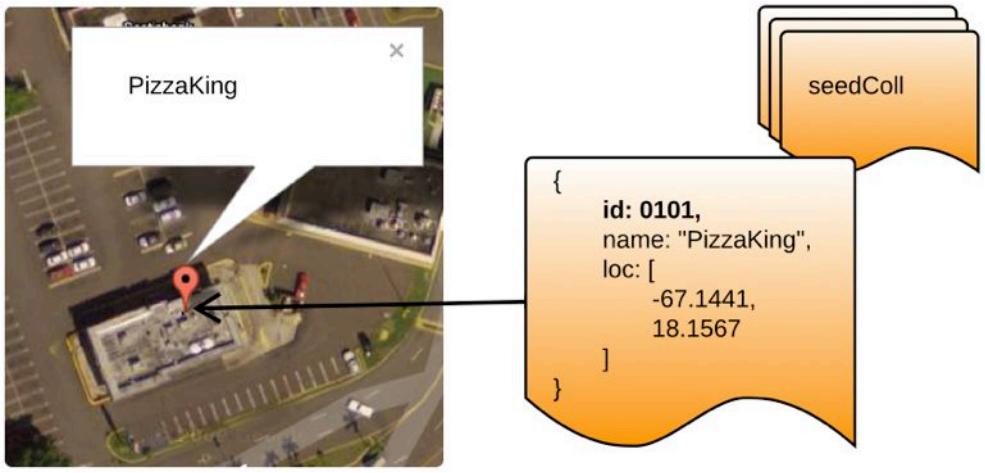

Figure 3.27 A "PlaceSeed" document created, which must exist before adding its area

Let's call this basic model a "PlaceSeed", with a schema that includes the place name, location coordinates, and a unique id. PlaceSeed documents are stored in a collection called "seedColl". All this is shown in Figure 3.27, with a place called "PizzaKing".

After having a PlaceSeed document for the place called PizzaKing, the next step is to associate an area to it. These areas are defined by geohashes chosen by users. For example, let's say that User1 adds the 2 geohashes shown in Figure 3.28.

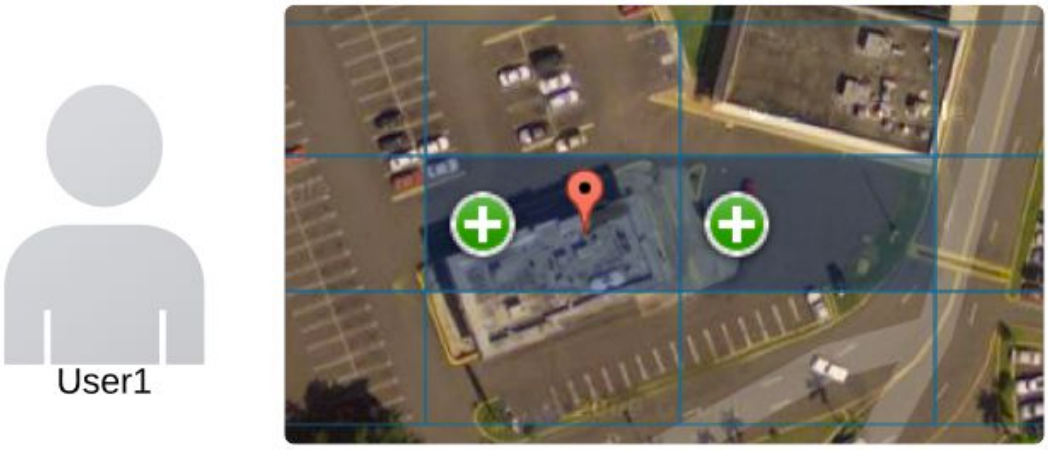

Figure 3.28 User1 adds two geohashes that he/she thinks belong to the place

These two geohashes will create a new type of document each. Figure 3.29 shows in more detail what happens.

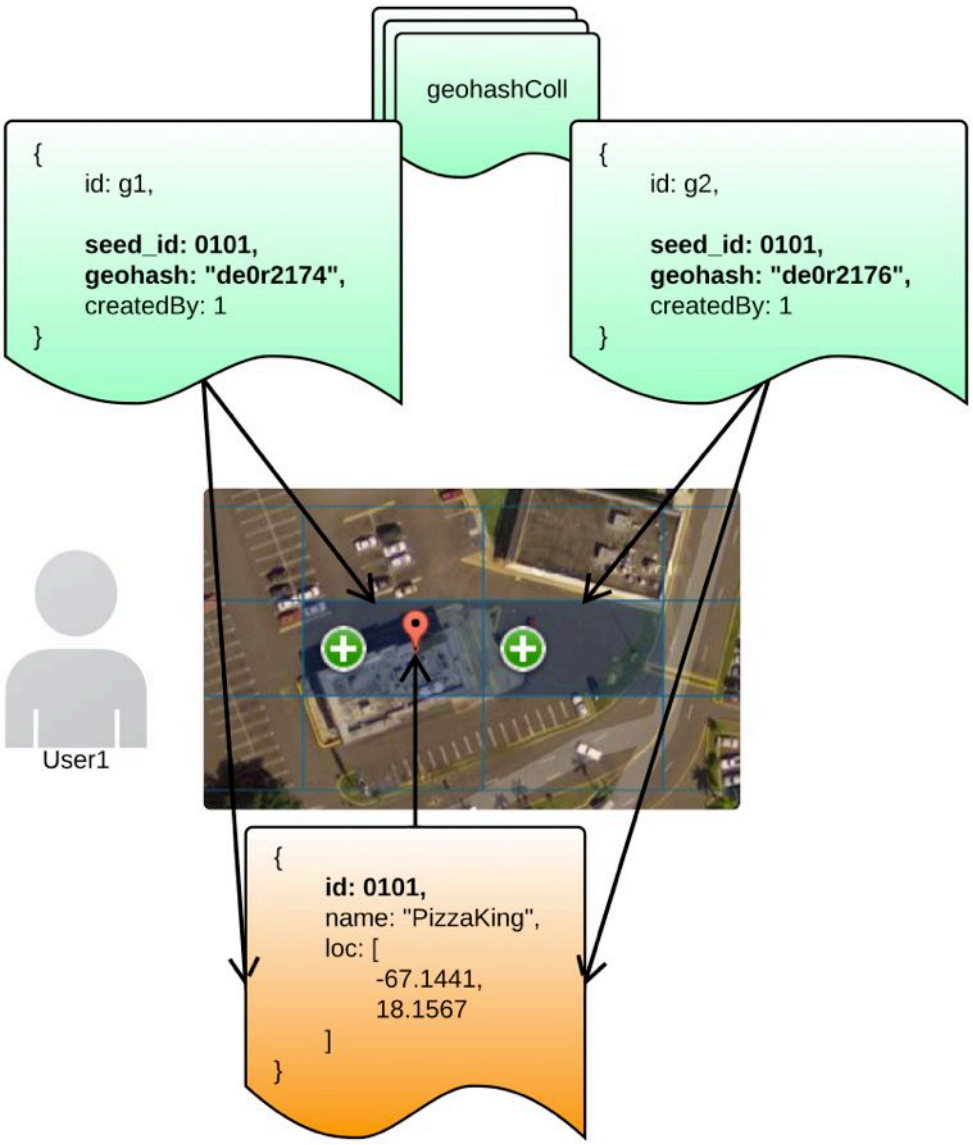

Figure 3.29 "PlaceGeohash" documents are created to associate a geohash to a place. Their unique constraints are the geohash and the reference to the place they belong.

The model of these new documents can be called a "PlaceGeohash" and are part of a new collection called "geohashColl". The key attributes of these new documents are the chosen geohash and the reference to the PlaceSeed. They also have a unique id that is convenient to have. The two PlaceGeohash documents created have the ids "g1" and "g2" respectively.

For now, the schema of a PlaceGeohash and it relation to a PlaceSeed can be modeled like Figure 3.30.

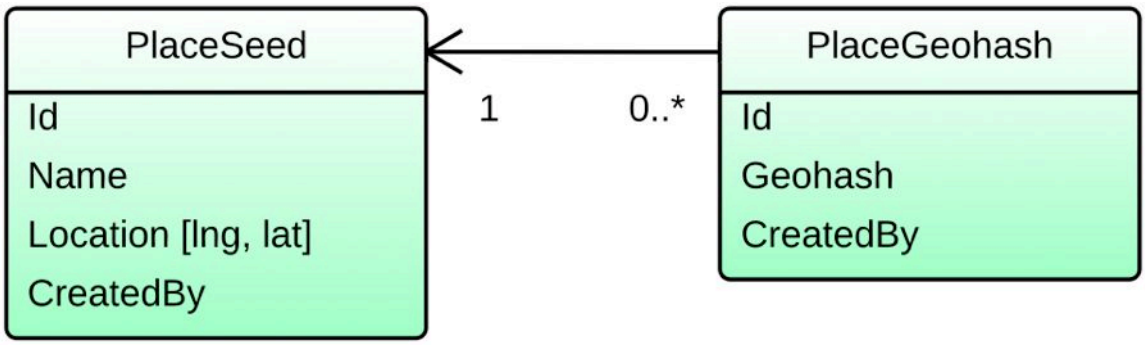

Figure 3.30 PlaceGeohash relation to a PlaceSeed

As mentioned, it includes the reference to a PlaceSeed and a geohash. It also includes the creator of this document, which represents the first user that suggested this geohash for the place. From the model, it can be seen that the relation allows a PlaceSeed to have 0 to many PlaceGeohashes associated to it. Until now, PizzaKing has two PlaceGeohash documents associated to it.

So basically, in order to have the area of a place, the query that is done in the geohashColl collection is one to return all documents that reference the PlaceSeed of interest. If there is an index for this reference, the query will be extremely efficient.

3.3.2 Evolving document

The goal of AREAS it to allow anyone to contribute to the area of any place. Naturally, this means that not all contributions will accurately determine the area. Because of this, the model

allows this determination of areas to be done in a democratic way. And this is done efficiently thanks to the flexibility in the schema offered by the MongoDB documents.

Let's show how the documents will evolve [3] when the crowd starts contributing to the area of the same place added at the previous section. Let's say a second user, called User2, does the actions shown in Figures 3.31 and 3.32.

Figure 3.31 One of the actions of User2 creates a new PlaceGeohash document

One of the actions will add a new PlaceGeohash document, similar to the g1 and g2 shown in Figure 3.29. But the other two actions will not create new documents. These will modify the existing g1 and g2 PlaceGeohash documents that already existed, as shown in the next figure.

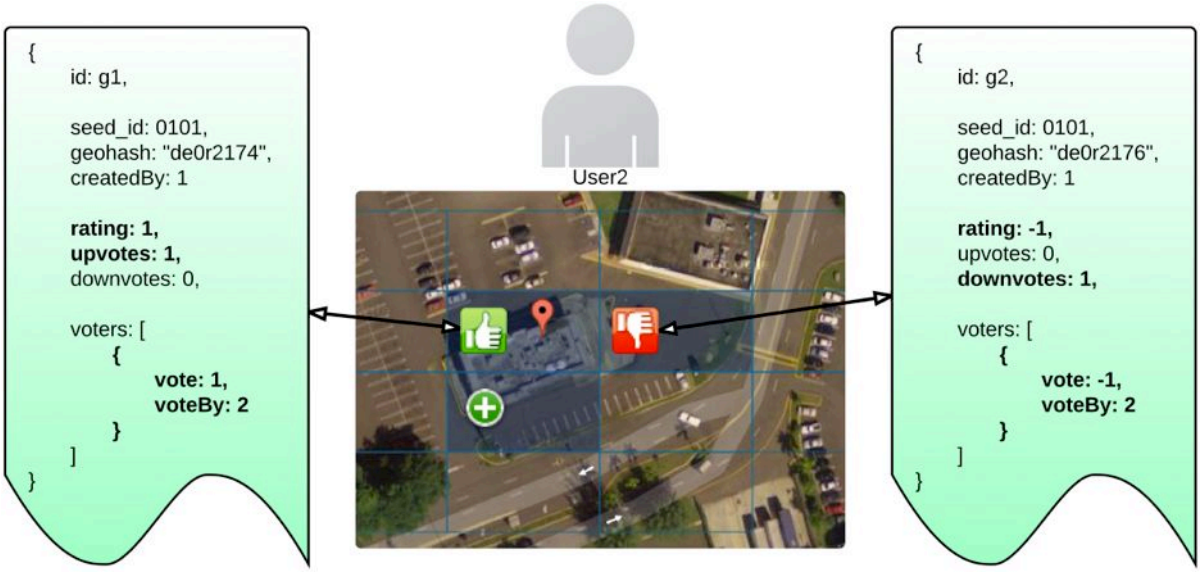

Figure 3.32 The other two actions modified existing PlaceGeohash documents

As shown in Figure 3.32, more attributes were added to g1 and g2, but the main one is an array of Voter documents. These "voters" represent if a user supported or not a geohash for an area. User2 supported g1 and didn't support g2, thus the 1 and -1 votes respectively. If more users interact with these geohashes, they will also be added to the same "voters" array. This makes another relation between the Voter and the PlaceGeohash entities like shown in Figure 3.33.

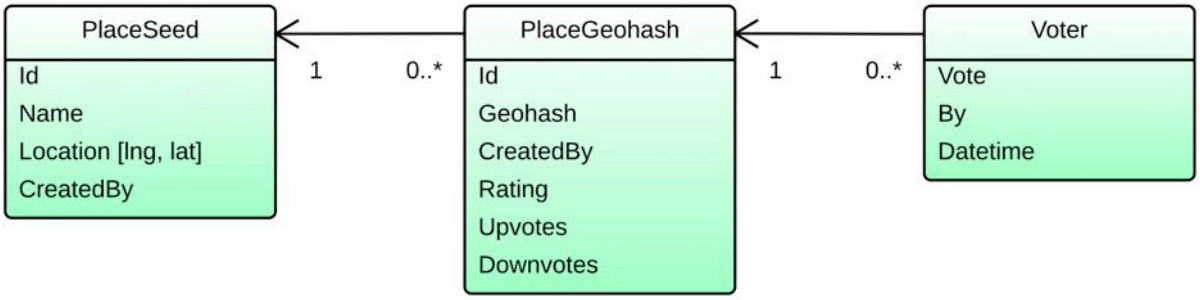

Figure 3.33 Relation between a Voter entity and the "evolved" PlaceGeohash entity (which relates to a PlaceSeed)

Aside from the voters array, the PlaceGeohash entity now has more attributes. These are counters for how many up-votes and down-votes are in the array. There is also a rating value to demonstrate that a rating could be calculated with each vote, in this case, is just the number of up-votes minus the number of down-votes. These counters make queries that want to consider these values much more efficient, not having to do any joins and calculating this number each time a query is done.

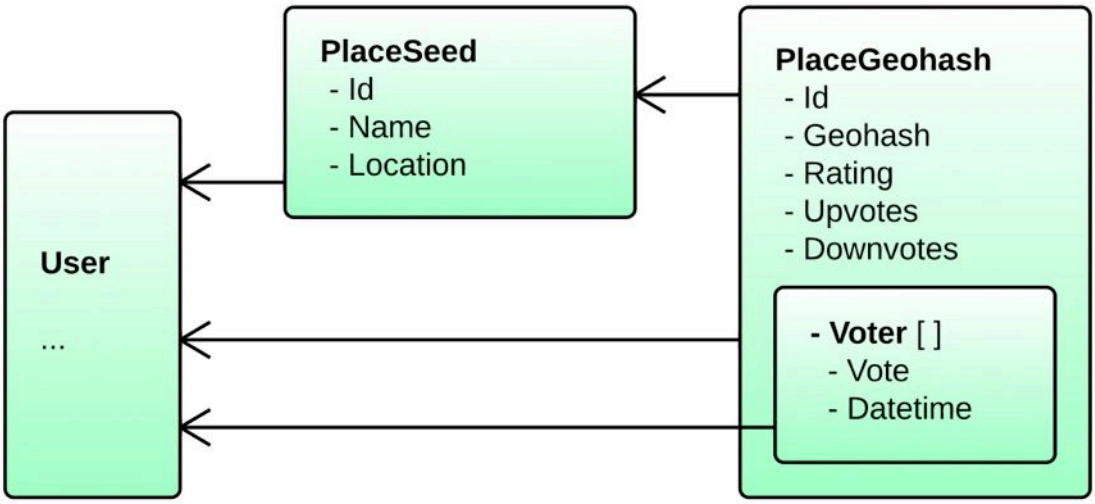

Figure 3.34 Data relations in MongoDB are done by embedding and references, not with expensive joins

Finally, Figure 3.34 shows how the relations are done in MongoDB documents. Showing that embedding the Voter documents inside the PlaceGeohash documents makes the crowd curation more efficient. It clearly represents the concept of "denormalization for performance", because this model already breaks the most basic level of normalization, 1NF, because these are non-atomic attributes.

4 CONCLUSIONS

AREAS demonstrates that it is possible to have the geographical area of places defined by the crowd. This can be useful for many things, like the improvement of the geo-awareness of mobile devices, to the evaluation and management of natural resources.

The geohash concept was studied, which defines locations in the world with areas. With the help of an open source application that visualized geohashes on a Google Map, an easy to use interface was created that allows the choosing of which geohashes define the area of a place. With this, users can start getting more used to the geohash concept, which makes it easier to manage locations when compared to latitude and longitude coordinates.

But having geographical areas represented by geohashes no only simplifies the usability for this type of application, it is also a model that is ideal for managing its data efficiently.

These areas are curated by keeping a rating that indicates if a geohash is appropriate or not to be included as part of the area of a place. By having data de-normalized, this rating can be calculated without having to do any joins or referencing other documents, all the necessary information is inside the same document. This is possible thanks to the capacity in MongoDB for a document to have flexible schemas, non-atomic values and allowing embedding other documents.

Using a RESTful interface, facilitated by the Restlet framework, makes the web service compatible with many devices. This makes possible many contributions, which have no negative effect on performance in terms of querying or modifying the area of one place. But

when a platform like this is used at a global scale, modifications do have to be made in the architecture.

For this reason, AREAS is being deployed in Amazon EC2. Amazon's cloud based Infrastructure as a Service (IaaS) will allow efficient horizontal scaling when the time comes.

This first version of AREAS served as a detailed proof of concept and of technology. The next step is to understand how users interact with it and make the necessary adjustments to the models and web service.

4.1 Future work

When dealing with document-oriented databases, the way data is being used is very important to determine the most efficient way to model it. MongoDB helps greatly in setting up a horizontally scalable database, but it has to be done responsibly. The database will be automatically partitioned based on a chosen shard key. If this value is not ideal, any benefit that could have had the automatic sharding will be wasted. So determining the best shard key must be tested, and this is a project by itself. [3]

Then, when the model is finalized, the web service API can also be finalized. This allows developers to do any kind of app that benefits from areas over points. From simple applications like the concept shown in this report, in which photos are automatically associated to places. But also, complex applications that use Big Data generated from other sources. Like being able to combine location-based environmental, social, and economics data for a real-time holistic sustainability report.

REFERENCES

[1] 10gen, Inc. (2012). *MongoDB*. Retrieved 2012 Oct from: http://www.mongodb.org/

[2] AWS. (2012). *Amazon Elastic Compute Cloud (Amazon EC2)*. Retrieved 2012 Oct from Amazon Web Services: http://aws.amazon.com/ec2/

[3] Banker, K. (2012). *MongoDB in Action*. New York: Manning Publications Co.

[4] DeLara, E., LaMarca, A., & Satyanarayanan, M. (2008). *Location Systems: An Introduction to the Technology Behind Location Awareness*. Morgan & Claypool Publishers. p. 88.

[5] Ewing, B., Reed, A., Galli, A., Kitzes, J., & Wackernagel, M. (2010). *Calculation Methodology for the National Footprint Accounts, 2010 Edition*. Oakland: Global Footprint Network.

[6] Fielding, R. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine, 2000.

[7] Foursquare Developers. (2012, Oct). *Venues Platform*. Retrieved 2012 Oct from Venues Platform: https://developer.foursquare.com/overview/venues

[8] GeoNames. (2012). *GeoNames Data*. Retrieved 2012 Oct from GeoNames webservice and data download: http://www.geonames.org/export/

[9] Glover, A. (2008, 22-Jul). *Build a RESTful Web service*. Retrieved 2012 Oct from IBM developerWorks: http://www.ibm.com/developerworks/java/tutorials/j-rest/section2.html

[10] Google Developers. (2012, Oct). *Getting Started*. Retrieved 2012 Oct from Google Places API (Experimental): https://developers.google.com/places/documentation/

[11] Louvel, J. (2012). *Introduction*. Retrieved 2012 Oct from Restlet: http://www.restlet.org/about/introduction

[12] Louvel, J., Templier, T., & Boileau, T. (2012). *Restlet in Action*. New York: Manning Publications Co.

[13] Monaco, A. (2012, 7-Jun). *A View Inside the Cloud*. Retrieved 2012 Oct from IEEE – The Institute: http://theinstitute.ieee.org/technology-focus/technology-topic/a-view-inside-the-cloud

[14] Niemeyer, G. (2008, Feb). *Geohash*. Retrieved 2012 Oct from geohash.org: http://geohash.org/

[15] OpenPlans. (2012). *OpenTripPlanner*. Retrieved 2012 Oct from openplans/OpenTripPlanner · GitHub: https://github.com/openplans/OpenTripPlanner

[16] OpenStreetMap. (2012). *Databases and data access APIs*. Retrieved 2012 Oct from OpenStreetMap Wiki: http://wiki.openstreetmap.org/wiki/Databases_and_data_access_APIs

[17] O'Reilly Answers. (2011, 20-Apr). *How to Gather Geographic Location Data*. Retrieved 2012 Oct from How to Gather Geographic Location Data: http://answers.oreilly.com/topic/2644-how-to-gather-geographic-location-data/

[18] solid IT. (2012). *popularity ranking of database management systems*. Retrieved 2012 Oct from DB-Engines Ranking: http://db-engines.com/en/ranking

[19] Troy, D. (2008). *Geohash Javascript Demonstration*. Retrieved 2012 Oct from davetroy/geohash-js · GitHub: https://github.com/davetroy/geohash-js

[20] Whelan, P. (2011, 15-Dec). *Geohash Intro*. Retrieved 2012 Oct from Big Fast Blog: http://www.bigfastblog.com/geohash-intro