Facultat d'Informàtica de Barcelona

UNIVERSITAT POLITÈCNICA DE CATALUNYA

# Noise Data Visualization and Identification Project

Lino Valdivia Jr

*Advised by*

Xavier Franch

MIRI Master Thesis

Barcelona 2015

## Contents

Introduction ........ 1
Smart Cities and the Internet of Things ........ 1
Smart City Initiatives Worldwide ........ 2
Santander ........ 2
Chicago ........ 2
Medellín ........ 3
Hong Kong ........ 3
Barcelona ........ 4
Noise and Urban Noise Pollution ........ 4
Noise Pollution in Barcelona ........ 6
The Noise Data Visualization and Identification Project ........ 7
Inspiration for the Project ........ 7
Goals ........ 8
Requirements ........ 8
Project Development Process ........ 9
Project Planning ........ 11
Architecture ........ 13
Data Model Design ........ 15
Project Data Query Use Cases ........ 15
NoSQL ........ 15
NoSQL and Denormalization ........ 16
Google App Engine and GAE Datastore ........ 16
BigTable Entity Design ........ 17
SensorLatestData ........ 18
SensorAggData ........ 18
SensorAggDOWData ........ 21
SensorRawAudioData ........ 22
SensorAlert ........ 22
Region ........ 23

Browser-Based Visualization ....24
Google Maps for Developers ....24
Google Maps and OpenStreetMap ....25
JavaScript Controls using jQuery UI ....27
Putting it all together ....28
Supported Browsers ....29
Noise Identification ....30
WAV Audio Format ....30
Preprocessing ....31
Feature Extraction ....31
Training and Classification ....32
The Modular Audio Recognition Framework (MARF) ....33
Training and test data samples ....33
Results of noise identification using MARF ....34
Integrating noise identification into the application ....35
REST Endpoints ....37
Test Cases ....38
Self-assessment of the Project ....40
Lessons Learned ....41
Future Work ....41
Acknowledgements ....43
Bibliography ....43

# Introduction

## Smart Cities and the Internet of Things

Although a comprehensive definition of "*Smart City*" agreed upon by academics and institutions has not been established, it generally refers to the employment of internet and communication technologies (ICT) in urban management and planning [1]. The primary goals of creating a Smart City are to better utilize public resources, improve services available to citizens, and lower the operational costs of managing the city [2].

The Smart City concept spans many different types of projects and initiatives across different cultures and cities around the world. Some of the more common Smart City projects are:

- **Digital Inclusion** – Making the Internet accessible and affordable by providing free Wi-Fi or computing resources, thus bridging the so-called "digital divide" between citizens who have ICT access and know-how and those who do not;
- **Data sharing and Open Data** – Making public data in the form of an online API or a file download (e.g. XML, CSV, spreadsheet, etc) available to increase transparency as well as provide a data source upon which developers can build new apps and services;
- **Green Policies for Sustainable Urban Development** – Reducing $CO_2$ emissions and addressing other environmental and safety concerns through ICT;
- **e-Government** -- Providing online access to government services;
- **City-wide Focus on Technology** – Improving the ICT infrastructure to encourage innovation, attract foreign investment and enable the transition to a knowledge-based economy [1].

A concept related to Smart Cities is the *Internet of Things* (IoT): the network of real-world objects embedded with circuitry, software, and connectivity. These so-called *smart objects* can sense factors in their environment (e.g. their GPS coordinates, air temperature, absence or presence of light, etc) and report the information they gather [3]. Smart Cities can take advantage of the technologies of IoT to monitor and improve the urban experience. Some of the possible IoT applications in Smart Cities settings include Smart Parking (tracking the availability of parking spaces), Smart Waste Management (monitoring trash container levels to optimize garbage collection routes), and Urban Noise Maps [4]. Data gathered from various sources such as sensors deployed throughout the city may be made available to citizens, thus increasing local government transparency and promoting awareness among citizens of the state of the city [2]. Smart City initiatives may even encourage citizen participation in government, in the form of facilitating active reporting of problems or concerns (e.g. a garbage container or a light post needs attention), or creation of new services or applications utilizing the data and/or tools provided by the city.

## Smart City Initiatives Worldwide

The following highlights some of the Smart City initiatives across the globe.

### Santander

SmartSantander is a project launched in 2010 in Santander, Cantabria province, Spain, in cooperation with several other institutions within the EU. It aims to be part of a wider network comprised of 20,000 sensors deployed in several other European cities (Belgrade (Serbia), Lübeck (Germany), and Guildford (UK)). The goal is to provide a test bed for Smart City experiments on a citywide scale, thus gaining true field results while providing new services to citizens [5].

### Chicago

Chicago is the third largest city in the US, with a population of 2.7 million people. Its Smart City objectives solidified in 2011 and aim to improve local government transparency, uncover new insight by performing analytics on city data, and spur economic development. The Smart Chicago Collaborative, a partnership between the city, the Chicago Community Trust, and the MacArthur Foundation (one of America's largest philanthropic foundations) was created with the express aim of "closing the digital divide" and using technology to improve citizens lives [6].

- **Sustainable Broadband Adoption** – By providing computers and training sessions to more than 11,000 residents and 500 small businesses/non-profits in five disadvantaged neighborhoods of Chicago, the city hopes to encourage economic development by increasing awareness and adoption of ICT.
- **Connect Chicago** – This loose network of 250 places (libraries, colleges, community centers, etc) throughout the city provides internet and computer access, digital skills training, and online educational resources for free.
- **Chicago Health Atlas** – The city created a website to visualize health-related information and other statistics on a city map. Users can browse the various districts of the city and view city statistics from various datasets such as chronic disease (e.g. asthma, hypertension) or crime (e.g. aggravated assault, simple battery) [7].

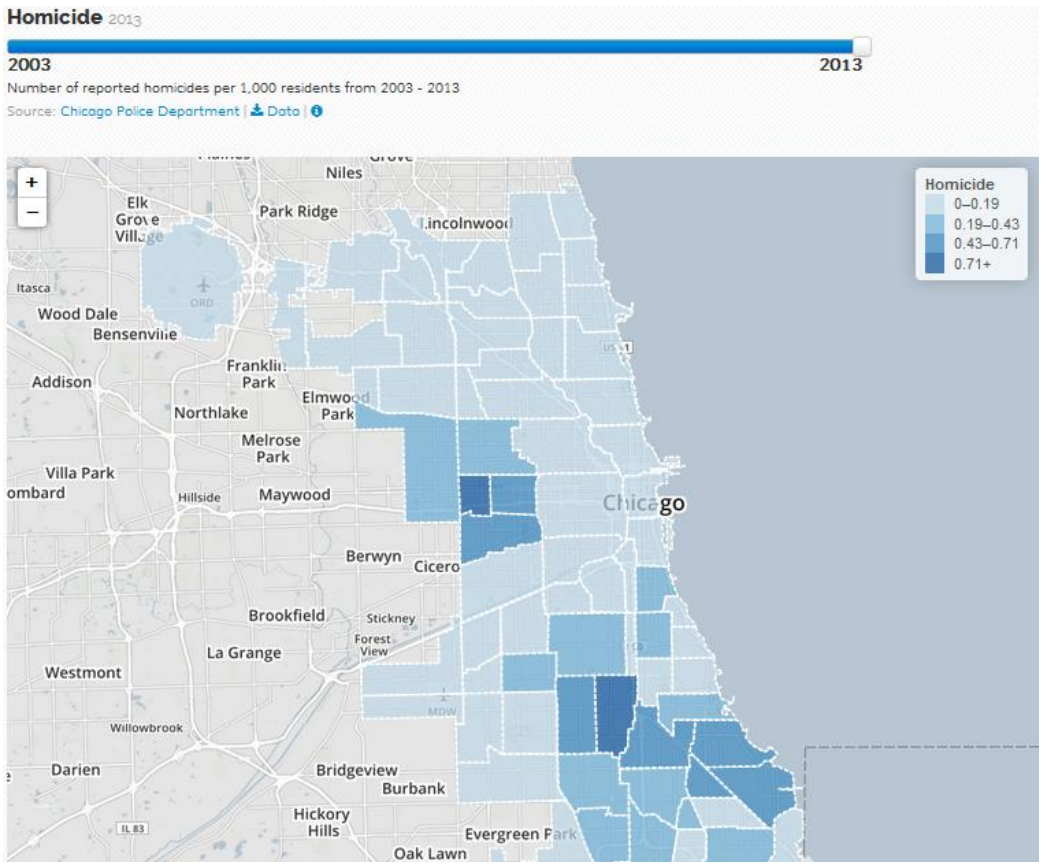


Figure 1: Visualization of homicide data in various Chicago neighborhoods (from chicagohealthatlas.org)

## Medellín

Colombia's second largest city is rising from the ashes of the 1980's and 1990's marred by drug trade-related violence (TIME Magazine once called it the most violent city in the world) with a renewed focus to be the leading center of technology in South America.

- **Medellín Ciudad Inteligente** – This city program has been promoting the use of ICT in the Medellín since 2007 and counts free public Wi-Fi, open data, and access to computer and learning resources as some of its projects [8].
- **Medllinnovation District** – An urban transformation project that aims to convert north Medellín into a technological park for entrepreneurs, institutions, and companies and foment a knowledge economy [9].

## Hong Kong

Hong Kong is a major economic powerhouse in Southeast Asia and one of the leading financial centers of the world. It's Smart City initiatives began nearly 20 years ago in 1998, when its Digital 21 Strategy was first proposed. Its current goals revolve around e-Government services, digital inclusion, and promoting Hong Kong as a world-class ICT hub

[6]. However there doesn’t seem to be a strong focus on environmental sustainability, which is considerable problem for one of the most densely populated cities in the world (6,650 people per square kilometer in 2013 [10]) and one that chronically suffers transport and housing issues as well as air, solid waste, and water pollution.

### Barcelona

This compact Mediterranean city aims to be the reference model in Smart Cities, investing in the creation of technology hub 22@Barcelona as well as actively hosting and participating in industry conferences such as the Smart City Expo and Internet of Things World Forum [11].

- **Barcelona Wi-Fi --** The city provides free Wi-Fi access at more than 400 access points throughout the city, making it the largest public free Wi-Fi network in the country.
- **Orthogonal Bus Network** -- New horizontal, vertical, and diagonal bus routes were introduced to improve urban mobility. Bus stops are equipped with panels displaying arrival information, while in-bus panels inform passengers of the next stop as well as available bus, train, and/or subway connections.
- **Irrigation Telemanagement** -- In March 2014 the city unveiled a remote management system for the automated watering system that controls the duration and frequency of irrigation in each area. Sensors on the ground record factors such as humidity, temperature, and sunlight, which gardeners can use to adapt the watering schedule of plants and avoid overwatering. While not as glamorous sounding as other projects, this project has been estimated to have produced more than 550,000 euros in savings that summer, while at the same time lowering the city’s water consumption [12].

## Noise and Urban Noise Pollution

Noise is broadly defined as “unwanted sound” [13], and is omnipresent especially in city living. It is thus considered a form of environmental pollution, as much as carbon monoxide (CO) is for air [2]. In one study ranking the noisiest cities in the world, Madrid ranked sixth, ahead of more populous cities like Shanghai and New York [14].

Sound levels are typically measured using decibels (dB), which in acoustics is typically the logarithm of a ratio of sound pressure (i.e. sound pressure produced by a noise source vs. a reference pressure such as the limit of sensitivity of human hearing) [15]. By this definition, a value of 0dB means that the measured sound pressure from a source is equal to the reference pressure, which is at the lowest threshold of the human ear, i.e. barely susceptible sound. The following chart compares the decibel values from common everyday noise sources.

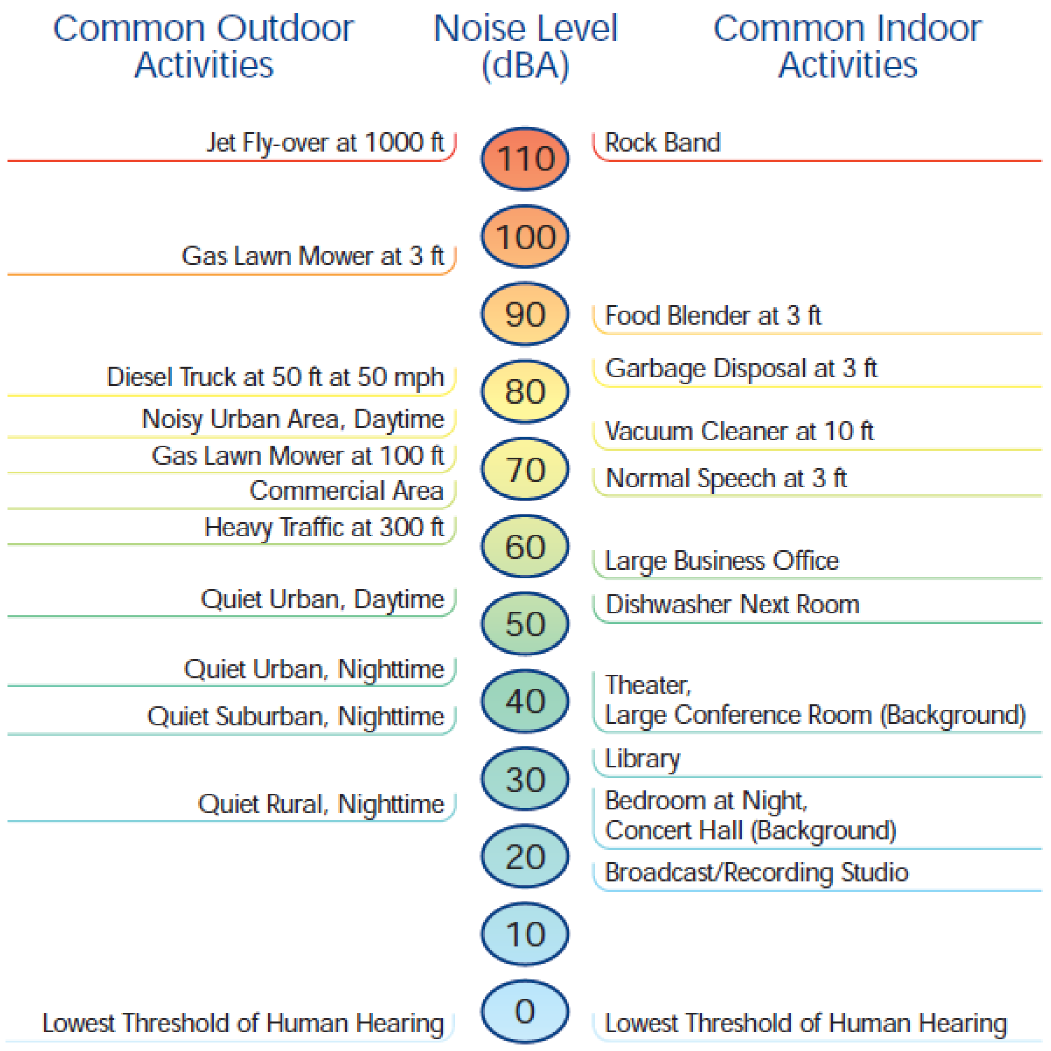


**Figure 2: Loudness Comparison Chart (Source: City of Redding, California/Shasta County/Caltrans)**

The World Health Organization considers 70dB as “uncomfortable” and 90dB as harmful [16]. In fact, sustained exposure to decibel levels around 90 - 95dB may result in hearing loss [17]. Examples of environmental sources that can produce these noise levels include a jackhammer heard from 50 feet away, or a subway train from 200 feet away. A rock concert can reach levels as high as 115dB.

Aside from auditory damage, prolonged exposure to high noise levels has also been linked to impaired performance, elevated heart rate and stress levels, and hypertension (particularly in industrial settings) [13].

## Noise Pollution in Barcelona

In Barcelona, according to the Barcelona Field Studies Centre, the effect of noise is further aggravated by high concentrations of people living together. Barcelona is Spain's second largest city, with nearly 28,500 inhabitants per square kilometer in its 52 $km^2$ of inhabited land (2008 data) [18]. Furthermore, over half of Barcelona's population is subjected to noise levels over 65dB during the entire day (from 8am to 10pm) [19]. Like most cities, the main sources of noise are traffic, large transport infrastructures, leisure activities, shopping districts and industrial activities. Although vehicle noise is the primary culprit, it has become more of a "background noise" to city living in Barcelona, and thus does not generate as much complaint as other types of noise. Noise generated in entertainment zones e.g. bars, restaurants, dance clubs, theatres, etc. are the second source of acoustic pollution in the city. Unlike traffic noise, however, this type of noise is more occasional and present in more concentrated parts of the city [20].

The City of Barcelona has outlined a 10-year plan to reduce noise pollution. The plan calls for a holistic approach involving citizen awareness as well as the use of technology to identify hotspots of noise in the city. This would drive the creation of policies and ordinances to combat acoustic contamination, and would also serve as a feedback mechanism to determine the effectiveness of these measures. Some of the action items mentioned in the report include

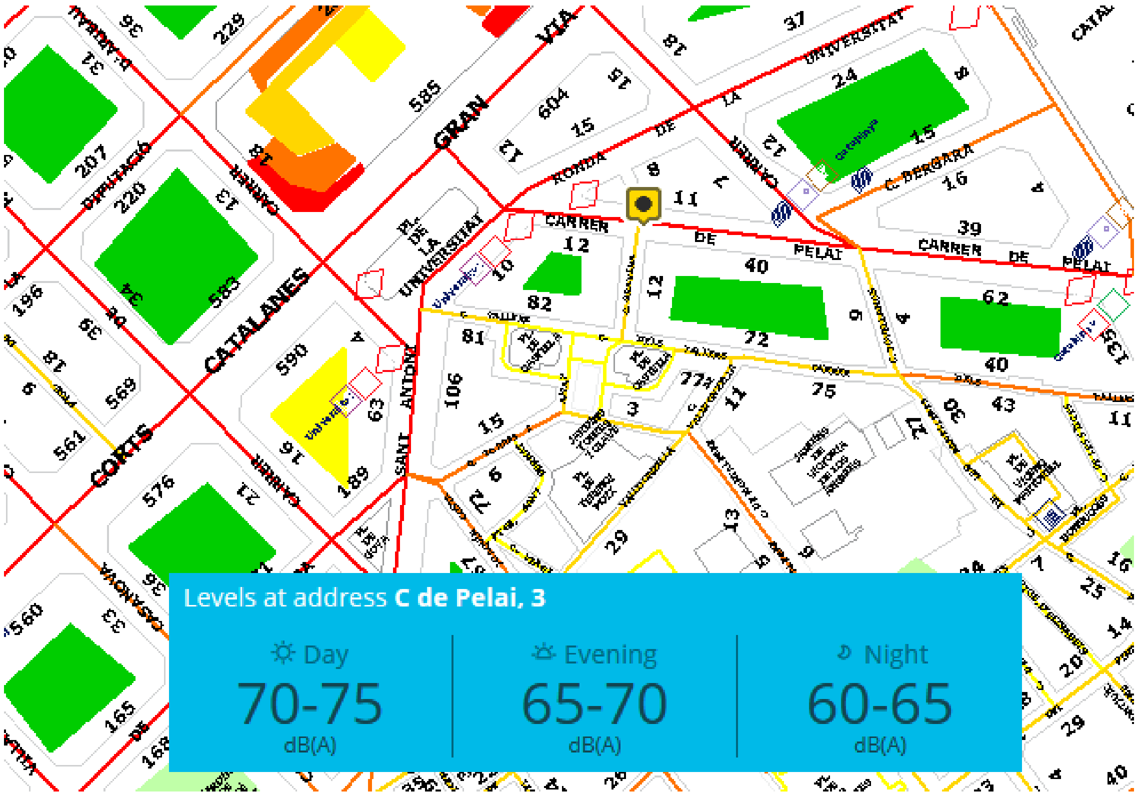
- Promoting the use of public transit and bicycles (since traffic is the leading cause of noise);
- Increasing pedestrian space and the use of noise-reducing pavements;
- Installing noise insulation screens along major thoroughfares such as Gran Via;
- Protecting quiet zones (e.g. parks, internal patios of buildings, etc.) and sensitive zones (e.g. schools, hospitals, senior citizens' residencies, etc) [21].

# The Noise Data Visualization and Identification Project

## Inspiration for the Project

Early on the plan was to work on a project that would leverage data generated by the city and produce a tool that may be of use to urban planners. One idea that came about was to create a visualization tool using the data from noise sensors in Barcelona. The visualization tool can be used by city planners to identify the areas that have the highest levels of acoustic contamination, and by normal users to find quiet zones within the city.

The City of Barcelona's Environmental Department does have an online Strategic Noise Map [21] showing the different levels of noise in various streets and districts. While it gives a general idea of the noise levels in different parts of the city, it does have some limitations. In particular, the data from this map is static data gathered from sensors and interpolated from other data (such as the amount of traffic on a particular roadway) in 2012 and 2009. The map shows the average noise levels for three predefined and fixed time ranges (daytime: 7am to 9pm, evening: 9pm to 11pm, and nighttime: 11pm to 7am) but it does not allow other ways of querying the data, such as by day of the week or a date range, or a custom time range.


Levels at address **C de Pelai, 3**

| Day | Evening | Night |
|---|---|---|
| 70-75 dB(A) | 65-70 dB(A) | 60-65 dB(A) |

Figure 3: Screenshot of the Strategic Noise Map showing Placa Universitat

It seems then that a more sophisticated visualization tool would be of further use. The data from the sensors can be gathered periodically and stored in a NoSQL database that can later be queried for historical, aggregated data. NoSQL was chosen as a way to further explore the paradigm as well as its claims of being a highly scalable solution for handling large amounts of time-series data [22].

Later while researching Smart City projects we came across the Zanella et al paper [2] which also mentioned the use of sensors to build a space-time map of noise pollution in the city. The authors went one step further and proposed a system that can identify the type of noise (e.g. car noise vs. glass breaking vs. humans talking) by means of sound classification algorithms. This might prove helpful in determining whether the authorities should be alerted of possible criminal activity or incivility in a particular area.

Interestingly there was also a similar noise identification pilot project in the 22@ Tech District of Barcelona, described in a blog post as "noise sensors detection (uncivil activities, car alarm…) followed by service processing and activation of the closest camera streams. Control room checks the image and decides if police must be alerted while a live video feed is routed to the nearest police car." [23]

It was then decided that noise identification be incorporated into the project. It would not only be an interesting value-add to the visualization tool but would also be a practical application of machine learning concepts.

## Goals

- Develop a system that makes use of available sensor data as part of the Smart Cities initiative of Barcelona.
- Design an architecture that is modular and extensible enough to allow for components to be swapped (e.g. other types of sensor data or classifiers).
- Apply concepts and technologies (web services, data warehousing for Big Data with NoSQL, machine learning for classification) that have been learned during the MIRI program.
- Contribute to the growing body of research in Smart City/IoT applications.

## Requirements

- The system should be able to accept data from more than one possible source (a *provider*). A provider represents a set of sensors that emit readings and whose data is gathered, analyzed, and visualized independently of other providers.

- The data store should be able to store large amounts (up to 6,000 sensor readings per minute per provider) of time series data.
- The query handler should be able to respond to the following types of data queries:
    - Latest readings of sensors from a provider
    - Geometric average of a provider's sensor readings for a day of the week (e.g. Monday) and time range (e.g. 6am to 2pm)
    - Geometric average of a provider's sensor readings for a date range
- The visualization module should allow the user to query for the data in the three ways described above, and then view the visual representation of the results on a map.
- The visualization module should be supported in at least the three major browsers (Firefox, Internet Explorer, Chrome).
- The noise identification module should be able to accept sound data in the 16-bit WAV format.
- The noise identification module should have a success rate (i.e. rate of correct identification) of at least 70%.
- When sound data has been identified as a "noise of interest" and an alert has been generated, the visualization module should display this alert within 1 minute of identification.

## Project Development Process

After reading several papers on Smart City applications, the project idea was further refined into a document outlining the vision, goals, risks, and preliminary work estimates. One of the possible risks identified early on was the possibility that the sensor data may not be available. It seemed from the initial research that the sensor data APIs were there under several layers of links, but in the event that we are not able to access them simulating the data was put forward as an option.

The development and collaboration infrastructure was built next, with the setup of an online project management tool (see next section), a shared Dropbox folder, and a Bitbucket git repository.

In order to look into the possibility of creating a mobile version of the visualization tool, we met with Jordi Marco for a crash course on Android development. While it was definitely interesting and useful, in the end the plan to create a visualization app for mobile devices was scrapped in order to focus on the browser-based visualization.

After several iterations a working version of the web application was successfully deployed on Google App Engine. The application utilized simulated data stored in the Google App

Engine Datastore, which is based on BigTable and follows a key-value, column family model.

Later a meeting was held with Julia Camps of the Environment Department of the City of Barcelona to talk about the project, understand more about the city's plans on sensor deployments, and get feedback on the application thus far. The takeaways from that meeting were the following:

- Only a few sensors have been deployed in the city, and while data is being gathered it is not currently available to the general public.
- Sensor data was being gathered every minute. That means that for every hour, if there are a 100 sensors deployed there could be up to 6,000 sensor readings to be analyzed.
- The computed average of decibel readings should be a geometric mean, not an arithmetic mean.
- They also recommended the use of a standardized color scheme for decibel ranges and predefined time periods (Day, Evening, and Night), as seen in the Strategic Noise Map website [20].

Their feedback was incorporated into the application, while satellite imaging and heatmaps were also added to the final visualization module. These changes were met with positive reactions from the department representatives.

As for noise identification, we contacted a hardware engineer from the Smart Citizen Kit project regarding the feasibility of capturing sound data from a sensor. He said that this is possible on a sensor board with a normal microphone and good sound card, and then data can be sent to another location for processing.

While looking for Java-based implementations of sound identification algorithms we came across MARF. Three classes of possible street noise were defined: person screaming, person laughing, and a motorcycle revving. Sound samples were retrieved from freesound.org. We tested these against several possible MARF configuration parameters, and achieved a 73.68% rate of correct identification based on 22 training recordings and 19 test recordings.

The noise identification module was then integrated into the application in the form of a REST endpoint and an alert visualization. However there was a slight hiccup when we attempted to deploy to Google App Engine, as some classes (particularly those from the javax.sound.sample.* package) that were used to read and parse the contents of WAV files were not on the whitelist. This then required rewriting of The WAV file loading class so that it wouldn't be dependent on the unapproved classes.

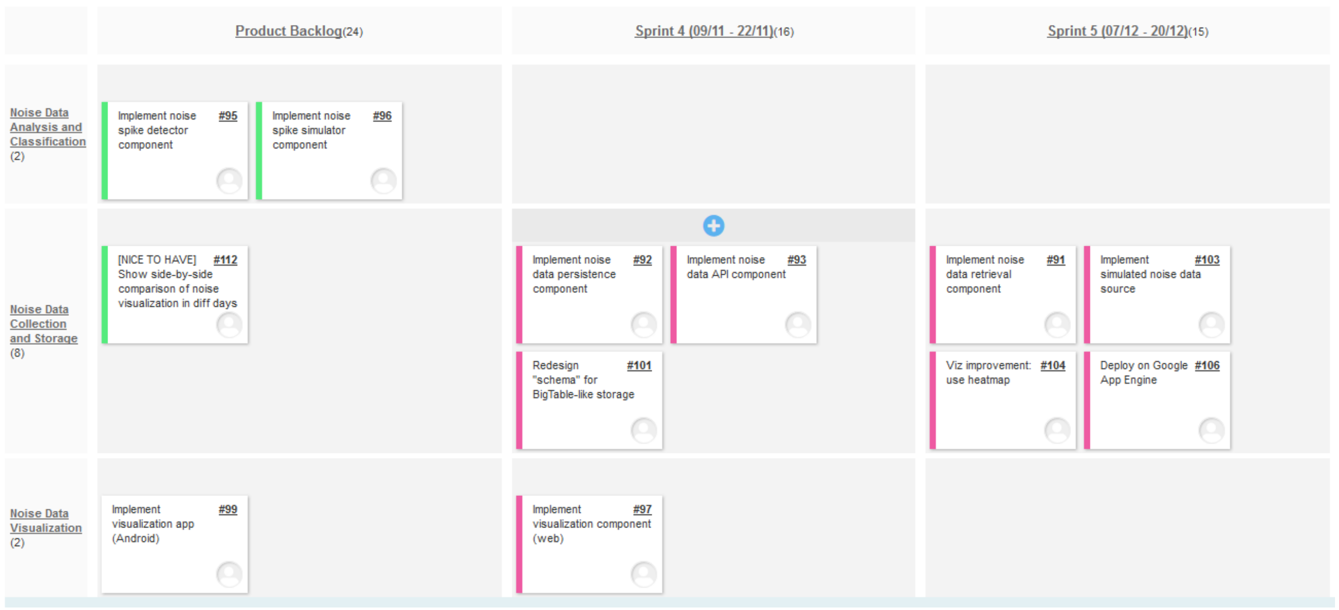
The remaining weeks were spent finishing documentation, cleaning up the data model design and the code, and polishing the UI module for the demo included in the final presentation.

## Project Planning

For this project a "light" Scrum approach was adopted. Scrum is a popular process framework used for managing complex projects. It employs an iterative and incremental approach to manage risk and increase the predictability of product development [24]. Scrum is typically used in larger projects with multiple stakeholders and several roles, but it was also suitable even for a one-man development team to manage tasks and keep development on track.

In Scrum, development is performed in set spans of time called sprints, at the end of which a "demoable" product is output. For this project, each sprint lasts two weeks. Tasks that need to be done in order to complete the project are defined in a product backlog as stories, which are then assigned to sprints. As each task is completed, it's story status is set to "Done". Unfinished tasks at the end of a sprint are reevaluated and either moved to another sprint or discarded.

An online tool called Mingle [25], which is free to use for 5 users or less, was employed in the project. Mingle is a project management tool that allows users to define and follow the progress of a project at a glance, and easily drag and drop tasks within sprints and the product backlog.



| | Product Backlog(24) | Sprint 4 (09/11 - 22/11)(16) | Sprint 5 (07/12 - 20/12)(15) |
|---|---|---|---|
| Noise Data Analysis and Classification (2) | Implement noise spike detector component #95; Implement noise spike simulator component #96 | | |
| Noise Data Collection and Storage (8) | [NICE TO HAVE] Show side-by-side comparison of noise visualization in diff days #112 | Implement noise data persistence component #92; Implement noise data API component #93; Redesign "schema" for BigTable-like storage #101 | Implement noise data retrieval component #91; Implement simulated noise data source #103; Viz improvement: use heatmap #104; Deploy on Google App Engine #106 |
| Noise Data Visualization (2) | Implement visualization app (Android) #99 | Implement visualization component (web) #97 | |

Figure 4: Mingle Project Management Tool (screenshot from Thoughtworks.com)

A task is represented by a Story card that belongs in one of three application features:

- **Noise Data Collection and Storage** -- includes tasks such as defining the data model and understanding the API to retrieve noise data from sources.
- **Noise Data Analysis and Classification** -- includes tasks such as researching and developing the noise identification module.
- **Noise Data Visualization** -- includes tasks such as understanding the JavaScript mapping library and adding features such as a heatmap.

The sprints defined for this project as well as some of the tasks completed in them are:

| Sprint | Tasks completed (partial list) |
|---|---|
| Sprint 1 | Identify and understand the noise data fields, create a repository for version control, find out if an Android client is feasible |
| Sprint 2 | Design system architecture, design data model |
| Sprint 3 | Implement noise data persistence, implement noise data retrieval REST API |
| Sprint 4 | Implement browser-based visualization module |
| Sprint 5 | Implement simulated noise data source, deploy webapp to remote Google App Engine |
| Sprint 6 | Add heatmap visualization improvement, research noise identification algorithm |
| Sprint 7 | Look for noise data training and test set, perform noise identification using the MARF library |
| Sprint 8 | Write Android app that simulates noise sensor for capturing audio |
| Sprint 9 | Integrate noise identifier module into the rest of the application |
| Sprint 10 | Complete project documentation and presentation slides |

**Table 1: Sprints and tasks completed**

Overall the project took 20 man-weeks of development to complete.

## Architecture

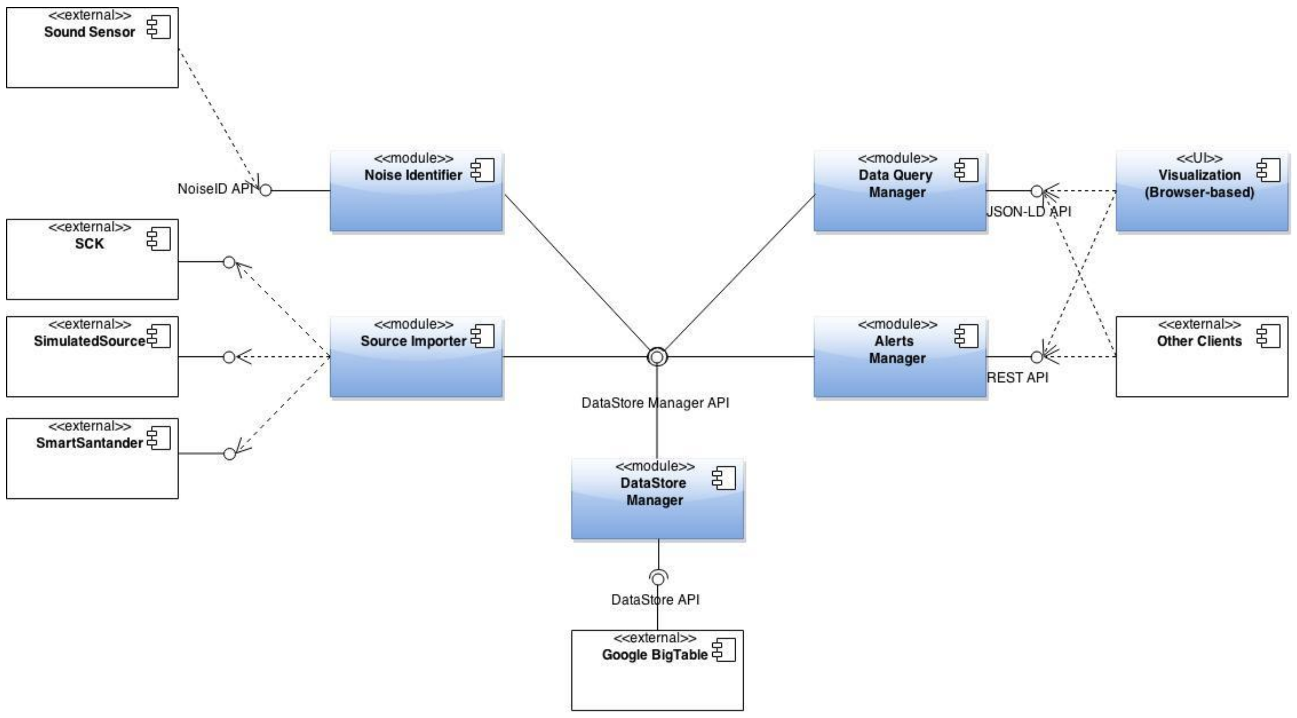

<<external>> Sound Sensor
NoiseID API
<<module>> Noise Identifier
<<module>> Data Query Manager
JSON-LD API
<<UI>> Visualization (Browser-based)
<<external>> SCK
<<external>> SimulatedSource
<<module>> Source Importer
<<module>> Alerts Manager
REST API
<<external>> Other Clients
DataStore Manager API
<<external>> SmartSantander
<<module>> DataStore Manager
DataStore API
<<external>> Google BigTable

**Figure 5: System Component Diagram**

The modules in blue show the components that are part of the project:

- **Source Importer** -- comprised of classes that read data from various sources of noise data. These classes periodically issue requests to data sources (such as the Smart Citizen Kit (SCK) Initiative) and converts the results of these requests into a representation that can be stored for later analysis.
- **Data Store Manager** -- manages access to the underlying data store. All other modules must not access the data store directly, but instead use the API exposed by the Data Store Manager. This allows the architecture to adapt to a different data store (switching from BigTable to MongoDB, for example) without affecting the rest of the components.
- **Data Query Manager** -- handles requests for noise data by external clients, converting REST-style requests into internal calls to the Data Manager, and transforming these results into a JSON response.
- **Alerts Manager** -- handles requests for managing alerts by external clients. An alert is generated by the Noise Recognizer module when it identifies a particular noise of interest, such as a person screaming.
- **Noise Identifier** -- receives raw audio data from a source, such as a sensor with a microphone, and attempts to classify it (e.g. person screaming, motorcycle starting,

etc). If the sound is recognized as something of interest to the system, an alert is generated.

- **Browser-based Visualization** -- the set of HTML, JavaScript, and CSS documents that render the visualization inside a browsers.

# Data Model Design

## Project Data Query Use Cases

Ultimately the choice of data store solution should depend heavily on how the data will be utilized. The most important piece of data gathered would be the timestamped decibel (dB) readings from fixed sensors. The data will then be retrieved periodically (e.g. every 5 minutes) from a data source provider and stored for later analysis. Some preprocessing may be done on the raw data prior to storage to facilitate later computations.

The data store must be able to support the following query use cases:

- Return the latest data that has been gathered from sensors of a particular provider.
- Return the geometric mean of dB readings over a certain date range (e.g. from Dec 30 to Jan 5) from sensors of a particular provider.
- Return the geometric mean of dB readings for a certain for a time of day (e.g. from 7am to 9pm) on a given day of the week (e.g. Mondays, weekends, or workdays) from sensors of a particular provider.

## NoSQL

The term "NoSQL" encompasses the next generation of databases that seek to differentiate themselves from traditional (relational) databases and are often characterized by [22] [26]:

- Distributed architecture and horizontal scalability, typically on commodity servers instead of specialized hardware
- Schemaless nature
- Eventual consistency model, as opposed to the transactional (ACID) model found in traditional relational databases
- Ability to handle large amounts of data that may be structured, semi-structured, unstructured, or polymorphic

For this project traditional database solutions such as MySQL or PostgreSQL can also be used, but a NoSQL data store is ideal because incoming sensor readings are similar to other popular NoSQL use cases such as log file entries or audit trails (tickets and receipts). That is, the data doesn't change once it is created, and the data is used in an "append-only" setting. Furthermore sensor readings are gathered continuously at periodic intervals. Such type of data is referred to as *time series* [9]. Because of the "append-only" nature of the storage of time series data the data store will grow as more data points come in. Thus they would need to be hosted in a solution that can easily scale horizontally.

## NoSQL and Denormalization

One of the more surprising aspects of working with NoSQL data stores, especially for those who come from a traditional SQL background, is embracing denormalization. Joins are considered inefficient when data is spread across multiple machines. Because most NoSQL solutions don't have native support for table joins, data values may have to be repeated in several locations. Utilizing disk space (generally considered as a cheap resource) in exchange for better scalability (in the form of faster reads from having data collocated instead of spanning multiple tables) is a key tradeoff in NoSQL applications [27]. Additionally, if the data is not expected to change (e.g. a UUID or a timestamp) then data inconsistency is not an issue because the problem of having to update multiple redundant entries does not exist.

## Google App Engine and GAE Datastore

Google App Engine (GAE) is Google's the Platform-as-a-Service (PaaS) offering. Applications can be hosted on Google's servers and automatically scales based on load. It supports applications written in several languages, such as Java and Python. Each application runs in a secure, sandboxed environment, unaware of the presence of other applications, the underlying operating system, or the physical location of the hosting server. The platform also offers a host of other support features, such as OAuth, Memcache, and XMPP [28].

The Datastore available on GAE is based on BigTable, a distributed storage system designed to handle large amounts of data (in petabytes) across thousands of servers. BigTable provides a simple data model based on a key/value store, and is used by Google for several of its products such as Google Earth and Google Finance [29].

The GAE Datastore builds on top of BigTable and adds indexes, replication, high availability as well as the possibility of having the data hosted on multiple-data centers [30]. There is also a MapReduce framework which may be taken advantage of in the future should more complex and processing-intensive computations on the data are needed.

Applications hosted on GAE are free to use whatever data storage solution they desire, but since the Datastore is already fairly mature and available for free on the platform, it made sense to take advantage of this technology.

Google does impose limits on free applications it hosts. For example, code and static data cannot exceed 1GB. The total amount of data stored that can be stored is also limited to

1GB for free applications. These limits can be increased by enabling billing on the application [31], however for the purpose of demonstrating the project this is not an issue.

## BigTable Entity Design

In BigTable, data is grouped into objects called *entities*, with each entity having one or more *properties*. Each entity belongs to a particular *kind*, and entities of the same kind have a unique *key*.

So far, all these concepts appear to be equivalent to their traditional SQL counterparts (tables, columns, primary keys, etc). However one key difference is that entities are *schemaless*, i.e. entities of the same kind need not have the same properties. For example, one entity of kind Employee might have a property **mobilePhoneNumber**, while another entity of the same kind might lack this property and instead have one called **homePhoneNumber**. The datastore will not enforce that all entities of the same kind have the same properties [32].

The following sections describe the entities used in the project. Following the earlier discussion on the need for denormalization when modeling data in NoSQL applications, some data values (e.g. latitude and longitude) may be duplicated in some entities.

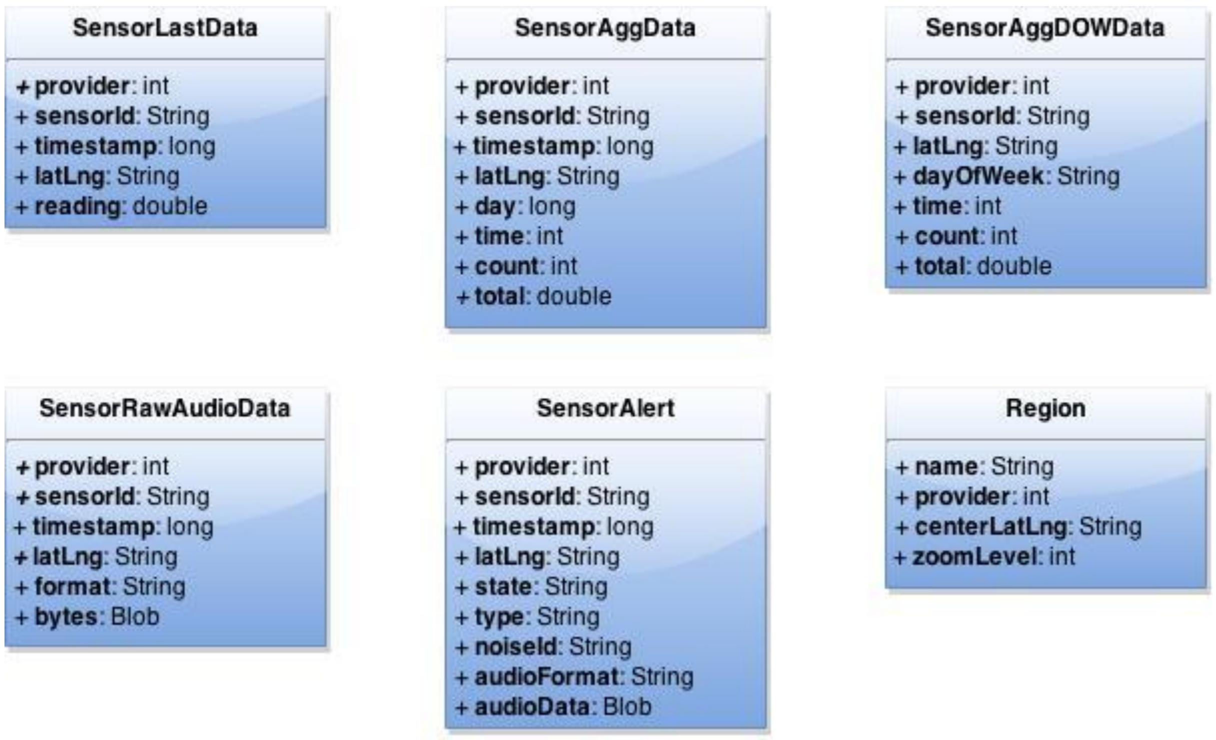


Figure 6: Datastore Entities

### SensorLatestData

Stores the latest data obtained from a source identified by the provider field. Unlike the other entities in the system this set of entities doesn't grow since it only keeps the latest data (i.e. data associated with a particular sensor is always overwritten when a new set of readings is retrieved), unless a new set of sensors is added.

| Field Name | Type | Description |
|---|---|---|
| provider:sensorId | Datastore Key | Unique key for this entity. |
| provider | int | Identifies the data source provider. |
| sensorId | String | Unique identifier for this sensor. Provided by source. |
| timestamp | long | Time when this sensor reading was reported, in UTC milliseconds |
| latLng | String | Latitude and longitude of the reporting sensor, separated by a comma |
| reading | double | Raw sensor reading, in dB |

### SensorAggData

Stores sensor data aggregated by date and time (in hourly increments). As previously mentioned, entities in the data store are denormalized, and fields such as the latitude and longitude of a sensor are repeated across several entities.

The project requires the computation of the (geometric) average of sensor readings on a given date range. However, Google App Engine does not support aggregate operations (e.g. SUM, AVERAGE) on entity fields [33]. This is something that traditional SQL databases offered, but in NoSQL this was something that was given up in order to handle large volumes of data that may be distributed on several machines. One alternative is to use an offline Map/Reduce solution, but this is not suitable for handling ad-hoc queries; that is, we don't know the date and time range that a user would be interested until a query is actually received.

Reading from the GAE datastore is extremely fast and cheap [34], so another approach is to shard the data to be aggregated, and perform the aggregation on the application side on demand, i.e. when a request comes in.

From the application's stated use cases we already know that the data will be queried given a date and time range (e.g. give me the average noise on April 10, 2015 from 6pm to 8pm) and by day of the week and time range (e.g. give me the average noise readings on Wednesdays from 6am to 2pm). We can't precompute the averages for a particular range since we don't know what ranges the clients would ask for; computing the averages for all the possible date and time ranges is not impossible but is inelegant. Instead we keep a **total** and a **count** field associated to a particular date and hour.

Because we are taking the geometric average and not the arithmetic average, the **total** field is actually the running total of the logarithms of the sensor readings. Recall that the definition of a geometric average is the nth root of the product of n values [35]; that is:

$$\left(\prod_{i=1}^{n} x_i\right)^{1/n} = \sqrt[n]{x_1 x_2 \ldots x_n}$$

Because multiplication of small values can often lead to underflows in computers, a simple trick is to apply two properties of logarithms: the logarithm of a product is the sum of logarithms, and the logarithm of a value raised to a power is the product of the power and the logarithm of the value. Here we can apply the natural logarithm $ln$, which gives us:

$$ln\left(\prod_{i=1}^{n} x_i\right)^{1/n} = ln\left( (x_1 x_2 \ldots x_n)^{1/n}\right) = {}^{1}/_{n}\sum_{i=1}^{n} ln\, x_i$$

This computed value is not yet the geometric average, but rather the natural logarithm of the geometric average. We therefore need to apply the inverse of the natural logarithm, which is $e^x$, to obtain the desired value.

In our model the **total** field stores the sum of the logarithm of the readings (i.e. $\sum_{i=1}^{n} ln\, x_i$) within a given date and hour. To obtain the geometric average, this **total** is divided by **count** ($n$), then $e$ is raised by that value to obtain the geometric average.

The following table illustrates the approach taken to compute the average over a time range:

| Date | Hour | Total | Count |
|---|---|---|---|
| 04/10/2015 | 1800 | 78.24 | 20 |
| 04/10/2015 | 1900 | 88.74 | 20 |
| 04/10/2015 | 2000 | 81.6 | 20 |

**Table 2: Example table of logarithm totals and counts grouped by date and time**

In this example, the geometric average for 6pm (1800) is $e^{(78.24/20)} \approx 50$. The average for the two- hour range of 6pm until 8pm (exclusive) is $e^{(78.24 + 88.74/40)} \approx 65$, while the average for the whole table is $e^{(78.24 + 88.74 + 81.6/60)} \approx 63$.

Now when a new piece of sensor data comes in with a timestamp of April 10, 2015 8:15pm and a reading of 65 dB, the row for hour 2000 is updated to reflect the new total and count values:

| Date | Hour | Total | Count |
|---|---|---|---|
| 04/10/2015 | 1800 | 78.24 | 20 |
| 04/10/2015 | 1900 | 88.74 | 20 |
| 04/10/2015 | 2000 | 85.774 | 21 |

Table 3: Example table with updated logarithm total and count

The **total** field is updated to 81.6 + ln(65) = 85.774, while the **count** field is simply incremented. The new geometric average for the whole table is therefore now $e^{(78.24 + 88.74 + 85.774/61)} \approx 63$.

A similar approach is employed for queries based on the day of the week (see **SensorAggDOWData** below).

| Field Name | Type | Description |
|---|---|---|
| provider:sensorId:day:time | Datastore Key | Unique key for this entity. |
| provider | int | Identifies the data source provider. |
| sensorId | String | Unique identifier for this sensor. Provided by source. |
| latLng | String | Latitude and longitude of the reporting sensor, separated by a comma |
| day | long | Day component with time fixed at midnight, in milliseconds since the Epoch. For example, given the date "April 7, 2015 22:26" (1428438360000 in UTC milliseconds), the value of this field would be 1428357600000, |

| | | |
|---|---|---|
| | | representing "April 7, 2015 00:00". This facilitates retrieval of aggregated values that fall on a particular date. |
| time | int | Time component of a 24-hour clock, represented as an integer, e.g. 2100 for 9:00pm. This facilitates retrieval of values within a time range, such as from 6am to 11am. |
| total | double | Sum of logarithms of raw decibel readings received on this day within this time slot. |
| count | int | Number of raw decibel readings received on this day within this time slot. |

### SensorAggDOWData

Stores sensor data aggregated by day of the week (e.g. Monday) and time (in hourly increments).

| Field Name | Type | Description |
|---|---|---|
| provider:sensorId:day:time | Datastore Key | Unique key for this entity. |
| provider | int | Identifies the data source provider. |
| sensorId | String | Unique identifier for this sensor. Provided by source. |
| latLng | String | Latitude and longitude of the reporting sensor, separated by a comma |
| dayOfWeek | String | Day of the week represented as a string: e.g. "mon", "fri" |
| time | int | Time component of a 24-hour clock, represented as an integer: e.g. 2100 for 9:00pm. |
| total | double | Sum of logarithms of raw decibel readings received on |

| | | |
|---|---|---|
| | | this day within this time slot. |
| count | int | Number of raw decibel readings received on this day within this time slot. |

### SensorRawAudioData

Stores the raw audio data received by the Noise Identifier module. The raw audio data is stored so that it can be played back later or used in other analyses.

| Field Name | Type | Description |
|---|---|---|
| provider:sensorId:timestamp | Datastore Key | Unique key for this entity. |
| provider | int | Identifies the data source provider. |
| sensorId | String | Unique identifier for this sensor. Provided by source. |
| timestamp | long | Time when this audio data was received, in UTC milliseconds |
| latLng | String | Latitude and longitude of the reporting sensor, separated by a comma |
| format | String | Audio data format, represented as a String: e.g. “WAV”. |
| bytes | Blob | Audio data bytes. |

### SensorAlert

Represents some type of exceptional event reported by a sensor.

| Field Name | Type | Description |
|---|---|---|
| provider:sensorId:timestamp | Datastore Key | Unique key for this entity. |
| provider | int | Identifies the data source provider. |
| sensorId | String | Unique identifier for this sensor. Provided by source. |
| timestamp | long | Time when this audio data was received, in UTC milliseconds |
| latLng | String | Latitude and longitude of |

| | | |
|---|---|---|
| | | the reporting sensor, separated by a comma |
| state | String | Indicates whether the alert is ACTIVE or INACTIVE. |
| type | String | Specifies the type of the alert, such as “NOI_ID” (Noise of Interest Identified). |
| noiseId | String | If type is “NOI_ID”, the id of the noise that was identified, such as “SHOUT” or “MOTO”. |
| audioFormat | String | Audio data format, represented as a String: e.g. “WAV”. |
| audioBytes | Blob | Audio data bytes. |

### Region

Represents a particular region of the map. This is used in visualization to allow different regions to be monitored from the same framework. Providers are typically associated with a particular region.

| Field Name | Type | Description |
|---|---|---|
| name | Datastore Key | Identifies this region. Also acts as the unique key for this entity. |
| provider | int | Identifies the data source provider. |
| centerLatLng | String | The latitude and longitude of the region’s center, separated by a comma |
| zoomLevel | int | Initial zoom level. Used by the visualization (mapping) module when displaying the region. |

## Browser-Based Visualization

This section describes the technology that was used to render the retrieved and persisted sensor data into a visual representation.

### Google Maps for Developers

Google introduced their popular Maps product to developers in 2005 and is now one of the most popular web mapping services available today [36]. The JavaScript API allows a highly customizable map to be embedded in a web application. The map is actually composed of individual squares called *tiles* that are downloaded asynchronously so that the page doesn't have to reload as the user zooms or pans through the map. This *slippy map*, so-called because the map seems to "slip" as the user drags it with the mouse or pointing device, produces a better, more intuitive user experience. Layers such as markers (i.e. the iconic red pushpin), circles, and informational popups can be also added to the map to produce a fully-functional solution.

The following screenshot shows a rendering of an area of Barcelona (around Placa Catalunya) with yellowish circles showing (simulated) sensor locations. The circles are shown in this color because the latest decibel readings from those sensors fall within a certain range (in this case between 60 and 65). Different decibel ranges are represented by different colors, allowing the user to get a sense of the noise level at a particular area at a glance.

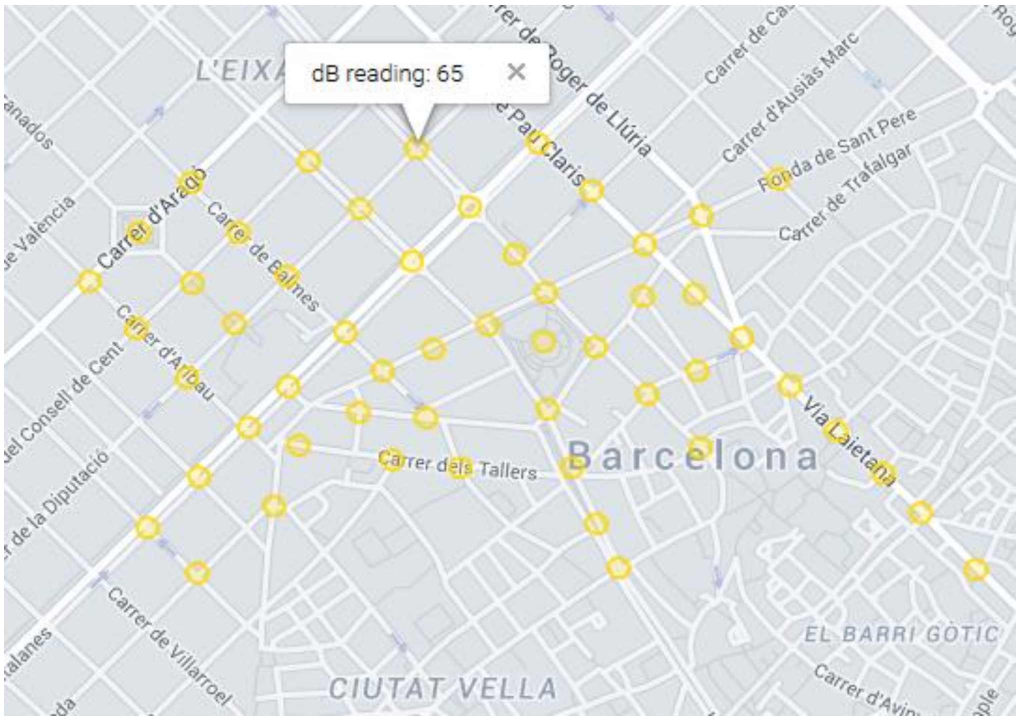


Figure 7: Google Maps rendering of Barcelona showing sensor locations as circles

The following screenshot shows the same area of Barcelona rendered this time using satellite images and with a heatmap effect. Higher noise level readings will be shown in a deeper red tone, giving the user a visual impression of a "hot spot" of noise.

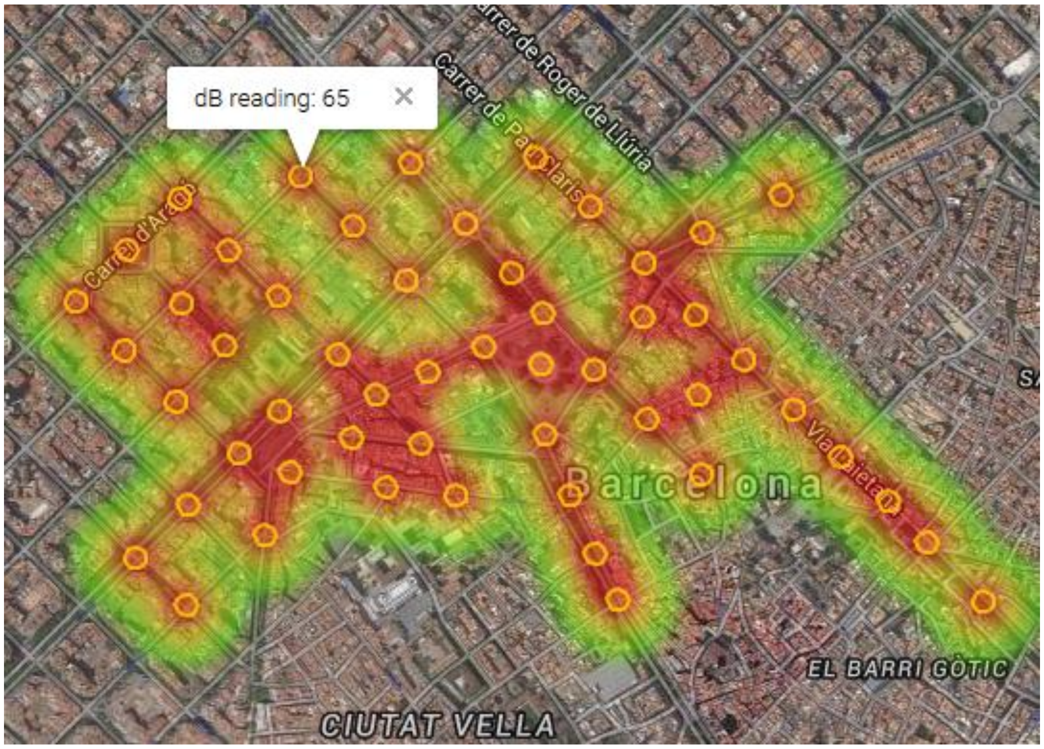


Figure 8: Google Maps rendering of same data but with satellite images and a heatmap layer

## Google Maps and OpenStreetMap

Using Google Maps for visualization was already the first choice right from the start, but in the interest of academic rigor, we also looked into the possibility of using an alternative to Google Maps, the most popular of which is OpenStreetMap (OSM).

OpenStreetMap is an online project founded in 2004 for creating and distributing free geographic data that covers the whole world. Hundreds of thousands of cartographers, both amateur and professional, contribute to its database of map data, which is then distributed under an Open Database License. OSM is often described as the "wikipedia of map data" due to this community-monitored, crowdsourced approach [37]. OSM data has been used successfully by several companies, such as Foursquare and Craigslist.

It should be noted that OSM produces map data, not tiles (map images). In order to embed a map in an application it must be used along with other libraries such as Leaflet (for creating layers such as markers and for interaction handling such as zooming and panning) or Mapnik (for tile rendering).

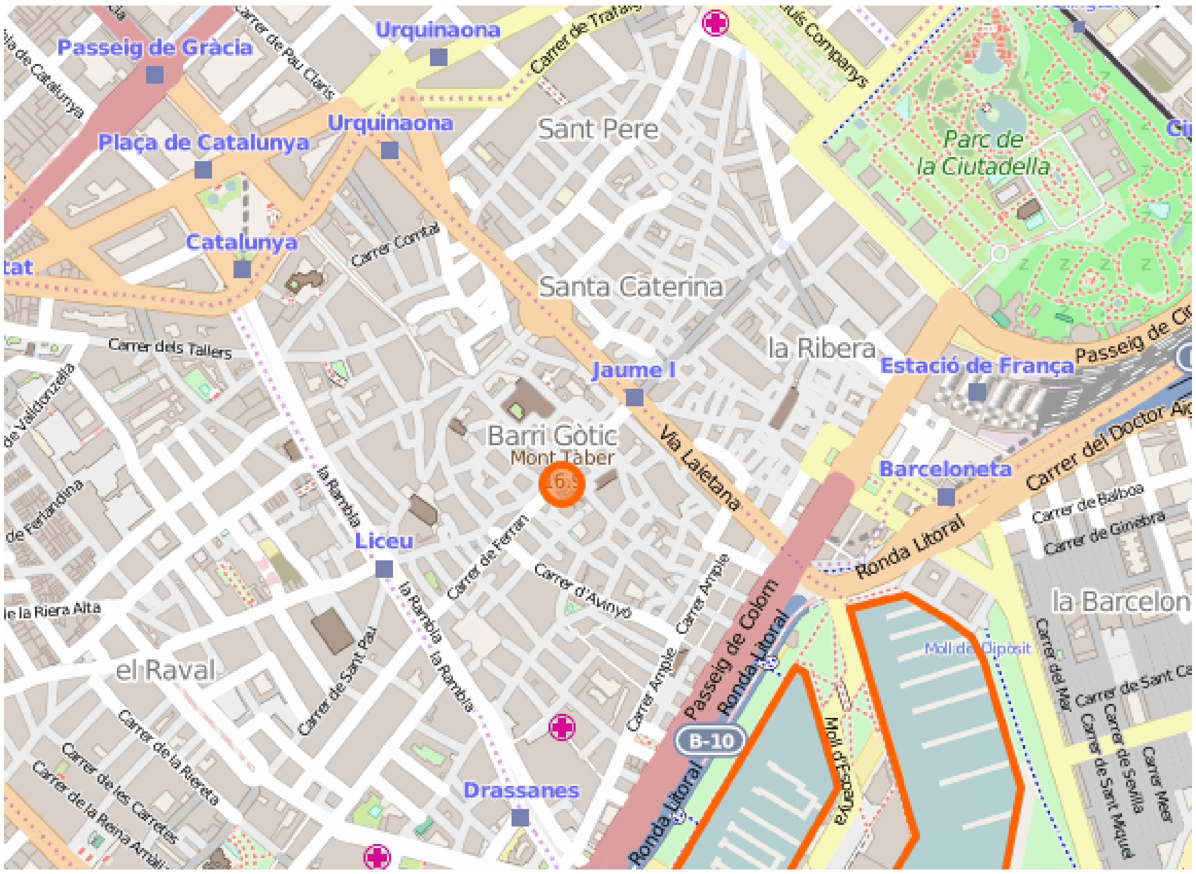


Figure 9: Mapnik rendering of Barcelona using OSM data (screenshot from openstreetmap.org)

One advantage of OSM is the possibility of creating offline maps, which can be achieved by pre-downloading OSM map data for a region and having a local installation of Mapnik, for instance, available to render the tiles. However this was not a requirement for this project, since the data to be visualized is expected to always be retrieved online.

The main argument against Google Maps is their use of proprietary data, provided by companies such as NAVTEQ and Tele Atlas. Applications that utilize Google Maps are subject to their Terms of Service, which include, for example, the right to display ads in the future [36] (it currently does not).

Also, Google imposes a limit of up to 25,000 map loads/day for non-paying clients. However since the project does not expect a huge volume of requests at this point, this was considered a non-issue. Also, Google will only deny requests when a site has exceeded the daily limits for more than 90 consecutive days, in order to allow for sites that experience short-term spikes in traffic [38].

In the end the decision to use Google Maps was based on several criteria:

- **Maturity and familiarity with the API** -- Google Maps has been around for more than 10 years and has plenty of developer support available online. Also, I already have some previous experience with Google Maps, and as such the curve for learning to use features such as heatmaps or infowindows would not be too steep.
- **Complete solution** -- Utilizing OSM requires integration and knowledge of several other libraries (Leaflet, Mapnik, etc.), whereas Google provides all of the functionality needed by the project (tiles, satellite imaging, zooming, panning, etc.) in one fairly straightforward API.
- **Responsiveness of rendering** -- Google has hundreds and thousands of servers worldwide that returns map tiles quite fast. This was a "nice-to-have", since, as noted above, the project at this point is not expected to handle a large number of requests. Having the map data and tiles rendered from a third-party server (i.e. Google's) is preferable than having to install and manage a rendering toolkit such as Mapnik on our own server.
- **Aesthetics** -- The tiles from Google Maps were considered better looking than the ones typically used with OSM.

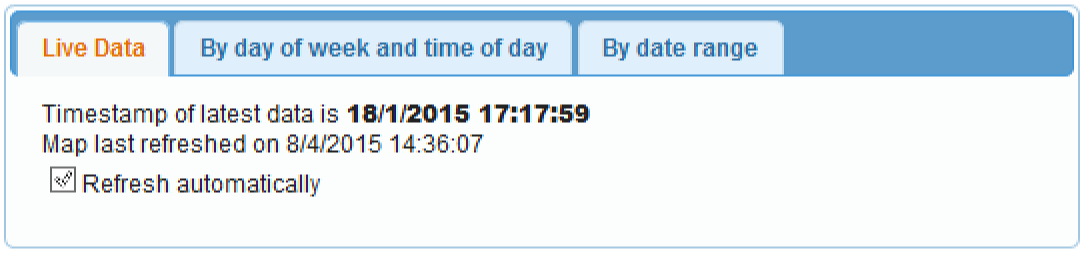
However, since the project does not utilize advanced features such as point-to-point navigation or panoramic street-level views, using OSM along with a mapping library is still a viable option for the project's visualization module. This could be a potential area for future work, and could also open new features that take advantage of OSM's high level of customization.

## JavaScript Controls using jQuery UI

The jQuery UI library is a managed set of commonly-used controls (widgets), themes, and interaction effects written in JavaScript and CSS. The library provides a set of cross-browser, cross-platform UI elements that are often used in many web applications, such as date pickers and sliders. It also extends existing HTML elements, such as buttons and select dropdowns, in order to overcome limitations and add functionality to enrich the web experience [39].

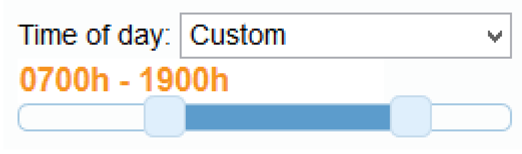
The following jQuery UI widgets were used in the web application:

- **Tabs** -- allows the user to switch between showing live data, or searching by day of week or by a certain date range.

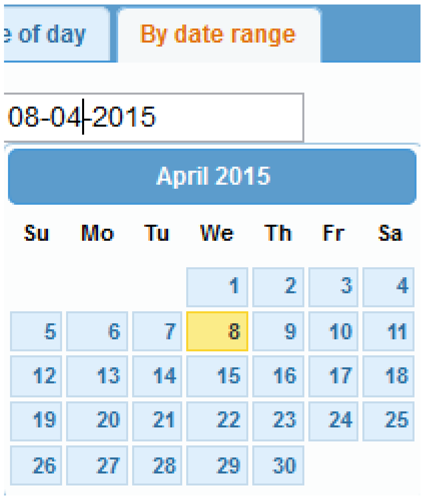


Timestamp of latest data is **18/1/2015 17:17:59**
Map last refreshed on 8/4/2015 14:36:07
☑ Refresh automatically

- **Slider** -- helps the user specify a time range when querying for noise data.


Time of day: Custom
0700h - 1900h

- **Date Picker** -- used when querying for noise data within a date range.



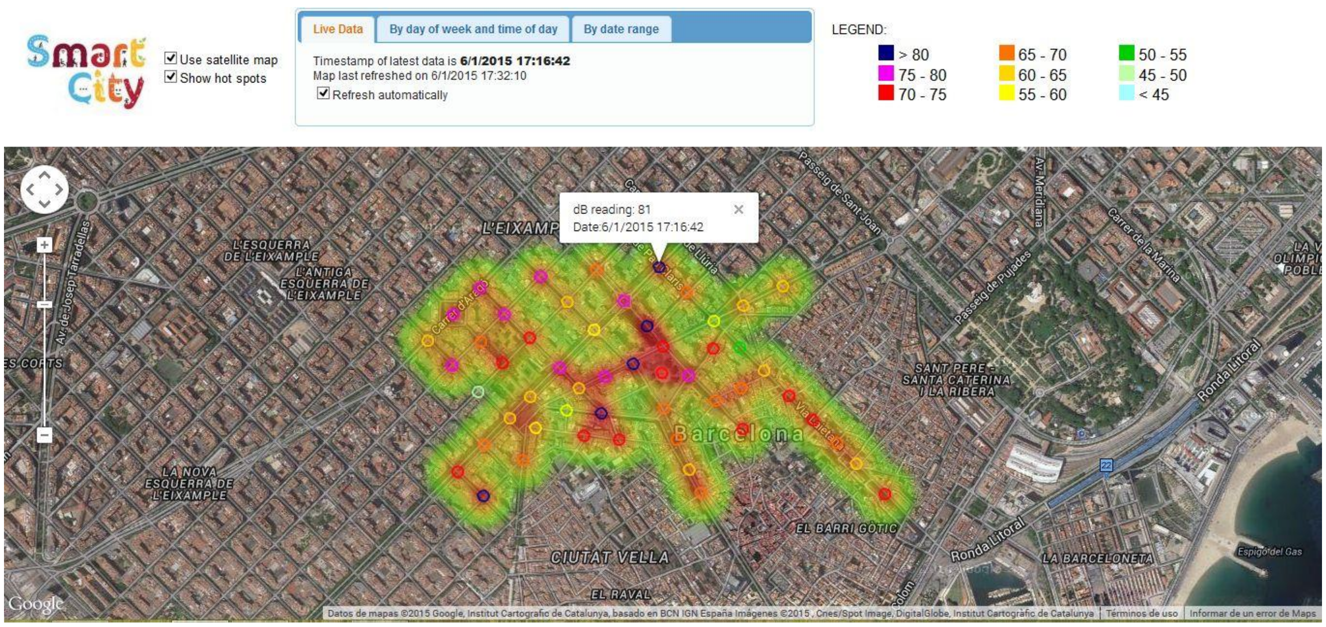
08-04-2015

April 2015

| Su | Mo | Tu | We | Th | Fr | Sa |
|---|---|---|---|---|---|---|
|  |  |  | 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 | 25 |
| 26 | 27 | 28 | 29 | 30 |  |  |

## Putting it all together

The following screenshot shows how the various components discussed above were put together to form the browser-based visualization module of the project. The tabs in the top center of the page let the user specify the query parameters of the data to retrieve, based on the three use cases mentioned earlier. The user can toggle between a satellite map or a normal map, and also whether to use the heatmap or not.



☑ Use satellite map
☑ Show hot spots

Timestamp of latest data is **6/1/2015 17:16:42**
Map last refreshed on 6/1/2015 17:32:10
☑ Refresh automatically

LEGEND:

| | | |
|---|---|---|
| 🟦 > 80 | 🟧 65 - 70 | 🟩 50 - 55 |
| 🟪 75 - 80 | 🟨 60 - 65 | 🟩 45 - 50 |
| 🟥 70 - 75 | 🟨 55 - 60 | 🟦 < 45 |

dB reading: 81
Date:6/1/2015 17:16:42

Datos de mapas ©2015 Google, Institut Cartogràfic de Catalunya, basado en BCN IGN España Imágenes ©2015 , Cnes/Spot Image, DigitalGlobe, Institut Cartogràfic de Catalunya | Términos de uso | Informar de un error de Maps

Figure 10: Application Screenshot

## Supported Browsers

Development and testing of the visualization component was performed using Firefox browsers **version 33** and higher. Other browsers in which the visualization has been tested are **Internet Explorer version 11** and **Chrome version 41**.

Support for mobile browsers was not part of the project's final scope, since a better solution would be to have a dedicated application, therefore development was not focused on ensuring that the visualization renders well on mobile devices.

It should also be noted that load testing was not part of the testing done for this project, as the focus was more on the visualization features and the ability to compute averages over a considerably large dataset.

## Noise Identification

*Noise identification* aims to automatically classify environmental noise captured by microphones or other sound monitoring systems. This can be accomplished using signal processing techniques combined with statistical methods and machine learning algorithms, similar to what is done in speech recognition [40]. Similar terms for this technology include “noise classification”, “noise detection”, and “automatic noise recognition (ANR)”.

Noise recognition is not a completely new field, and several studies can be found in literature [40] [41] [42]. However, with the rise of IoT and deployment of environmental sensors in urban environments, it seems like a natural idea to integrate this technology in the suite of Smart City applications.

The following activity diagram shows the pipeline typically employed in noise identification applications:

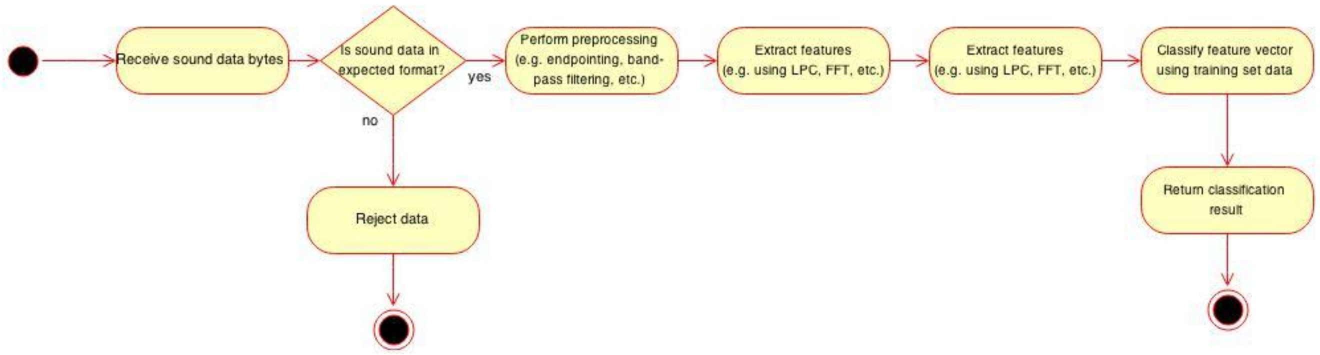

Receive sound data bytes
Is sound data in expected format?
yes
no
Perform preprocessing (e.g. endpointing, band-pass filtering, etc.)
Extract features (e.g. using LPC, FFT, etc.)
Extract features (e.g. using LPC, FFT, etc.)
Classify feature vector using training set data
Reject data
Return classification result

Figure 11: Noise Identification Activity Diagram

### WAV Audio Format

There are many audio formats available to represent digitized audio. One of the most popular is the Waveform Audio File Format, more commonly known as WAV. WAV files are comprised of *samples* of sound taken at a certain frequency, specified in Hertz, such as 8000 Hz or 44100 Hz (CD quality). Samples are typically 8 or 16 bits in size (the *bit depth*), and are typically uncompressed, resulting in *lossless* digitization of audio. WAV files also support multiple channels (e.g. mono or stereo) of audio [43].

The simplicity of the WAV file format makes them popular for capturing and editing audio files. Uncompressed audio also means that the captured sound is of high quality and is ideal for audio applications, with the added advantage of not having to perform decompression beforehand. However its major drawback is file size: a four-minute uncompressed WAV recording even at a “low quality” setting (8000 Hz, single-channel, 8-bit sample size) can easily consume 2MB.

For the purposes of noise identification, however, the size of the resulting WAV file is not a big concern, since the input sound samples are expected to be a few seconds long. At the "low quality" setting (8000 Hz, single-channel, 8-bit sample size), the input sound files will range from 30 to 300KB in size. These sizes are manageable enough to make them feasible for transmission (e.g. from sensor to the backend), processing, and storage in the system.

## Preprocessing

Once the sound has been sampled and digitized the result is essentially an vector of bytes that represent the sound data. Preprocessing may be done to further refine the input prior to the actual analysis. Two common ways of preprocessing the sound input are normalization and endpointing.

*Normalization* refers the process of adjusting a given set of values so that they are based on a common scale. In audio normalization the amplitude (loudness) is scaled so that two sound samples can be compared to each other correctly, without the results being affected by how loud one sample is compared to the other. One simple way to normalize amplitude is to find the maximum absolute amplitude value in the vector, then dividing all the amplitudes in the vector by this value so that they all fall into a certain range (e.g. [-1,1]).

*Endpointing* is a way of simplifying the sound input by only saving the local peaks (maxima) and valleys (minima) in the changes in amplitude. A data point is a local peak if it is greater than the data points before or after it in the vector; similarly a local valley is less than its neighboring data points. The detected peaks and valleys are collectively known as *endpoints*. The detected endpoints are then what's used in further processing; the non-endpoints are effectively discarded. This technique is usually performed on "noisy" time-series data. One variation is to also include the first and last data points in the vector, as well as contiguous points of equality (i.e. consider a data point as an endpoint if it is equal to the one before or after it in the vector) [44].

Other ways of audio preprocessing include high frequency boost filtering, low-pass filtering, and Fast Fourier Transform (FFT) filtering. The latter is also a way to extract features out of a sound sample, and is explained further in the following section.

## Feature Extraction

The process of finding the characteristics (*features*) that are useful for classification of a given input, whether it's text, image, or in this case, sound, is called *feature extraction (*also referred to as *feature detection).* The idea is to extract those features of the input audio that "stand out" and can be used to match against existing patterns, while discarding those other features that are not as useful. Features extracted from audio data are typically represented as a single *feature vector* of floating point values.

Two methods for extracting features are briefly described below.

*Linear Predictive Coding* (LPC) produces a “compressed” representation of the input by creating a linear predictive model of the data points. The output of LPC analysis is the set of coefficients of the linear predictive model along with an error term. These coefficients can be used to reconstruct the original input by predicting the value of data point i given the data points before it (i - 1, i - 2,...,i - k). This technique is widely used in speech recognition and synthesis systems [45]. The coefficients of the linear predictive model derived from the whole input are used as the feature vector for one given input.

*Fast Fourier Transform* (FFT) is an algorithm to compute the Discrete Fourier Transform (DFT) of a set of input values. The goal of a DFT is to transform a finite set of sample values from its original domain (i.e. time, in the case of sound, since each sample value represents sound at a given point in time) to the frequency domain (i.e. how much of the input samples fall into a range of frequencies). This frequency domain representation of the input can be considered as the features of that input, and is then suitable for use in classification [44].

## Training and Classification

The features extracted from the input can then be used either to build a *training set,* or compared against the training set in order to *classify* the input. The training set can be thought of as a collection of features that are grouped into known *classes* (e.g. recordings of a person screaming, or a motorcycle revving its engine). The training set is typically built a priori using previously-obtained *training data*. Once a training set has been created, a *classifier* can use it to determine to which class (if any) an input feature vector belongs. Two possible types of classifiers (out of many) are briefly described below.

*Distance-based classifiers* compute the *distance* between the feature vector to be classified and the *center* of each of the classes in the training set. The center of a class is computed as the average of the feature vectors that belong in that class. The classifier returns the “closest” class, that is, the one which has the least distance between its center and the input feature vector, as the classification result. There are several distance calculation formulas that can be applied: Euclidean, Chebyshev, Minkowski, Mahalanobis, just to name a few.

*Stochastic classifiers* are essentially statistical models whose parameters are gleaned from the training data. Classification is then performed by computing the conditional distribution given the training data and the input feature vector, and thus producing the probability that the input belongs to a certain class. Some examples of stochastic classifiers include Gaussian mixture models and hidden Markov models (HMMs).

There are other possible types of classifiers, including artificial neural networks and support vector machines.

## The Modular Audio Recognition Framework (MARF)

While searching for free Java-based implementations of sound recognition algorithms, I came across the MARF project on sourceforge. MARF was developed by students at Concordia University in Montreal and is described by its authors as “a collection of voice/sound/speech/text and natural language processing (NLP) algorithms written in Java and arranged into a modular and extensible framework facilitating addition of new algorithms” [44]. The source code is released under a BSD-style license.

The MARF tarball includes a speaker identification sample application, which uses MARF and its built-in preprocessing, feature extraction, and distance-based classification modules to identify the speakers of a set of WAV files recorded by the students themselves. Their best result was a rate of 82.76% correct identifications [44].

## Training and test data samples

For the sake of simplicity, input sound files were restricted to 8000 Hz, single-channel, 16-bit PCM WAV files.

A similar study on environmental noise detection [41] used samples of five different types of noise (cars, trucks, mopeds, aircraft, and trains). from the MADRAS database. Unable to gain access to this resource, we found another website called freesound.org and obtained 22 recordings of screams, 10 recordings of laughter, and 9 recordings of motorcycle noises.

The following images show the oscillograms of two WAV recordings. The oscillogram shows the amplitude (loudness) of the signal as time passes.

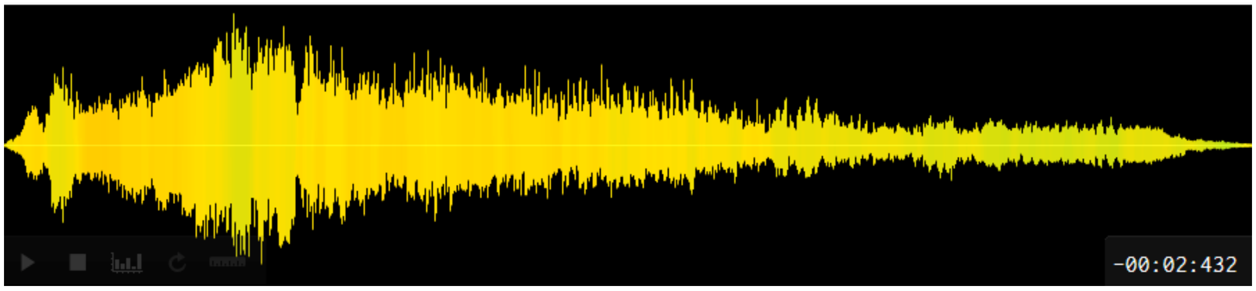


Figure 12: Oscillogram of a "scream" sample

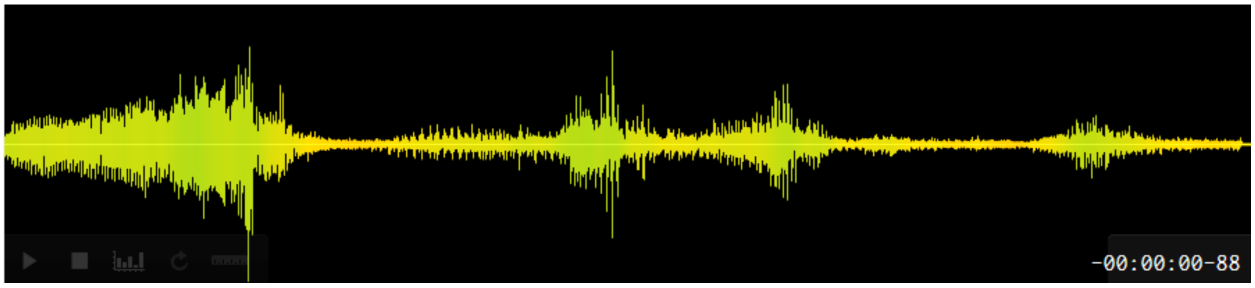


Figure 13: Oscillogram of a "laughter" sample

The oscillograms of the two recordings show two different amplitude patterns. While having distinctive oscillogram patterns does not necessarily imply that two samples are different (amplitude reflects loudness and does not by itself automatically constitute a feature), the images above provide a good visualization of the types of samples being used.

Out of the 41 total samples, 22 samples (12 screams, 5 laughter, 5 motorcycle) were used as training data, while the rest were used as test data to validate the resulting classifier.

## Results of noise identification using MARF

The noise identification module was then tested with the remaining 19 recordings, classifying them as either a person screaming, a person laughing, or a motorcycle running. The best outcome was 14 out of 19 correct, resulting in an accuracy of **73.68%**. The original authors of MARF achieved their best accuracy rate of **82.76%** in their speaker identification application [44].

The results of the noise identification using various combinations of preprocessing, feature extraction, and classification are shown in the table below:

| Preprocessing Method | Feature Extraction Method | Classification Method | **% Correct (correct/total)** |
|---|---|---|---|
| endp | lpc | cheb | 73.68% (14/19) |

| | | | |
|---|---|---|---|
| endp | lpc | eucl | 73.68% |
| endp | lpc | mink | 73.68% |
| endp | lpc | diff | 73.68% |
| endp | fft | cheb | 57.89% |
| endp | fft | eucl | 63.16% (12/19) |
| endp | fft | mink | 52.63% (10/19) |
| endp | fft | diff | 57.89% |
| norm | lpc | cheb | 68.42% (13/19) |
| norm | lpc | eucl | 57.89% (11/19) |
| norm | lpc | mink | 57.89% |
| norm | lpc | diff | 68.42% |
| norm | fft | cheb | 57.89% |
| norm | fft | eucl | 57.89% |
| norm | fft | mink | 57.89% |
| norm | fft | diff | 57.89% |

**Table 4: Results of identification using MARF (endp = endpointing, norm = normalization, lpc = linear predictive coding, fft = fast Fourier transform, cheb = Chebyshev distance, eucl = Euclidean distance, mink = Minkowski distance, diff = Diff distance)**

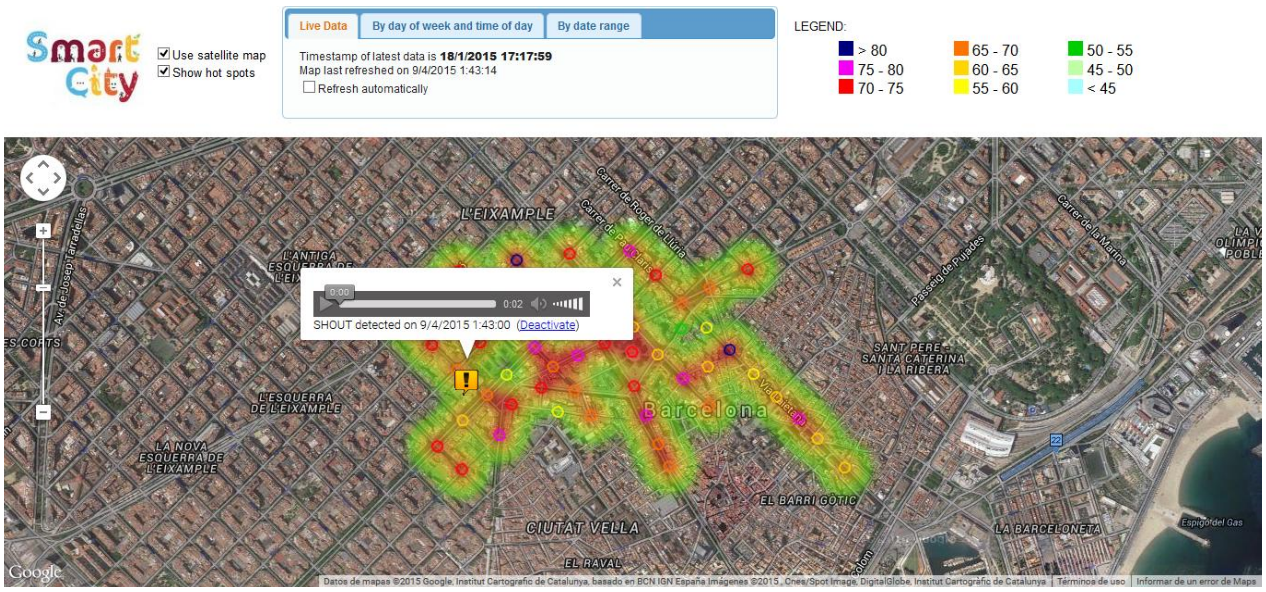
Note that with endpointing and linear predictive coding, the choice of distance-based classifier didn't matter since they all produced the same result with this dataset. In the end we decided to use the same configuration that produced the best result in the speaker identification study done by the authors of MARF, which was:

- Preprocessing: endp (endpointing)
- Feature extraction: lpc (Linear Predictive Coding)
- Classification: cheb (Chebyshev distance)

## Integrating noise identification into the application

The concept behind integrating noise identification with the visualization is to be able to pinpoint in near real time the location of a certain type of noise detected by a sensor. The REST endpoint /noiseid receives the sound data to be identified. The data is received as an HTTP POST with a "multipart/form-data" encoding type.

When sound data is received, it goes through the noise identification pipeline (format verification, preprocessing, feature extraction, classification). If the noise is identified as a something of interest (e.g. a person screaming), an alert is generated in the system. The browser-based visualization module detects this alert at the next refresh and displays the location where the noise was detected with a special marker icon. The user can also play back the sound that was sent by the sensor through the info window when the icon is clicked. The sound playback control is rendered using the HTML5 <audio> tag. The user can also dismiss the alert by clicking on the Deactivate link in the same info window.



☑ Use satellite map
☑ Show hot spots

Timestamp of latest data is **18/1/2015 17:17:59**
Map last refreshed on 9/4/2015 1:43:14
☐ Refresh automatically

LEGEND:

| | | |
|---|---|---|
| 🟦 > 80 | 🟧 65 - 70 | 🟩 50 - 55 |
| 🟪 75 - 80 | 🟨 60 - 65 | 🟩 45 - 50 |
| 🟥 70 - 75 | 🟨 55 - 60 | 🟦 < 45 |

0:00 0:02

SHOUT detected on 9/4/2015 1:43:00 (Deactivate)

Datos de mapas ©2015 Google, Institut Cartogràfic de Catalunya, basado en BCN IGN España Imágenes ©2015, Cnes/Spot Image, DigitalGlobe, Institut Cartogràfic de Catalunya | Términos de uso | Informar de un error de Maps

Figure 14: Screenshot showing an alert of an identified noise with audio playback

Because noise is constantly present especially during the day it makes more sense to enable noise identification only at certain times (e.g. from 3am to 6am, for example). It will also be most useful in areas of a city where human presence is not usually expected (e.g. parks late at night, alleyways or cul-de-sacs where people don't normally pass) and therefore certain environmental noises may be considered unusual and "out of place" and thus require special attention.

## REST Endpoints

The following table describes the REST endpoints exposed by the web application, along with the expected parameters and URI structure.

| URI | HTTP verb | Description |
|---|---|---|
| `/data/<prov>/latest` | GET | Returns the latest readings from all the sensors from the given provider <prov>. |
| `/data/<prov>/<sd>/<st>/<ed>/<et>` | GET | Returns the geometric average of sensor readings from the given provider <prov> in the specified date and time range, bounded by the start date <sd>, start time <st>, end date <ed>, and end time <et>. Dates should be in the format dd-MM-yyyy, e.g. "2015-04-10". Times should be represented as an integer, e.g. "2100" for 9:00pm. |
| `/data/<prov>/<dow>/<st>/<et>` | GET | Returns the geometric average of sensor readings from the given provider <prov> in the specified day of the week (<dow>) and time range (start time <st> to end time <et>). Aside from the seven days of the week ("mon", "tue", "wed", "thu", "fri", "sat", "sun"), the values "any", "wkd" (weekend) and "wrk" (workdays: represents monday to friday) are also supported. Times should be represented as an integer, e.g. "2100" for 9:00pm. |
| `/alerts/active` | GET | Returns all alerts in state ACTIVE in the system. |
| `/alerts/deactivate/<id>` | GET | Sets the state of the alert identified by <id> to INACTIVE. |
| `/alerts/noisedata/<id>` | GET | Returns the sound data (in WAV format) associated with the alert identified by <id>. |
| `/noiseid` | POST | Receives sound data to be identified as a potential noise of interest.<br>Parameters:<br>soundfile -- the sound data to be identified<br>latLng -- latitude and longitude, separated by a comma, of the location where the sound was captured |

## Test Cases

The following table enumerates some of the test cases used to verify the application:

| Name | Description and expected results |
|---|---|
| Home page load | When the user goes to the home page, they should see the latest sensor data visualized on a map of the default location. The visualization should show multicolored circles representing sensor locations over, by default, satellite images and a heatmap layer. |
| Automatic page refresh with Live Data | Under the **Live Data** tab, if the **Refresh Automatically** checkbox is checked, the application should fetch the latest sensor readings from the backend and update the map without user intervention. The page should also show the date when the map was last refreshed. |
| Info window popup on hover | When the mouse hovers any of the circles representing sensor locations, an info window should appear showing the latest reading (if the active tab is **Live Data**) or the average reading (on the other tabs) for that sensor. If the active tab is **Live Data**, it should also show the timestamp of that sensor reading (in a human readable format). There should only be one info window open at any time. Clicking on the x mark on the upper right hand corner of the info window should close it. |
| Color-coded sensor readings | The color of the circle representing the sensor reading should match the colors in the **Legend** shown in the upper right hand side of the page. |
| Toggle heatmap layer | When the **Show hot spots** checkbox is checked, the heatmap layer should be displayed on the map. When this checkbox is unchecked, the heatmap layer should disappear. |
| Visualizing data for a particular day of the week and time of day | Under the **By day of week and time of day** tab, the user should be able to show aggregated data based on the day of the week or the time of day. When the selected value of the **Time of Day** dropdown is **Custom** the user should be able to use the slider to select a valid time range, e.g. from 1100h to 2100h. The map should refresh automatically based on the user's selections. |
| Visualizing data for a particular date range | Under the **By date range** tab, the user should be able to specify a date range using the calendar controls that appear when the focus is on the text boxes. Note that the map does not refresh automatically while the user enters dates; they have to click on the **Show** button in order for the map to refresh with the requested data. When the By date range tab is selected, the map should be cleared of sensor circles and the heatmap layer should disappear. |
| Date validation | Under the **By date range** tab, if the user enters an invalid date range (**From** date is later than **To** date, or manually entered date does not follow the expected format), a popup should appear informing the user of the error. When this happens, the map should be cleared of sensor circles and the heatmap layer should disappear. |

| | |
|---|---|
| Noise data upload and identification | When an HTTP POST to the REST endpoint for noise data uploads is performed with the expected parameters (see next section), the application should respond with a JSON document indicating the results of the identification. |
| Alert display | When an alert is generated in the system (because of a positive identification of a noise of interest), it should be shown on the map within 1 minute of identification. The alert should be shown as a distinctive marker icon with an exclamation point. When the user clicks on the icon, an info window should appear showing the type of noise identified, the timestamp, and a link to deactivate the alert. The user should also be able to play back the reported audio within the info window. Clicking on the x on the upper right hand corner of the info window should close it. |
| Alert deactivation | When the user clicks on the **Deactivate** link in the alert info window, the alert and the info window should disappear from the map. |

**Table 5: Test Cases**

## Self-assessment of the Project

Overall, the resulting visualization and backend performance (querying and data storage) matched expectations. Google App Engine and its datastore made it relatively simple to host the application and its data, and query results are mostly performant: results were returned in a few hundred milliseconds, however no load testing was done to verify Google's claims of automatic scaling.

Google Maps produced beautifully rendered maps for the visualization. Expected map features such as panning, zooming, and switching between satellite images and drawn maps come for free and are familiar to the majority of users, who are already accustomed to other applications based on Google Maps. Other features such as circles, markers, info windows, and heatmaps were fairly easy to incorporate into the project, thanks to well-written documentation available online.

One observation of the final product is that it may seem a bit too "Google-centric". Another choice for the data store might have been a way to "balance" the choice of dependencies. In particular, a document store such as MongoDB would've been perhaps a better choice for the data store. A document store might have simplified some of the date- and time-range queries and aggregation, at the expense of extra network calls outside of the domain of the application host (Google App Engine in this case) to the document store server. "Might" is the operative keyword here, because unless modeling, development, and testing is done on an architectural decision one can only make conjectures on the result.

With regard to the data, we would have definitely preferred to have used real sensor data instead of simulated ones. Having real data would have made the visualization much more "meaningful", in the sense that it would reflect the real-world conditions of an area. More thoughts on the lack of access to real sensor data is noted in the Lessons Learned section below.

Having more data to use for training and testing is (usually) a good thing when it comes to classification applications, and the noise identification module is no exception. Implementing better algorithms for classification (see Future Work below) could also yield a higher success rate of 80% or more. This would be more in line with what's expected from a tool used in a security context such as this.

## Lessons Learned

Aside from the technical knowledge and experience garnered from designing and implementing the project, there were some other takeaways in the course of the research:

- The Smart City concept has taken off internationally, however a universally agreed-upon definition of what exactly comprises a Smart City is still a subject of debate. Annalisa Cocchia performed a comprehensive literature review of papers from 1993-2012 on this topic and found out that many cities would identify themselves as "smart" without clearly referring to an established meaning [1]. This might lead to the risk that the term "Smart City" becomes a hollow buzzword that will be used to blindly justify ICT projects in the city without clearly outlining their benefits. As one expert from Universitat Ramon Llull La Salle commented, the Smart City projects should "come from needs", instead of solutions trying to find problems to solve [12]. It is for this reason that several entities such as the Universidad Politecnica de Madrid choose to use the term "City Science" to emphasize knowledge coming from various fields applied rigorously with accurate and measurable results [46].
- One obstacle was the difficulty in obtaining sensor data. One of the assumptions at the start was that there would be an API available that would return up-to-date data from real sensors on the ground. It turns out that this was a fairly naïve assumption, as it seemed that the amount of sensors deployed in most cities is much less than expected. Furthermore even though data may be gathered, the API at best returned stale data (from months or years back). In other cases the documentation on how to access the data was unclear, and "dead links" were encountered in some of the pages. Clearly there is still a lot of work left to take advantage of sensor technology in cities. Hopefully when the time comes that live sensors and functional APIs are in place the project can be fully realized as a potential tool in urban design.

## Future Work

The final output of the project is a usable web application, however there are still possible areas for future work. The following lists the most prominent ideas:

- **Adapting the framework to other types of sensor data.** It's quite possible to extend the current data model and visualization modules to retrieve, store, analyze, and display other types of environmental data that can be returned by sensors, such as temperature and air pollution. However focusing on noise data was decided upon early on in order to limit the scope of the project .

- **Noise data visualization for mobile devices**. One of the early ideas in the project was to create both a browser-based and a mobile version of the visualization, but it was later scrapped due to time constraints. Aside from providing the user with on-the-go time and space map of noise data in the city, a mobile version of the visualization can also use the user's location to show the nearest quiet spots, for example. Alerts generated by the Noise Identification Module can also be pushed to the app so that the user is immediately notified.
- **Noise identification using Hidden Markov Models**: the Noise Identification module could use some other type of classification algorithm than a distance-based one. One promising alternative is Hidden Markov Models (HMMs), which has been shown in a several studies to achieve a success rate (i.e. the rate of correctly classifying input noise) of 85% or higher on environmental noise samples [40] [41] [42]. There are open source Java-based implementations of HMMs available that could potentially be integrated into MARF.

## Acknowledgements

I would like to thank my advisor, **Xavier Franch**, for all his support, guidance, words of encouragement, and unwavering patience.
I would also like to thank **Julia Camps** of Barcelona Medi Ambient for her valuable feedback on the project, as well as **Jordi Marco** for the crash course on Android development.
Special thanks also go to the engineers at the **Smart Citizen Kit** project (http://smartcitizen.me/) for providing additional technical guidance for this project.

## Bibliography

[1] A. Cocchia, "Smart and digital city: a systematic literature review," *Smart City*, pp. 13-43, 2014.

[2] Andrea Zanella, Nicola Bui, Angelo Castellani, Lorenzo Vangelista, and Michele Zorzi, "Internet of Things for Smart Cities," *IEEE Internet Of Things Journal*, vol. 1, no. 1, Feb 2014.

[3] H. Kopetz, "Internet of Things," in *Real-time systems.*: Springer US, 2011, pp. 307-323.

[4] Libelium.com. Top 50 Internet of Things Applications -- Ranking. [Online]. http://www.libelium.com/top_50_iot_sensor_applications_ranking/

[5] SmartSantander. [Online]. http://www.smartsantander.eu

[6] "Global Innovators: International Case Studies on Smart Cities," UK Government Department for Business, Innovation and Skills, BIS research paper number 135 2013.

[7] Chicago Health Atlas. [Online]. http://www.chicagohealthatlas.org

[8] MDE Ciudad Inteligente. [Online]. http://www.mdeinteligente.co

[9] Distrito MedellINovation. [Online]. http://medellinnovation.org/distrito/

[10] Hong Kong Special Administrative Region Government Information Services Department. (2014, June) Hong Kong Fact Sheets -- Population. [Online]. http://www.gov.hk/en/about/abouthk/factsheets/docs/population.pdf

[11] Ajuntament de Barcelona. Barcelona Smart City. [Online]. http://smartcity.bcn.cat/en

[12] Lucas Laursen. (2014, November) Barcelona's Smart City Ecosystem. [Online]. http://www.technologyreview.com/news/532511/barcelonas-smart-city-ecosystem/

[13] Stephen A Stansfeld and Mark P Matheson, "Noise pollution: non-auditory effects on health," *British Medical Bulletin*, no. 68, pp. 243-257, 2003.

[14] Citiquiet.com. (2014, July) The Top 10 Noisiest Cities in the World. [Online]. http://www.citiquiet.com/blog/the-top-10-noisiest-cities-in-the-world

[15] University of New South Wales School of Physics Joe Wolfe. dB: What is a decibel? [Online]. http://www.animations.physics.unsw.edu.au/jw/dB.htm

[16] World Health Organization. World Health Organization's Guidelines for Community Noise 1999. [Online]. http://www.who.int/docstore/peh/noise/ComnoiseExec.htm

[17] Galen Carol Audio. (2007) Decibel (Loudness) Comparison Chart. [Online]. http://www.gcaudio.com/resources/howtos/loudness.html

[18] Wikipedia.org. Barcelona -- Population Density. [Online]. http://en.wikipedia.org/wiki/Barcelona#Population_density

[19] Barcelona Field Studies Centre. (2013, May) Barcelona: Urban Pollution. [Online]. http://geographyfieldwork.com/BarcelonaPollution1.htm

[20] Barcelona pel Medi Ambient. Noise in Barcelona: A Strategic Noise Map. [Online]. http://w110.bcn.cat/portal/site/MediAmbient/?lang=en_GB

[21] Barcelona Pel Medi Ambient. Pla per la reducció de la contaminació acústica de la ciutat de Barcelona 2010-2020. [Online]. http://w110.bcn.cat/MediAmbient/Continguts/Vectors_Ambientals/Energia_i_qualitat_ambiental/Documents/Fitxers/pla-soroll-web.pdf

[22] Barcelona Pel Medi Ambient. Strategic Noise Map. [Online]. http://w20.bcn.cat:1100/WebMapaAcustic/mapa_soroll.aspx?lang=en

[23] NOSQL Databases. [Online]. http://nosql-database.org/

[24] Juhani Kantola. (2012, May) Sensorisation Pilots in the city of Barcelona (22@ district area). [Online]. http://connectedsmartcities.eu/sensorisation-pilots-in-the-city-of-barcelona-22-district-area/

[25] Jeff Sutherland and Ken Schwaber, "The Scrum Guide," Scrumguides.org, 2013.

[26] Thoughtworks. Project Management Software | Mingle. [Online]. http://www.thoughtworks.com/mingle/

[27] MongoDB, "Top 5 Considerations When Evaluating NoSQL Databases," Whitepaper 2015.

[28] Ilya Katsov. (2012, March) NoSQL Data Modeling Techniques. [Online]. https://highlyscalable.wordpress.com/2012/03/01/nosql-data-modeling-techniques/

[29] Google.com. What is Google App Engine? [Online]. https://cloud.google.com/appengine/docs/whatisgoogleappengine

[30] F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M.,. & Gruber, R. E. Chang, "Bigtable: A distributed storage system for structured data," *ACM Transactions on Computer Systems (TOCS)*, vol. 26, no. 2, p. 4, 2008.

[31] Ilya Katsov. (2011, June) 35 Use Cases for Choosing Your Next NoSQL Database. [Online]. http://highscalability.com/blog/2011/6/20/35-use-cases-for-choosing-your-next-nosql-database.html

[32] Google.com. Quotas - App Engine -- Google Cloud Platform. [Online]. https://cloud.google.com/appengine/docs/quotas

[33] Google.com. Java Datastore API - Java -- Google Cloud Platform. [Online]. https://cloud.google.com/appengine/docs/java/datastore/

[34] Google.com. (2014, October) Aggregate functions in GQL. [Online]. https://code.google.com/p/googleappengine/issues/detail?id=3862

[35] Joe Gregorio. (2014, August) Sharding counters. [Online]. https://cloud.google.com/appengine/articles/sharding_counters?csw=1

[36] Wikipedia.org. Geometric Mean. [Online]. http://en.wikipedia.org/wiki/Geometric_mean

[37] Wikipedia.org. Google Maps. [Online]. http://en.wikipedia.org/wiki/Google_Maps

[38] OpenStreetMap.org. OpenStreetMap Wiki. [Online]. http://wiki.openstreetmap.org/wiki/Main_Page

[39] Google.com. Google Maps JavaScript API -- Daily Usage Limits. [Online].

https://developers.google.com/maps/documentation/javascript/usage

[40] jqueryui.com. jQuery UI. [Online]. https://jqueryui.com/

[41] A. Rabaoui, Z. Lachiri, and N. Ellouze, "Automatic environmental noise recognition," in *IEEE ICIT '04. 2004 IEEE International Conference on Industrial Technology, 2004.* , 2004, pp. Vol. 3, pp. 1670-1675.

[42] C. Couvreur, V. Fontaine, P. Gaunard, and C. G. Mubikangiey, "Automatic classification of environmental noise events by hidden Markov models," *Applied Acoustics*, vol. 54, no. 3, pp. 187-206, 1998.

[43] L. Couvreur and M. Laniray, "Automatic noise recognition in urban environments based on artificial neural networks and hidden Markov models," in *InterNoise*, Prague, Czech Republic, 2004, pp. 1-8.

[44] Wikipedia.org. WAV. [Online]. http://en.wikipedia.org/wiki/WAV

[45] MARF Research and Development Group. (2006) Modular Audio Recognition Framework v.0.3.0.5 (0.3.0-devel-20060226) and its Applications. [Online]. http://marf.sourceforge.net/docs/marf/0.3.0.6/report.pdf

[46] Steve Cassidy. Linear Predictive Coding. [Online]. http://web.science.mq.edu.au/~cassidy/comp449/html/ch09s02.html

[47] Universidad Politecnica de Madrid. Master in City Sciences. [Online]. http://www.citysciences.com/