PLATFORM FOR REPORTING ILLEGAL DUMPS

Trash Out

Jozef Vodička

Bachelor's Thesis
May 2011

Degree Programme in Information Technology
School of Technology

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

Author(s)

VODIČKA, Jozef

Type of publication
Bachelor´s Thesis

Date
29.4.2011

Pages

Language
English

Confidential
( ) Until

Permission for web publication
(X)

Title

PLATFORM FOR REPORTING ILLEGAL DUMPS

Degree Programme

Information Technology

Tutor(s)

PELTOMÄKI, Juha

Assigned by

VELIČ, Roman - Initiative group of University students

Abstract

This thesis describes designing and implementation of a platform for locating illegal dumps via mobile Android application and App Engine web application.

The final result presents a working mobile Android application able to get its current location using network or GPS provider, store it in local database and share information with web application. The web application on top of Google App Engine will be able to store all data received from an Android application. All stored locations in web application are going to be shown in Google Maps for internet visitors.

This solution allows public to report illegal trash around the world and help to keep the Earth cleaner. It will also help globally Ministry of environment and police to have a better overview over the current state of illegal dumps in their area.

Keywords

Java, Android, App Engine, Datastore, jQuery, Google Maps, XML, GPS, Illegal dump, trash

Miscellaneous

CONTENTS:

FIGURES

TERMINOLOGY

Java

Java is a programming language and computing platform first released by Sun Microsystems in 1995. It is the underlying technology that powers state-of-the-art programs including utilities, games, and business applications.

(www.java.com)

Cloud computing

Cloud computing is Internet-based computing, whereby shared resources, software, and information are provided to computers and other devices on demand, like the electricity grid. Cloud computing describes a new supplement, consumption, and delivery model for IT services based on the Internet, and it typically involves over-the-Internet provision of dynamically scalable and often virtualized resources. (Cloud_computing)

Google App Engine (shortcut GAE)

Google App Engine is cloud computing technology. It virtualizes applications across multiple servers and data centers.

Google App Engine is a platform for developing and hosting web applications in Google-managed data centers. It was first released as a beta version in April 2008. (code.google.com/appengine)

Datastore in Google App Engine

The App Engine datastore stores and performs queries over data objects, known as entities. An entity has one or more properties, named values of one of several supported data types. A property can be a reference to another entity. Unlike traditional databases, the datastore uses a distributed architecture to manage scaling to very large data sets. An App Engine application can optimize how data is distributed by describing relationships

between data objects, and by defining indexes for queries. The App Engine datastore is strongly consistent, but it is not a relational database.

(code.google.com/appengine/)

Android

Android is a mobile operating system. Android's kernel was derived from Linux but has been tweaked by Google outside the main Linux kernel tree.

The Android operating system software stack consists of Java applications running on a Java based object oriented application framework on top of Java core libraries running on a Dalvik virtual machine featuring JIT compilation.

(www.android.com)

API - Application programming interface

An application programming interface (API) is an interface implemented by a software program that enables it to interact with other software. It facilitates interaction between different software programs similar to the way the user interface facilitates interaction between humans and computers.

(Application_programming_interface)

GPS - Global Positioning System

The Global Positioning System (GPS) is a space-based global navigation satellite system that provides reliable location and time information in all weather and at all times and anywhere on or near the Earth when and where there is an unobstructed line of sight to four or more GPS satellites.

(Global_Positioning_System)

BTS - Base transceiver station

A base transceiver station (BTS) or cell site is a piece of equipment that facilitates wireless communication between user equipment (UE) and a network. UEs are devices like mobile phones (handsets), WLL phones, computers with wireless internet connectivity, WiFi and WiMAX gadgets etc.

(/Base_transceiver_station)

JavaScript

JavaScript is an implementation of the ECMAScript language standard and is typically used to enable programmatic access to computational objects within a host environment. It can be characterized as a prototype-based object-oriented scripting language that is dynamic, weakly typed and has first-class functions.

JavaScript is primarily used in the form of client-side JavaScript, implemented as part of a web browser in order to provide enhanced user interfaces and dynamic websites. However, its use in applications outside web pages is also significant. (JavaScript)

SQLite

SQLite is a software library that implements a self-contained, server less, zero-configuration and transactional SQL database engine. SQLite is the most widely deployed SQL database engine in the world. (www.sqlite.org)

gMap3

gMap3 is a jQuery plugin which allows many manipulations of the Google Map API version 3. (gmap3.net)

1 INTRODUCION

1.1 Thesis background

People are consumers. Nearly in all areas of our lives we produce trash. Nowadays all things are wrapped for protection. Things are used and not all parts can be used, so in the end there is too much trash.

People generally do not want to pay extra money for collecting trash in a proper way so they throw trash away on hidden or public locations. Some of this trash “is found” or “if somebody finds it”, is properly destroyed or recycled, but a lot of countries deal with the problem of illegal dumps.

On the other hand, there are people trying to protect our planet, by keeping it clean and safe for their kids and future generations, trying to help animals to survive, inventing and using new technologies to for example produce cleaner energy

Some of these people work or volunteer in “green” environmental organisations trying to protect our planet from any kind of destruction. The best known are *Greenpeace*, *WWF*, *EEA*, WRI, *Clean Up the World* - there are more of them. Members act on places they know. Somehow they need to get information what is wrong in all regions in the world. People are sometimes anonymously reporting areas of illegal dumps but there is no “official” or “organized and global” way how to do it.

The majority of people who are willing to report illegal dumps are not involved in any environmental organisation or NGO. Therefore the objective of this thesis was to build an easy to use service targeted on these ordinary people to unleash the potential of reporting illegal dumps.

The thesis project with codename *“TrashOut”* will allow people with an Android phone able to get accurate geographical position to report any illegal dump in the world. These reports will be sent to one global server, where all reports will be populated and shown on a website.

If people start using the new way of reporting trash, volunteers in neighbourhood or global organisations can act more quickly.

1.2 Motivation

The main motivation was to build a platform to locate trash all over the world and give people an opportunity to make proper steps to clean dump areas.

The aim of the thesis was to build a user friendly and responsive mobile application, capable of communicating with a web application over the Internet and store there information about locations. The web application enables visitors to see all places around the world, which were reported by other people, maybe even their neighbours.

There is no web-based application online yet which aims at collecting info about illegal dumps by ordinary people and therefore this solution can become a pioneer.

Hopefully this application attracts people and changes their actions against illegal dumps. The project is mainly experimental to be able to see how people react if they find reported trash in their area of interest or neighbourhood.

1.3 Initiative group of University students

Initiative group of University students wants to work with this application and maintain its idea. This application should help them in their next experiments with people who locate the trash or people who find this project.

They already know there are many initiative groups and organisations all over the world trying to get rid of illegal dumps, so they are open to allow others to enter this project.

They also want to see, how people will react, when they find out that some illegal dumps were located in their neighbourhood. This behaviour will help them to prepare proper steps to clean these areas.

1.4 Structure of the thesis

The thesis is divided into two main parts.

The first one (chapter two) describes the used technologies and tools needed to understand before starting designing and implementing this project.

The second one focuses on the design and implementing process of this project. Each part of the thesis describes both used platforms developed on top of Android OS and App Engine. In the last chapter Conclusion (chapter 5) the project is summarized.

2 TOOLS AND TECHNOLOGIES

2.1 Potential end-users of the application

The platform created in the thesis project for locating illegal dumps or trash on streets over all world will be used by several groups of people.

- The first group are the employees in the Ministry of Environment in cooperation with mayors of towns and cities. These people have influence and can prepare proper steps to keep their neighborhood and areas, they are responsible for, clean.
- The second group are woodsmen taking care of clean environment in their wards.
- The third group are volunteers and employees from large and well-known organizations who are trying to protect our planet. They can organize cleaning events for volunteers to clean reported illegal dumps or use their influence and enforce major corporations to act against these illegal dumps.
- The fourth but not the least group are ordinary people willing to protect our planet and keep it clean.

2.2 Used technology

Following paragraphs shortly describe the used technology from a software developer's point of view.

There is plenty of documentation available online or in written form. As a source of knowledge documentation written by Googlers on website *developer.android.com* and *code.google.com/appengine* was used. All used books are mentioned in the section *References*.

2.2.1 Android

> *„The 22-month-old startup, based in Palo Alto, Calif., brings to Google a wealth of talent, including co-founder Andy Rubin, who previously started mobile-device maker Danger Inc."*
>
> *17 August 2005 businessweek.com*

Six years after its founding Android has attracted millions of users all over the world. Nowadays major cell phone manufacturers like HTC, Sony Ericsson, Motorola, and Samsung use Google's mobile OS Android as their major platform. Building an application for Android market can potentially reach various people all over the world.

This mobile platform has several specifications. In the following sub-chapters Android OS devices are discussed in more detail.

2.2.1.1 Structure of Android programs

The following figure illustrates the Android system architecture.

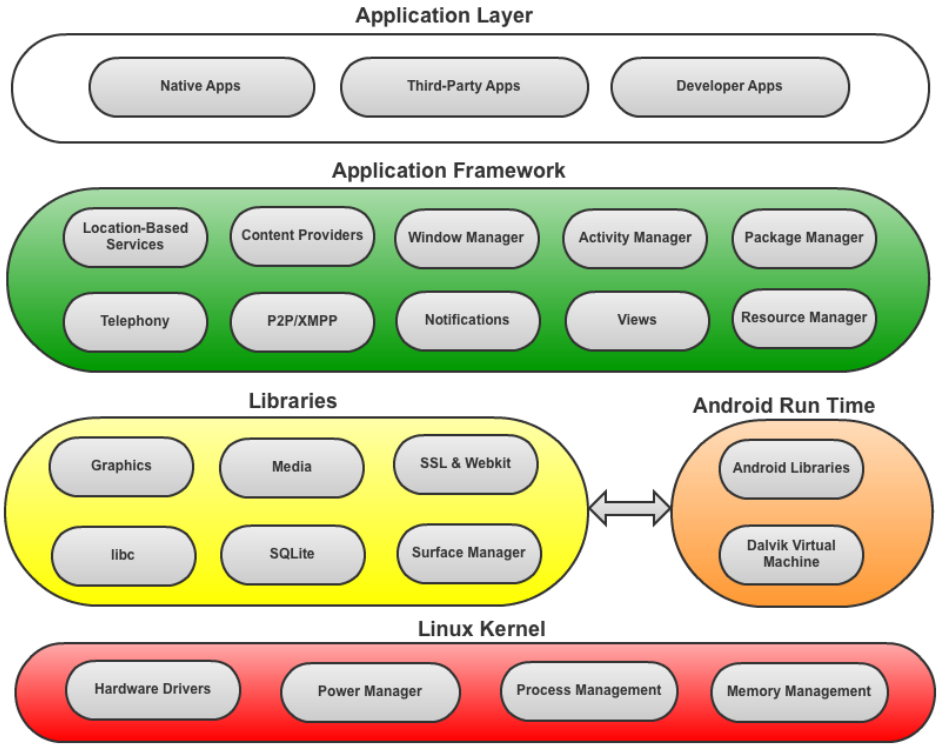

FIGURE 1. Android system architecture

The bottom layer is the Linux kernel (version 2.6), originally created by Linus Torvalds, which provides the hardware abstraction layer for Android, allowing Android to be easily ported to a variety of platforms. Android uses Linux for core system services such as security, memory management, process management, networking and other operating system services from Linux kernel.

Above the kernel is the layer, which contains Android native libraries all written in C or C++ and compiled for the particular hardware. These libraries are used by various components of the Android system. These capabilities are exposed to developers through the Android application framework.

On top of the kernel is also Android Runtime, which includes a set of core libraries with most of the functionality available in the core libraries of the Java programming language. Every Android application runs in its own process, with its own instance of the Dalvik virtual machine. Dalvik virtual machine is Google's implementation of Java optimized for mobile devices. It runs .dex files converted during compilation from .class and .jar.

Source files (.java files) are compiled into virtual machine-readable class files, which have a .class extension. The source code is compiled into an output file known as *bytecode*, which is stored in a .class file. A .jar file allows Java runtimes to efficiently deploy a set of classes and their associated resources. The elements in a JAR file can be compressed.

Compiled .dex files bring efficiency to battery-powered and limited memory devices. The virtual machine is register-based and runs classes compiled by a Java language compiler. Dalvik virtual machine relies on the Linux kernel for underlying functionality such as threading and low-level memory management.

Sitting above the native libraries and runtime is the Application Framework layer. This layer offers developers rich and innovative applications to build. Advantage can be taken of the device hardware, location services, run background services updated status bar and others. This layer was designed to simplify the reuse of components, so applications can share their capabilities with other applications. For example, a call can be made from our application just by calling an application, which can handle calls. These calls are made via Intents, which are described in section *2.2.1.5 Intents*.

The top Application layer uses advantage of all above mentioned layers and is depend on them. All Android based devices come with core applications such as Maps, SMS program, Browser and most of them are written using the Java programming language. New Android applications can be built also on top of the Adobe Air platform or Android NDK. The Android NDK is a companion tool to the Android SDK that lets build performance-critical portions of our apps in native code. It provides headers and libraries that allow building activities, handling user input, using hardware sensors, and accessing application resources when programming in C or C++. If native code is written, the applications are still packaged into an .apk file and they still run inside a virtual machine on the device. (developer.android.com)

2.2.1.2 Life cycle of Android application

Android operating system is designed for mobile devices. Activities are used for better management of Android application. Activities can be in one of several states as shown in Figure 2.

There is no control over what state the application is. Everything is managed by the system. Android applications can have different activities. If a new activity is started, it is placed on the top of the stack and it becomes the running activity whereas the previous activity always remains below it in the stack, and will not come to the foreground again until the new activity exits.

An activity can go frequently between the resumed and paused states, for example when a device goes to sleep or when a new intent is delivered. Activities that are not running in the foreground may be stopped. Therefore it is important that the application is designed from the beginning with this in mind. (developer.android.com)

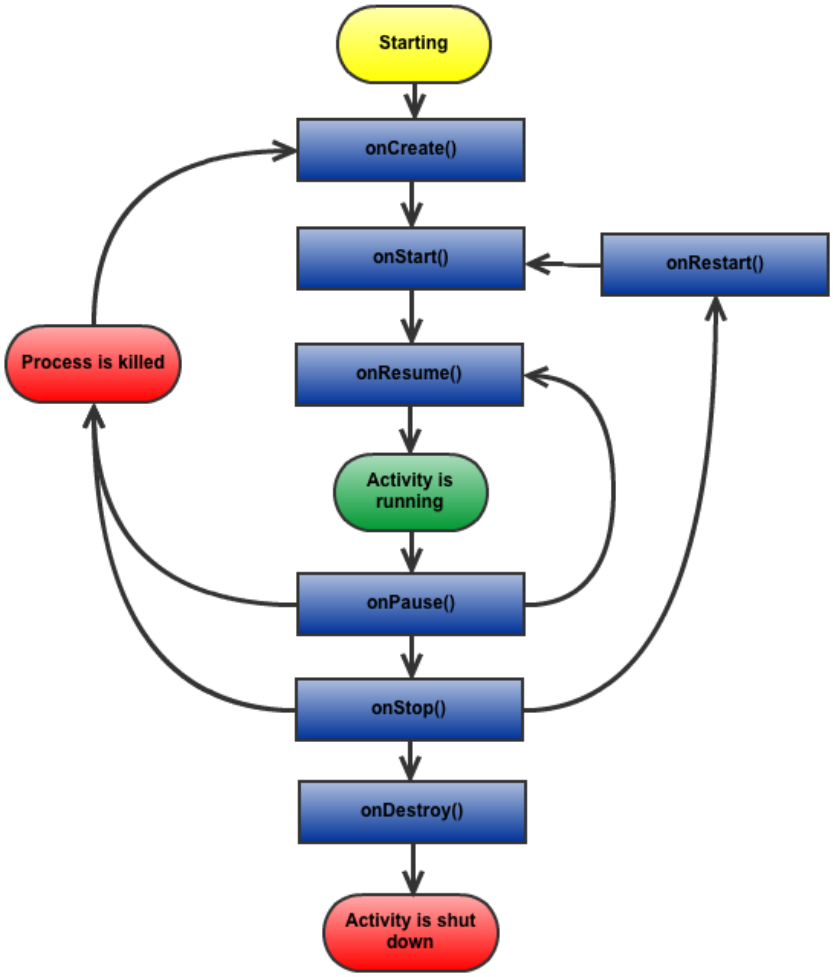

FIGURE 2. Life cycle of an Android activity

2.2.1.3 Activity

An activity can be thought as being the Android analogue for the window or dialog in a desktop application. Applications can define one or more activities to handle different phases of the program. Activity is responsible for saving its own state to be restored later as part of the application life cycle. (developer.android.com)

2.2.1.4 Services

A Service is not a separate process and it is not a thread. It runs in the same process as the application it is part of.

Services are designed to keep running, if needed, independent of any activity. A service might be used for play back music even if the controlling activity is no longer operating. It will run without the user's direct interaction. The music may be started by an activity, but the users might want to keep it playing even when they have moved on to a different program. Android comes with many services built in, along with convenient APIs to access them.

(Meier, 2008, 250)

2.2.1.5 Intents

Intent is an abstract description of an operation to be performed. Intents are system messages, running around the inside of the device, notifying applications of numerous events from hardware state changes to incoming data, or to application events.

This mechanism can be used for "take photo" or "call number". In Android, just about everything goes through intents, and therefore there are plenty of opportunities to replace or reuse components.

For example, there is intent for "take photo." If this project application needs to take a photo, that intent can be invoked and the standard Camera program can be easily replaced.

The developers can develop their own intents to launch other activities, or let users know when exact situations happen. For example, intent is raised when the user gets an SMS. (Burnette E., 2010, 39)

2.2.1.6 Content providers

Content providers provide a level of abstraction for any data stored on the device that is accessible by multiple applications. Content providers store and retrieve data and make it accessible to all applications. There is only one way to share data across applications. There is no common storage area that all Android packages can access. Android ships with a number of content providers for common data types (audio, video, images, personal contact information etc.).

Content that is created by developers can be available to other applications, while maintaining complete control over how their data gets accessed.

(developer.android.com)

2.2.1.7 Security and permissions

Every application runs in its own Linux process and is assigned a specific user ID. Linux isolates applications from each other and from the system.

> *"Additional finer-grained security features are provided through a "permission" mechanism that implements restrictions on the specific operations that a certain process can perform, and per-URI permissions for granting ad-hoc access to specific pieces of data."*
>
> (developer.android.com)

Permission to use must be specifically asked in a file named *Android-Manifest.xml*. When the application is installed, the *Package Manager* either grants or does not grant the permissions based on certificates and, if necessary, user prompts.

A central design point of the Android security architecture is that no application, by default, has permission to perform any operations that would harmfully impact the user, other applications, or the Android itself. This includes reading or writing the user's private data (such as contacts or e-mails), reading or writing another application's files, getting GPS location, etc.

(developer.android.com)

2.2.2 Database SQLite for Android

Android provides a lightweight relational database for each application using SQLite.

There are several reasons why SQLite is used by Android and the most popular ones are listed below as follows:

- It is for free. The authors have placed it in the public domain and do not charge for its use.

- It is very small. The current version in Android 2.3 (Gingerbread) uses SQLite 3.6.22 and is about 240KB.
- It requires no setup or administration. There is no config file, no server, and no need for a database administrator.

Because Android's SQLite database is specific, it is strongly recommended that all tables include an auto-increment key field to function as a unique index value for each row. If a table needs to be shared using *ContentProvider*, this unique ID field is mandatory.

Android comes with a helper class called *SQLiteOpenHelper*, which is made to manage database creation and version management.

A subclass was created in the project implementing *onCreate(SQLiteDatabase), onUpgrade(SQLiteDatabase, int, int)* and optionally *onOpen(SQLiteDatabase), onDowngrade(SQLiteDatabase, int, int)* . These functions take care of opening the database if it exists, creating it if it does not, and upgrading or downgrading it if necessary. It also helps *ContentProvider* implementations to postpone opening and upgrading the database until the first use.

(Burnette, 2010, 178-179)

2.2.3 Location services

Android devices can have access to several different means of determining the user's location. Some will have better accuracy than others. Some may be able to tell more than just the current position, but also elevation over sea level, or the current speed.

The most popular location providers are listed below:

- very popular GPS, which provides location and time information in all weather and at all times and anywhere on or near the Earth when and where there is an unobstructed line of sight to four or more GPS satellites
- cell tower triangulation, where our position is determined based on signal strength to nearby cell towers,
- proximity to public WiFi "hotspots" that have known geographic locations

Not all location providers are necessarily immediately responsive. GPS, for example, requires activating a radio and getting a fix from the satellites before a location is obtained. There is always “warming up” time that should be counted with. GPS is very hungry for battery power, so it needs to be used wisely otherwise the GPS based application will drain battery power in few hours. Choosing the right provider always depends on the requirements users have. Android can be asked for a provider with the best accuracy, information about altitude or cost free provider. (Burnette, 2010, 329 - 330)

2.2.4 Cloud computing with Google App Engine

Google App Engine lets run a web application on Google's infrastructure. It supports apps written in several programming languages. With App Engine's Java runtime environment, the app can be built using standard Java technologies, including the JVM, Java servlets, and the Java programming language, or any other language using a JVM-based interpreter or compiler, such as JavaScript or Ruby. App Engine can be communicated with via Python runtime environment, which includes a Python interpreter and the Python standard library.

Security of App Engine

The Java and Python runtime environments are built to run quickly, securely, and without interference from other apps on the system. Apps run in a secure environment that provides limited access to the core operating system. These restrictions allow App Engine to allocate web requests for the application across numerous servers. App Engine can start and stop servers to meet traffic demands. App Engine isolates an application to have a secure and reliable environment independent of the hardware, operating system and physical location of the web server.

Apps cannot write directly to the file system. An app can read files, but only files uploaded in this thesis project. The app must use the App Engine datastore, memcache or other services for all data that persists between requests. (code.google.com/appengine)

Request handling

App Engine is designed for web applications that respond to requests quickly. If an app routinely takes a long time to respond to requests, App Engine will decrease the priority of slow requests to make room for faster ones in the schedule. The less work the app does in response to a request, the more efficiently those requests can be distributed across multiple servers.

Figure 3 illustrates an App Engine's request handling architecture.

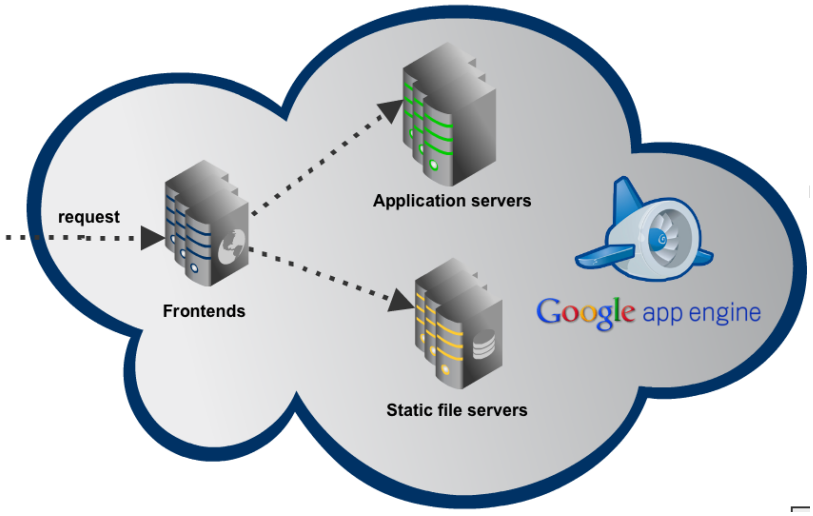

FIGURE 3. App Engine's request handling architecture

Frontend is a load balancer - a dedicated system for distributing requests optimally across numerous servers, and it routes the request to one of many frontend servers.

If the URL path of the request matches:

- the path of one of the app's static files, the frontend routes the request to the static file servers.
- one of the application's request handlers, the frontend sends the request to the app servers.
- with anything in the app's configuration, it returns an HTTP 404 "Not Found" error response.

Behaviour of fronted can be customized by App Engine developer.

(Sanderson, 2010, 63 - 64)

2.2.5 HTTP Servlets on Google App Engine

Java Servlets is a class used to extend the capabilities of an application hosted on server using request-response programming model. Servlets are not tied to HTTP requests, but they are most often used with this protocol. Servlets can be generated automatically from JavaServer Pages (*JSP)* by the JavaServer Pages compiler. The difference between Servlets and JSP is that Servlets normally embed HTML inside Java code, however JSPs embed Java code in HTML.

Ideally, a web application is an application that responds to web requests quickly, doing the smallest amount of work required to return a response.

All Servlets can be represented via customized URL in project file *web.xml*.

This is a recommended technology how to handle requests in App Engine, but other not yet included technologies can be configured such as Oracle JavaServer Faces for App Engine. (Java_Servlet)

2.2.6 Datastore in Google App Engine

App Engine provides a distributed data storage service and it includes a query engine and transactions. As the distributed web server grows with traffic, the distributed data store grows with data.

Data objects in App Engine *datastore*, or "entities" have a *kind* and a set of *properties*, which are not like in a conventional relational database. Queries can retrieve entities of a given kind filtered and sorted by the values of the properties. Property values can be of any of the supported property value types.

> *,,In Google App Engine datastore entities are 'schemaless'. "*
>
> *Google*

It can be stated that the structure of *datastore entities* is provided by the application code. The *datastore* is strongly consistent and uses optimistic concurrency control. Two types of storing data can be used - *Master/Slave Datastore* or *High Replication.*

Master/Slave option is a default for App Engine and it uses a master-slave replication system, which asynchronously replicates data as it is written to

another data center. At any given time, there is only one master in datacenter available for writing. This offers strong consistency for all reads and queries. The problem is during planned downtime, so it can become temporarily unavailable. This option costs less because of lower consumption of storage and CPU.

Data in *High Replication datastore* is replicated across data centers using a system based on the *Paxos algorithm*. It offers the highest level of availability, but requires around three times of storage and CPU than *Master/Slave* option.

All versions of an App access the same *datastore*, memcache, and other services, and all versions share the same set of resources.

(code.google.com/appengine)

2.2.7 JSP – Java server page

App Engine includes support for Java Server Pages (JSPs). In JSPs HTML and Java code can be mixed to make the page dynamic. App Engine compiles JSP files automatically before uploading them to server. App Engine server stores the compiled servlet classes.

By default, each JSP is mapped automatically to a URL path equivalent to the path to the JSP file from the application root. All JSP can later be mapped to customize URL in *web.xml* file. (Sanderson, 2010, 74)

2.2.8 Google Maps API

Google Maps API is a web mapping service application and technology. Google Maps API allows integrating Google Maps into developers' websites. It can be embedded in static or dynamic way, into the website using HTML, JavaScript or Flash. Before using this service, an API key needs to be obtained from Google. It is available for free after registration. This limit does not belong to Google Maps JavaScript in version 3 anymore. (code.google.com/apis/maps)

3 ANDROID APPLICATION DESIGN AND IMPLEMENTATION

3.1 System logic

The aim of the application is to get the most accurate geographic location for reported areas by the user. The faster it is done, the better it is. The same reporting experience has to be provided to an offline user as to an online user. Therefore all reported locations and needed information is stored to a local mobile database of this thesis project. Locations will get to the server when the user gets online again.

For each location, users can retrieve feedback and be motivated to continue in reporting other areas with illegal dumps. These feedbacks are submitted by other registered users on the server and they can describe their opinions about the reported locations. Feedbacks are downloaded from server, if there are some and a user is online. Finding the most accurate location will be transparent to users. In section *3.3. Location*, the way to get the most accurate position is described.

3.2 Life cycle of TrashOut application

The visualized life cycle of the application can be seen in *Figure 5*.

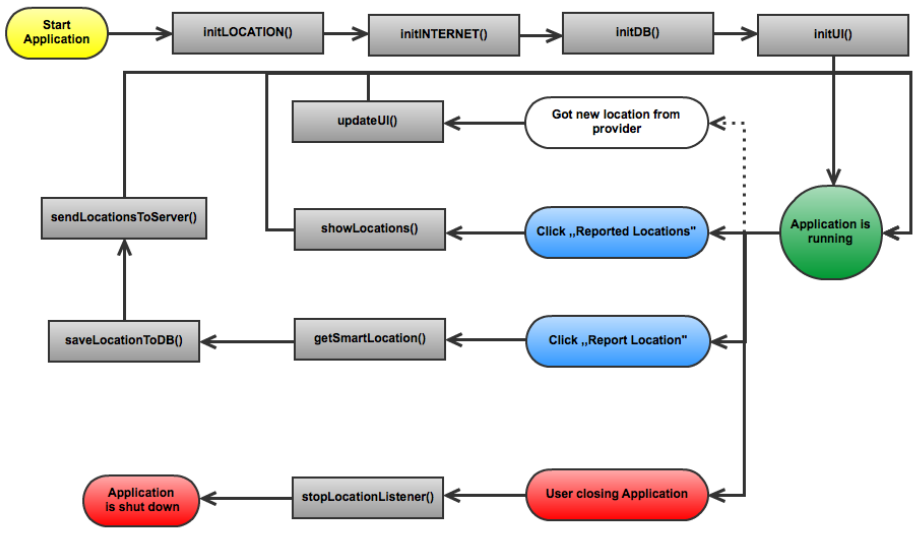

FIGURE 4. Life cycle of TrashOut application

Method **OnCreate()** is called every time a new activity is launched in Android. This happens on application start. The following implementation of this function in Java illustrates this code.

```
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        /** initialize all location providers to retrieve fresh location */
        initLOCATION();
        /** find out if we are online */
        initINTERNET();
        /** initialize our database */
        initDB();
        /** initialize user interface */
        initUI();
}
```

All activity needs to be properly initialized by calling *super.onCreate(bundle)*. In the thesis application there is a user interface so it is initialized with *setContentView(R.layout.main).* For a closer look on how user interface is implemented chapter *3.6 User Interface* can be referred to.

The warm up time for GPS provider is very long, typically more than 60 seconds. That is why communication is initialized with all location providers as soon as it is possible. It is done inside function *initLOCATION().* Firstly, it is checked if there are any location providers. If the device does not have any appropriate hardware for getting the geographical location, the user is informed with a dialog window that this application is pointless without any location provider and the application is stopped. In case hardware was found but it is not enabled, the user is asked via a dialog window to enable it in settings. In the last option the hardware is enabled and there are rights to get the accurate location info. Listeners are launched for these providers. These listeners will update the last known location to increase the accuracy.

Function *initINTERNET()* takes care about the internet connection. It is found out if the hardware for connecting to the internet is available. If there is no possibility to connect to the internet, user is informed about this reality and the application is closed. If hardware was found, but it is not enabled, application continues without any information to user. In case hardware is enabled public

domain as *http://www.google.com* is trying to be reached. If it works, users are online and this connection can be used later for communication with the server.

In function *initDB()* the object, child of the class *DbLogic* is initiated, which extends Android class *SQLiteOpenHelper.* This object takes care of the mobile database and holds all necessary functions for inserting, updating and deleting rows in each table. During the first lunch, when no database was found, it will create all tables in the database. It also checks the version of the database and if the version number was increased, it launches the upgrade database where all data from tables is transported to temporary tables; all tables are dropped recreated based on a new schema. After successful creation all data is transferred to the new database, based on a new schema.

Method *initUI()* will put to life all elements on a user's screen, because the application is listening to all button clicks. Info about location accuracy and count of uncommitted reported areas are also initiated. How the user interface looks can be seen in chapter *3.6 User Interface*.

Connected to the life cycle of Android mobile application, different actions can occur. The whole Android application life cycle can be seen in chapter *2.2.3.1 Life cycle of Android devices*. Users can leave devices unattended for longer time, devices run out of memory or battery; users get call and so on. Because of these unexpected events, different methods are implemented to handle these events.

When an activity is going into the background, but has not yet been killed it is calling *onPause().* Listening to location providers is stopped for any updates to save battery power and to kill all running tasks.

Method *onResume()* initiates listening to location providers and checks, if the Activity is in the front – if users can see it. Also users' screens are updated based on current accuracy and internet availability.

Method *onStop()* and *onDestroy()* implements the same code as it is inside method *onPause()*.

3.3 Location

It needs to be kept in mind that all location providers, which are not meant for army, are unreliable to some extent. When users walk inside a building, for example, GPS signals cannot reach them. That is why all resources, which are available, need to be used to determine the most accurate location.

3.3.1 Getting the best location

We do not know yet if location providers contain fresh and accurate locations so the developers' own algorithm is implemented to determine the best location.

The location UTC time, accuracy and current speed are used in Application so these informations are obtained from each location provider. These steps are iterated in interval of five seconds, in maximum up to 120 seconds. This interval was chosen because many of the GPS providers will get accurate position under two minutes.

1. From all available providers the last known location is asked.
2. Time will tell, if provided location is fresh enough to operate with. If the time is less than 20 seconds old it can be operated with. 20 seconds is the approximate process time of unlocking the phone, looking for an app and opening.
3. The information is based on Wikipedia article Walking about walking speed (Walking). If the speed is less than 5.43 km/h the user is walking; otherwise he/she is running, going by bike or by car. Walking in this project means that the location info age can be older than the location info obtained during riding a bike or a car. For walking a tolerance of 15 seconds was set and otherwise it is 5 seconds.
4. The accuracy tolerance was set to 25 meters, which means that exact location is less than 25 meters away from the current position.
5. This location is stored into a local database.
6. If the time of location is not fresh enough or the accuracy is more than 25 meters the provider is asked again for better location.

After 120 seconds or less there are many locations from different location providers for one reported area.

The internal logic is looked at next. The most accurate one is discussed first. Sometimes it can happen, that an older location can contain more accurate location than a younger one. This can happen if GPS signal is suddenly lost because of battery, for example.

Thus all saved locations are ordered by accuracy and then the top 3 by times. Then the youngest one with the best accuracy is chosen.

3.4 Storing data

All data saved in App database are stored in database tables *TypeOfTrash*, *Trash*, *Location*, *StatusOnServer*, *Feedback* and *Logger*. Their relations can be seen in Figure 5. Creating and upgrading database occurs after the first start of the application or after an upgrade. It is described in chapter *3.2 Life cycle of TrashOut application.*

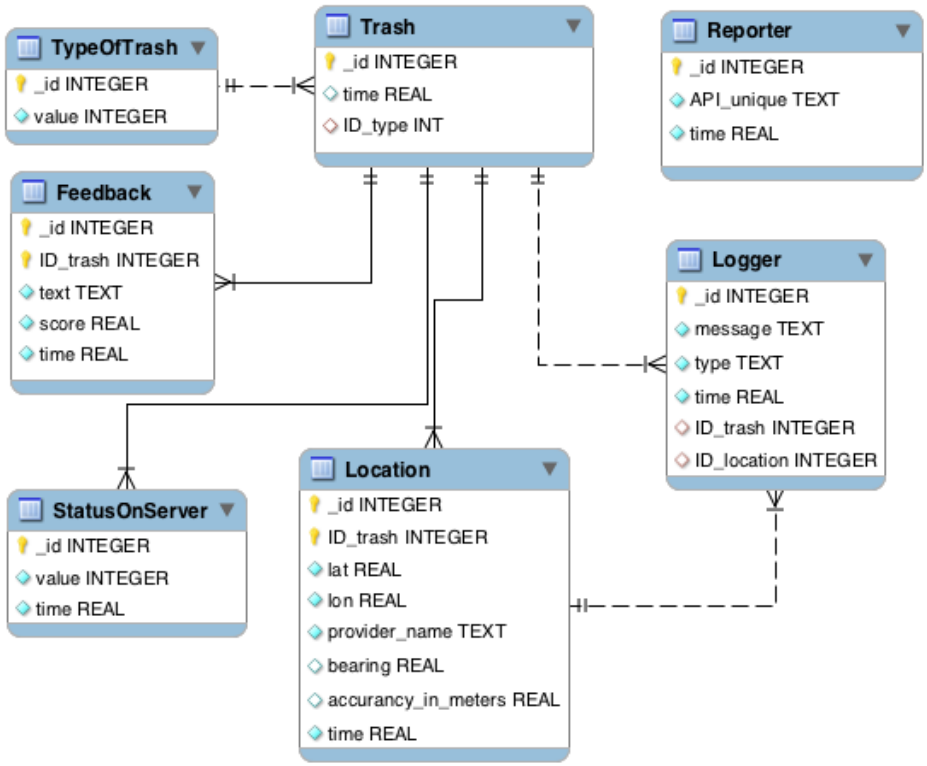

FIGURE 5. Android App Enhanced Entity-Relationship Model

TypeOfTrash

In this table information about types of trash is stored. Currently four different types of trash are stored.

0 - cleaned,

1 – small area,

2 – medium area,

3 – huge area

The description can be changed or new types added without affecting the already reported locations.

Trash

UTC time is stored when this location has been reported. It will be used for determining the best location coordinates, based on the application's smart location algorithm described in section *3.3.1 Getting the best location.*

The type of trash states the size of the reported area or other additional information.

Location

Location contains information about geographic location sensed at a particular time connected to the reported area. Latitude is stored in column *lat*, longitude in column *lon*, name of the location provider in column *provider_name*, approximate initial bearing in degrees East of true North in column *bearing*, accuracy in meters in column *accuracy_in_meters,* the UTC time of particular time for received geographic location stored in column *time*.

Status On Server

For each reported area is stored information based on the status of location in server.

The reported area can be in one of these states:

0 - not sent yet

1 - is sent to server

2 - is approved by server

3 - is cleaned

4 - is SPAM

The default value is 0 and it means it was not sent to the server yet. Server admin or registered users can mark any of the reported areas as SPAM. Column *time* states when the status was changed.

Feedback

On the server, registered visitors can be asked to feedback each location, or change its status. This feedback can be later viewed by other visitors and will be sent to the reporter's Android device. For each feedback its content is stored in column *text*, UTC time of given feedback in column *time.* The score can be in interval from 0 to 1. It will state how accurate this reported area is, measured by other users and the algorithm on the server.

Logger

In App development mistakes are made and the best way how to find out what went wrong is to see logs. That is why four different types of log exist and they are stored in columns named *type.* The types can be

- "error"
- "debug"
- "info"
- "warning"

The note, or content of an error message is stored in column *log_message* and to column *time* the actual UTC time is inserted.

If log is somehow connected to reported trash or location, it will be referred to an appropriate row in the table of trash, location or media.

3.5 Sending data to server

The whole service focuses on having all reported locations in one place - server. Therefore the Android based App has to be able to send all stored locations to it.

Obtaining a unique API key

Before submitting any location to the server, the user's device needs to register and obtain a unique API key. After each App starts it is determined if there is some appropriate hardware for the connection with the internet. This procedure is described in chapter *3.2 Life cycle of TrashOut application*.

If the hardware that is able to connect user to the internet was found it can be proceeded to the next step.

1. For testing HTTP connection used for communication with the server a globally accessible and reliable website *www.google.com* will be used. This test can last longer than 5 seconds, depending on the internet connection latency; therefore it is carried out in a separate thread to avoid unresponsive UI. Now when the Internet reachability is curtained another thread is created for communication with the server.
2. The next thread will firstly check if a *unique API key* has already been obtained in the local database. This aims at decreasing server requests and focussing just on requests that are needed. If there is no *API key* an XML request is sent to the server servlet:

```
<?xml version="1.0" encoding="UTF-8" ?>
<request>
  <id>012345678911223</id>
  <device_type>ANDROID</device_type>
</request>
```

   - *ID* of our Android device, it is SHA-1 hash from IMEI or uniquely generated Android ID and
   - *device_type*, in our case type of device is *android*.
3. Servlet *reporterRegister*, running on server will process this XML request. It will check if there was already a registered device with this ***id***.

- If yes, it will just return it is *unique API key* to device. This will happen if App is uninstalled and installed on a device.
- If there is no record with this ***id,*** servlet will create it with info about the device type and send a new generated *unique API key* back to the device in XML response. :

```
<?xml version="1.0" encoding="UTF-8" ?>
    <response>
      <api>915c2ddfcd1095f05296c422f4a1971f3cc889e1</api>
      <error></error>
    </reponse>
```

- How a *unique API key* is created is described in section *4.2 Receiving data from Android mobile device.*

4. Android App will extract API key from this XML response and store it in local database.
5. If there was any error in communication, all errors are logged and the retrieval of API for next App start is postponed.

Send uncommitted locations

When the App already has obtained the *unique API key* then it is possible to authorize the locations to the server and store them there. A simple procedure *sendLocationsUnsended()* was created which will determine check in local database if there are some locations with status "***0 – not sent yet".***

All found locations are going to be sent. All information about location is going to be sent to server via XML request in this format:

```
<?xml version="1.0" encoding="UTF-8" ?>
    <request>
      <id_api>915c2ddfcd1095f05296c422f4a1971f3cc889e1</id_api>
      <loc_provider_name>gps<loc_provider_name>
      <loc_accuracy_in_meters>23.8</loc_accuracy_in_meters>
      <loc_bearing>11.5</loc_bearing>
      <loc_latitude>35.123456</loc_latitude>
      <loc_longitude>115.123456</loc_longitude>
      <loc_time_reported>2415488651315</loc_time_reported>
```

```
      <loc_speed>0.0</loc_speed>
    </request>
```

Submitting locations to server is time consuming, so the procedure *sendLocationsUnsended()* runs in a separate thread. In current version of App each location is separately sent to a servlet running on server via HTTP Post Request and it needs to be waited for the response if the location was submitted correctly.

In case everything went well this XML response is received:

```
<?xml version="1.0" encoding="UTF-8" ?>
    <response>
       <lat>35.123456</lat>
       <lon>115.123456</lon>
       <status>inserted</status>
       <error></error>
    </response>
```

Information in XML response about *lat* – latitude and *lon* – longitude is just to ensure that appropriate response is received for a location already sent. The status can contain two options:

1. *inserted*
2. *duplicated*

This *status* information is just informative and can be used in further development.

Now the status of this location can be updated in the local database to *2 – Approved by server.*

Errors during submit

During communication with server an error can occur. Internet connection can be lost, or a user will close the app during location submit to server. That is why communication between Android App and server needs to be ensured that everything worked without any problems.

The server can send user in response also *error information,* which can look like the following:

```
<?xml version="1.0" encoding="UTF-8" ?>
    <response>
      <error>wrong api</error>
     </response>
```

These error messages are logged to a local database and the current task is ended.

3.6 User interface

Android uses XML based layout. This principle becomes commonplace for many platforms, because it puts the layout details in an XML file and the program code in source files. Thus the user interface can be easily changed anytime in the future without changing the internal logic.

UI of this App is designed to fit any kind of device with a different screen resolution and it was made as simple as possible.

In case the user changes the orientation of the screen from default - vertical to horizontal view, the same layout is kept as before.

Home screen

The first screen users will see after launching this app is the home screen, which can be seen in *Figure 6*.

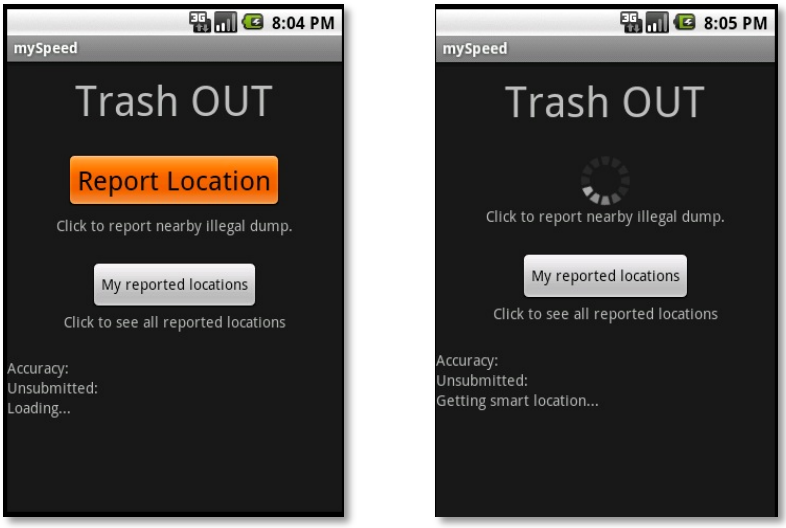

FIGURE 6. Home screen on the left side. Loading new location on the right side.

Via button “Report Location” users can initiate the process of getting the most accurate location from available providers.

Button “My reported locations” will get the user to another screen where there is the history of reported locations. This screen is shown in Figure 7.

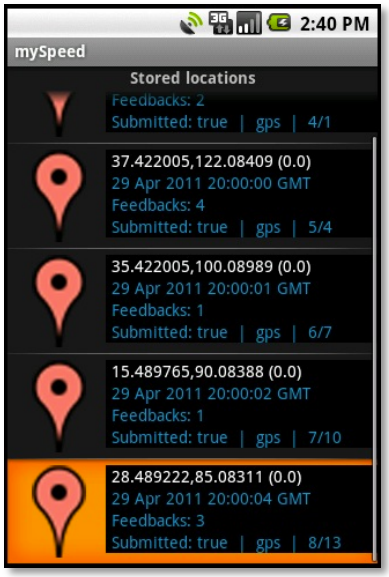

FIGURE 7. Image with list of reported locations

All reports are in one list. If some location was submitted already, it has information about it next to the marker icon.

Select record in list

After a tap on unsubmitted locations the user gets the option to delete this record as is shown in Figure 8.

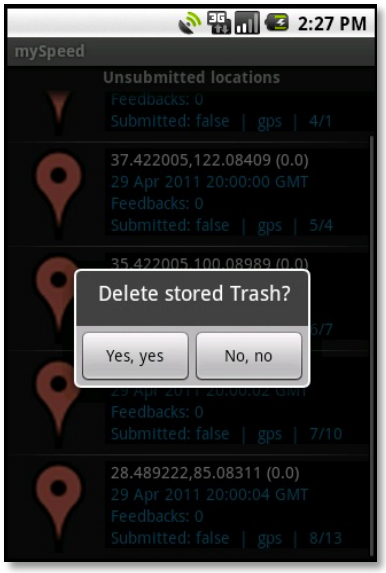

FIGURE 8. Image with question to delete this record

"Yes, yes" – selected record will be deleted from local database and will not be submitted to server.

"No, no" – selected record will remain in database

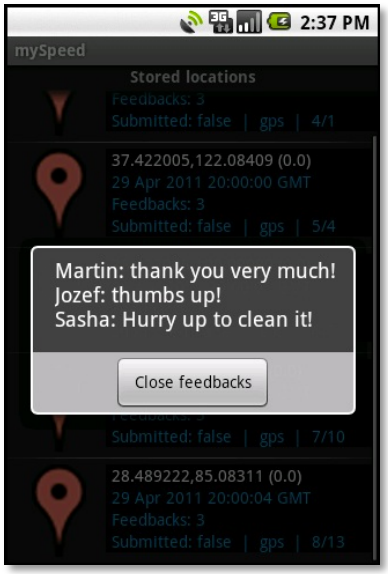

FIGURE 9. Image with 3 feedbacks on location

User can also tap on records which were already submitted to server. In this case, three latest feedbacks from users on this location are shown. This is illustrated in *Figure 9*.

User notifications

For notifying a user about upcoming or finished events, Android Toast Notifications are used. This notification overlays all elements showed on screen for a selected amount of time.

It is used for messages like

- "We are online"
- "We get the best location"
- "All locations were submitted to server"

If users' screen needs to be updated from another thread than was created, *Handler* included in package *Android.os* has to be used.

4 JAVA APPLICATION ON GOOGLE APP ENGINE: ITS DESIGN AND IMPLEMENTATION

4.1 Server side

Each application written for Google App Engine on top of Java Virtual Machine can contain static files, servlets, JSPs and Java applications.

The developer's server tries to take advantage of all of them, considering performance and further development of this application.

Before submitting this application to Google App Engine cloud system, a new account has to be created there.

4.2 Storing data in App Engine datastore

Scheme of the project's *datastore*, based on kinds is shown in Figure 10.

During setting up this application on App Engine, it had to be chosen what kind of data store would be used. Two options in section *2.2.6 Datastore in Google App Engine* were introduced previously. For this application *Master/Slave* option was used, because it is possible to live with planned downtime and save CPU time and storage – costs for running our server.

In this database information will be stored about

- registered devices and reporters,
- submitted locations with statuses, country and town info,
- submitted feedbacks for locations.

All created objects are saved to *datastore* via persistence manager.

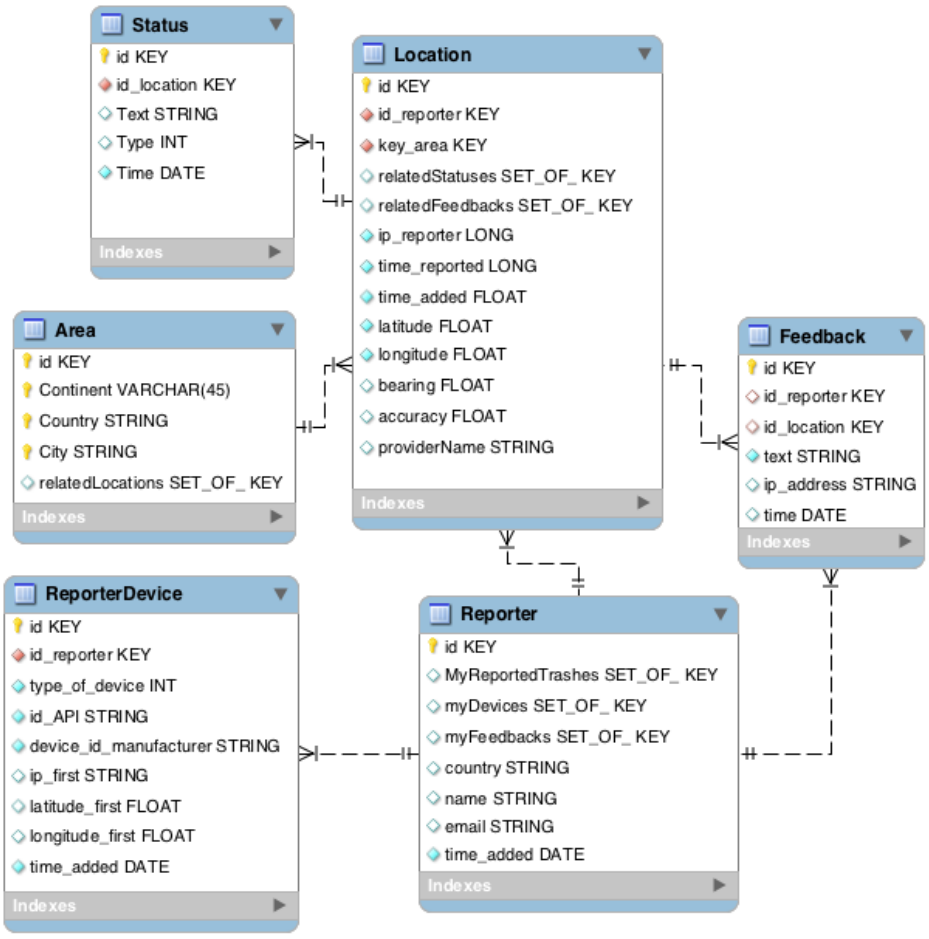

FIGURE 10. Datastore scheme of our App Engine application

Reporter

Reporter includes all entity instances with information about registered reporter. Basic information is stored here about registration UTC time in attribute *time_added*, other attributes such as name, e-mail and country are filled just after online registration of reporter. In attribute *MyReportedTrashes* all location keys reported by this reporter are stored, list of reporters devices *keys* are saved in Set *myDevices* and if reporter leaves a feedback on some other location feedback *key* will be saved in Set *myFeedbacks*. Unique *datastore key* is generated automatically, when his object is persisted to the *datastore*.

ReporterDevice

Reporter can submit locations from different mobile devices. For each device its unique ID – IMEI or Android ID, type of device (ex. “Android”) and generated *unique API key* are stored. Unique API key can be generated just after saving this object to datastore, because there is a need for its generating *datastore key* of this instance, generated just after saving it to datastore. When reporter submits first location with this device, its latitude, longitude and IP address of reporter are stored. This information will be used for future internal statistics.

Location

Only a registered reporter can add location, so in attribute *id_reporter key* of reporter is referred to. For location all needed data about latitude is stored in attribute *latitude,* longitude in attribute *longitude,* information about bearing, accuracy and provider name in attributes with the same description. Attributes *time_reported* and *time_added* state, when the reporter submits location to server and when it was originally saved in the reporter’s device. Reporter’s IP stored in *ip_reporter* is used just for future internal statistics. In attribute *id_area* object *Area* is referred to, where information name of place is stored on map. Location can get feedback from registered reporter, so all related feedback *keys* is stored in Set *relatedFeedbacks.*

Area

To have a better overview to which continent, country or city locations are related, additional information is stored here. For each city, locations *key* is stored in *relatedLocations*. Every city is placed in some country and continent, thus this information is stored as well Locations latitude and longitude. Location does not need to belong to any city, so it can be blank.

Status

Additional information about *Location* and about the status it currently belongs to is stored. It can get to status SPAM, submitted, cleaned or deleted. If there is any additional information for the current state of location, it is stored in attribute *Text.* In attribute *id_trash* Location *key* to know to what location this

status belongs is kept. *Time* attribute stores the date in UTC format of inserting status.

Feedback

Registered visitors on this site are able to leave feedback for location and here reporter *key* in attribute *id_reporter* as author of this feedback will be stored. Message of this author is stored in attribute *text,* date of feedback in UTC format in attribute *time* and related Location *key* in attribute *id_location*.

4.3 Receiving data from Android mobile device

For communication with Android App API – application programming interface is designed. Via this API it is possible to receive requests to server from App and send responses back to App. Well designed API will be used for further development on other platforms and later will allow other developers to develop applications to be able to communicate with the server.

Access to servlet via URL

To map a servlet to a URL path pattern, the <servlet-mapping> element is used and placed in a project file called *web.xml*. A mapping includes the *<servlet-name>* that matches a servlet declaration, and a *<url-pattern>.*

An example of mapping *locationAdd* servlet in *web.xml* file:

```
<servlet>
    <servlet-name>addLocation</servlet-name>
    <servlet-class>com.okk.servlets.locationAdd</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>addLocation</servlet-name>
  <url-mapping>/addLocation</url-mapping>
</servlet-mapping>
```

Registering new device

Servlet r*eporterRegister* mapped to URL address *"/reporterRegister"* will allow this Android App to register and obtain *unique API key* to authorize itself later

for submitting new locations and retrieving feedbacks. Android App will send via *HTTP Post* self-generated XML.

Here is an example of this request:

```
<?xml version="1.0" encoding="UTF-8" ?>
     <request>
       <id>XXXXXXXXXXXXXXXXXXXXXXXXXXXX</id>
       <device_type>ANDROID</device_type>
     </request>
```

This XML is checked against the project's XSD schema, if it is correct and was not damaged on its way to the server.

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema attributeFormDefault="unqualified"
elementFormDefault="qualified" version="1.0"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="request">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="id" type="xsd:string" />
        <xsd:element name="device_type" type="xsd:string" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

If it is not correct XML response is sent back with information *"xml is in wrong format"* inside tag *<error>.*

In case it is correct given ID is searched in the project's datastore. If this device is already registered the already generated *unique API key* for this device is returned, otherwise a new *Reporter* and *ReporterDevice* is created and made persistent in *datastore*. Then a new *unique API key* is generated and sent in response with no error message.

Example of response:

```
<?xml version="1.0" encoding="UTF-8" ?>
<response>
  <api>915c2ddfce1095f05296c422f4a1971f3cc889e1</api>
  <error></error>
</reponse>
```

API is generated as SHA1 hash from string and contains *unique device ID, unique datastore key of ReporterDevice* and secret string stored on the project's server. The output of this SHA1 hash is 32-length string which is used for authorization of each device.

The whole process of communication with Android App is described in section *3.5 Sending data to server - Obtain unique API key.*

Submitting locations

Servlet *locationAdd* mapped to URL address */locationAdd* will allow this Android App to submit all stored locations. Android App will send via *HTTP Post* self-generated XML with information of location.

Here as an example of this request:

```
<request>
 <id_api>915c2ddfcd1095f05296c422f4a1971f3cc889e1</id_api>
 <loc_provider_name>gps</loc_provider_name>
 <loc_accuracy_in_meters>45.0</loc_accuracy_in_meters>
 <loc_bearing>11.5</loc_bearing>
 <loc_latitude>35.12345</loc_latitude>
 <loc_longitude>115.12345</loc_longitude>
 <loc_time_reported>2415488651315</loc_time_reported>
 <loc_speed>0.0</loc_speed>
</request>
```

This XML is checked against the project's XSD schema, if it is correct and was not damaged on its way to the server.

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema attributeFormDefault="unqualified"
elementFormDefault="qualified" version="1.0"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="request">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element name="id_api" type="xsd:string" />
   <xsd:element name="loc_provider_name" type="xsd:string" />
   <xsd:element name="loc_accuracy_in_meters" type="xsd:float" />
   <xsd:element name="loc_bearing" type="xsd:float" />
   <xsd:element name="loc_latitude" type="xsd:float" />
```

```
    <xsd:element name="loc_longitude" type="xsd:float" />
    <xsd:element name="loc_time_reported" type="xsd:float" />
    <xsd:element name="loc_speed" type="xsd:float" />
   </xsd:sequence>
  </xsd:complexType>
</xsd:element>
</xsd:schema>
```

If it is not correct an XML response is sent back with information "*xml is in wrong format*" inside tag *<error>.*

With correct input data it can be proceeded to check if the sent *API key* is authorized to use this service. If it belongs to already registered reporter, it can be proceeded to next steps, otherwise an XML message is sent back informing about the wrong *API key*. The instructions of the proceedings are listed below as follows from the author's point of view:

1. We check if this reporter already reported location with same Latitude and Longitude. If yes we will not store it again and send back message with info about duplication. I not we proceed to next step.
2. We save new Location object with given data to datastore.
3. Status of fresh inserted location is "submitted", this is saved in Status object into datastore with reference to our location.
4. Last reference, we need to update is Reporter's list of reported locations. So we add *datastore key* of newly inserted Location object to list this list.
5. We can send back XML response to client and inform about success.

Getting geographical address of saved location

For statistical information all reported locations are geolocated. Via service *Google geocode*, for each location info is obtained about locality *(political), administrative areas (Town, name of area in the town), country and postal code.*

The focus is on country and name of the city, if it is provided. This info and also the name of the continent this country belongs to is then stored in the data store object called *Area* with reference to this location.

4.4 Showing stored data on webpage

JSP is used for generating dynamic HTML code enriched with data from App datastore. Inside JSP, stored locations are dynamically generated from App datastore. Main focus is on showing locations inside Google Map, so it takes the majority of user screen as is shown on Figure 11.

FIGURE 11. Google Map with reported locations (Google Maps)

All pins are added to Google Map with help of jQuery and gMap3. Each point has a unique ID and also text which is shown after clicking on it. Info message about selected location is shown on Figure 12.

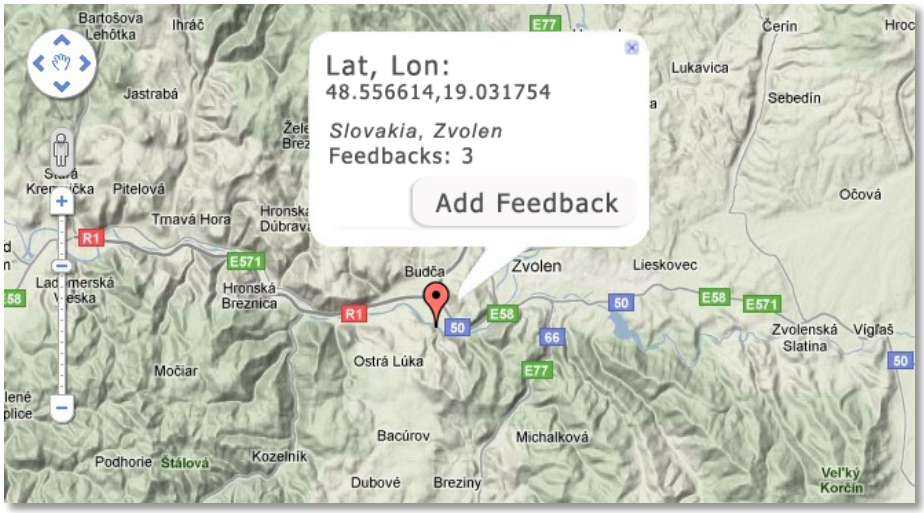

FIGURE 12. Google Map with message on pin

This message contains details of location such as

- latitude and longitude,
- accuracy in meters,
- time of report in current time of user
- number of feedbacks and button **Add feedback**.

Feedbacks and user registration

Via button *"Add feedback"* the developer's own comment on this location can be submitted. Before that the Google account has to be authenticated with.

The user will log in to his/her Gmail account and this application will ask for her/his Name and e-mail address. After this, new *Reporter* is created in the datastore, with the name and e-mail the user provides the developer. A new *ReporterDevice* is also created with *device_id_manufacturer* initialized to SHA1 hash of *unique ID* from *Google account*. Atribute *device_type* is initialized to value *0 – "internet user".* After this procedure the user is able to submit feedback for selected location.

Each feedback is stored in datastore with reference to its author and the location it refers to.

Administrating

Administration layout is shown on Figure 13.

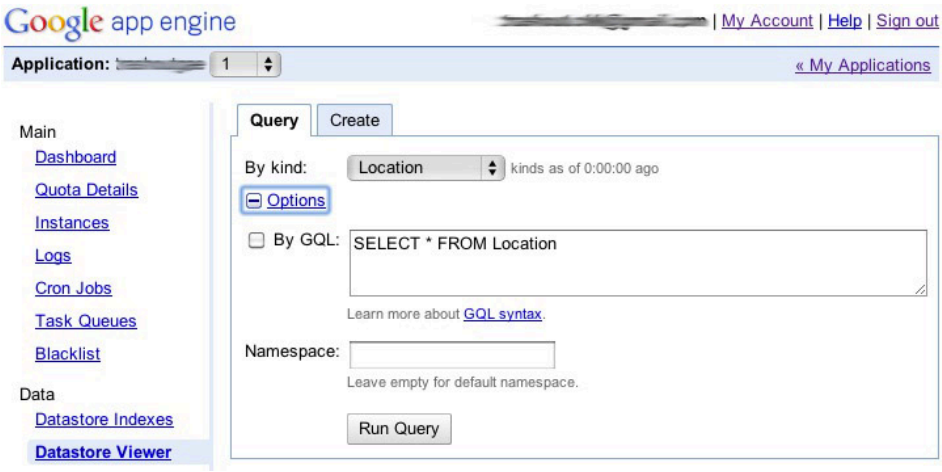

FIGURE 13. Google App Engine administration layout

All systems need an administrator and this one is no exception. All entities stored in datastore can be accessed via the administration control provided by Google App Engine.

All records for each stored entity can be listed and their values can be seen, updated and deleted.

If logged in with the “administrator” account, there is an option to delete each reported location shown on map, or submitted feedback for locations.

5 CONCLUSION

5.1 Results

A web application is developed on top of a cloud system from Google called Google App Engine to ensure fast response and scalability for this service in the future. Currently it is in private beta, so just few people are allowed to join testing our applications. For testing phase we invite people with different backgrounds to ensure that the applications are user friendly and stable. So far the application looks stable.

From the latest response, people are delighted of the project idea and are looking forward to using it or asking us to join private beta testing.

We hope our solution will help to solve this problem of illegal dumps, which is growing year by year.

5.2 Further improvements

Before Applications get from private beta to public beta, we are going to do several steps.

- Make several versions of Android App to use the advantage of new Android OS versions, which brings new possibilities to developers.
- Localize our Android and web application for several language regions in the world. Localized applications have a better chance to become successful and widely spread.
- Add JSON compatibility for requests and response as an alternative to XML.

REFERENCES

Burnette E., 2010, Hello, Android

Cloud computing, Referred to on April 2011, http://en.wikipedia.org/wiki/Cloud_computing

gMap3, Referred to on April 2011, http://www.gmap3.net

Google Android documentation, Referred to on April 2011, http://developer.android.com

Google App Engine, Referred to on April 2011, http://code.google.com/appengine

Google Maps API, Referred to on April 2011, http://code.google.com/apis/maps

GPS - Global Positioning System, Referred to on April 2011, http://en.wikipedia.org/wiki/Global_Positioning_System

HTTP Servlet, Referred to on April 2011, http://en.wikipedia.org/wiki/Java_Servlet

Java, Referred to on April 2011, http://www.java.com

JavaScript, Referred to on April 2011, http://en.wikipedia.org/wiki/JavaScript

Meier R., 2008, Android Application Development

Murphy Mark L., 2008, The Busy Coder's Guide to Android Development

Sanderson D., 2010, Programming Google App Engine

SQLite, Referred to on April 2011, http://www.sqlite.org

Walking, Referred to on April 2011, http://en.wikipedia.org/wiki/Walking