DEGREE PROJECT IN COMMUNICATION SYSTEMS, FIRST LEVEL
STOCKHOLM, SWEDEN 2015

# Location based sports/fitness app

*Android app for smartphones*

JIMMY QUARESMINI

KTH ROYAL INSTITUTE OF TECHNOLOGY
*INFORMATION AND COMMUNICATION TECHNOLOGY*

# Location based sports/fitness app

## *Android app for smartphones*

Jimmy Quaresmini

2015-02-18

Bachelor's Thesis

Examiner and Academic adviser
Gerald Q. Maguire Jr.

KTH Royal Institute of Technology
School of Information and Communication Technology (ICT)
Department of Communication Systems
SE-100 44 Stockholm, Sweden

## Abstract

The City of Stockholm's budget for sports is 1.6% of the total city budget in 2014. About 60,000, licensed (age 15 and up, about 50%) and younger actively played soccer in Stockholm in 2005. That is roughly 24% of the total number of participants in sports activities in Stockholm's district. There is a need for a location-based application (commonly abbreviated "app") to help people spontaneously meet for different sports activities.

The app developed and analyzed in this thesis will address this need and assist potential participants in organizing sports activities, deciding to participate in these activities, make friends, etc. Not only large team sports will be considered, but sports done in pairs such as tennis and other sports as well that you do with others. Consider a person who has moved to a new city and wants to play football (soccer), but does not know people in this city. This app can help this person find others who would like to play in a quick and easy way by looking at a map and seeing where others would like to participate in this sport, for example at a particular soccer field, using location information concerning the potential participants and this specific field. Apps that use location information have become very common with the widespread use of smartphones. Such an app provides a location-based service.

This thesis describes the design and implementation of an Android app with a location-based service and how to set up this app, what technology is needed to get the needed location information, and the programming language used. In addition, the thesis considers users' needs and how the app caters to those needs. An analysis will be done of how well this app communicates with a database server as the number of users increases, scales. Performance and load on both the server and app will be considered. The performance will be analyzed to see how well it matches the users' expectations.

The app developed in this thesis will be for the Android platform and Apple's iOS (but the focus in the thesis is on the Android-version). This app will communicate with a database server running a Linux OS, an Apache HTTP server, a MySQL database, and using a PHP programming web infrastructure (such a setup of services is commonly called by the acronym LAMP). The app will connect to Facebook and Twitter to exchange information (however, that is outside the scope of this thesis).

### Keywords

Location-based, application, app, service, Android, scales, database, server, LAMP, sports

## Sammanfattning

Stockholms stads budget för sport är 1,6% av den totala budgeten 2014. Ungefär 60 000 licensierade (15 år och uppåt, 50 %) och yngre aktiva fotbollsspelare fanns i Stockholm år 2005. Det motsvarar ungefär 24 % av dem som utövade sport det året i Stockholms distrikt. Det finns ett behov av en platsbaserad applikation (vanligen förkortat ”app”) som hjälper människor att spontant träffas för att utöva olika sporter man utför tillsammans.

Appen som utvecklas och analyseras i detta examensarbete försöker tillgodose detta behov och hjälper potentiella deltagare att organisera sportaktiviteter, besluta att delta i dessa aktiviteter, skaffa nya vänner m.m. Inte bara stora lagsporter, men också sporter man utför i par som tennis och andra sporter man utför tillsammans med andra kommer beaktas. Betänk en person som nyss har flyttat till en ny stad och vill spela fotboll, men inte känner några i denna stad. Denna app kan hjälpa den här personen hitta andra som vill sporta på ett enkelt och snabbt sätt genom att titta på en karta och se var andra vill utöva denna sport, t.ex. på en viss fotbollsplan, med användning av platsinformation om potentiella deltagare och denna specifika plan. Appar som använder platsinformation har blivit väldigt vanliga med det utbredda användandet av smarta mobiltelefoner. En sådan här app erbjuder en platsbaserad tjänst.

Denna rapport beskriver designen och implementationen av en Android app med en platsbaserad tjänst och hur man sätter upp en sådan här app, vilka teknologier som behövs för att få den platsinformation som behövs och det programmeringsspråk som används. Därutöver kommer rapporten överväga användares behov och hur appen tillgodoser dessa behov. En analys kommer genomföras av hur denna app kommunicerar med en databas server när antalet användare ökar, när den skalar. Prestanda och belastning av både appen och servern kommer tas med i beräkningen. Prestandan kommer analyseras för att se hur väl det motsvarar användares förväntningar.

Appen som utvecklas i detta examensarbete kommer vara för Android och Apples iOS (men fokuset i detta examensarbete kommer vara på Android versionen). Denna app kommer kommunicera med en databas server som kör ett Linux OS, har en Apache http (webb) server, en MySQL databas och som använder en PHP programmerings-infrastrutur (en sådan uppsättning tjänster kallas vanligen LAMP, en akronym). Appen kommer ansluta till Facebook och Twitter för att utbyta information (men det ligger dock utanför ramen för detta examensarbete).

### Nyckelord

Platsbaserad, application, app, tjänst, Android, skalar, databas, server, LAMP, sport

## Acknowledgments

I would like to thank Professor Gerald Q. Maguire Jr. for being my examiner and academic adviser and accepting to help me with a short notice.

I would also like to thank Chima Okechukwu and digiArts Entertainment AB for providing this thesis project.

Stockholm, February 2015
Jimmy Quaresmini

**Table of contents**

## List of Figures

## List of Tables

## List of acronyms and abbreviations

| | |
|---|---|
| apk | Android Application Package |
| API | Application Programming Interface |
| app | Application |
| GLONASS | GLObalnya NAvigatsionnaya Sputnikovaya Sistema |
| GNSS | Global Navigation Satellite System |
| GPS | Global Positioning System |
| GSM | Global System for Mobile communications |
| JAR | Java ARchive |
| LAMP | Linux OS, Apache HTTP server, MySQL database, PHP programming |
| LBS | Location-Based Services |
| OS | Operating System |
| SDK | Software Development Kit |
| SSID | Service Set IDentifier |
| RSSI | Received Signal Strength Indication |
| WLAN | Wireless Local Area Network |
| W3C | World Wide Web Consortium |
| WWW | World Wide Web |
| XML | Extensible Markup Language |

# 1 Introduction

Location-Based Services (LBSs) have become very common due to the increase in the number of smart mobile phones (smartphones). LBSs use the coordinates of a device or a place in some way, for example to give tips about something fun to do close by or even as a form of reality based game.

Applications (commonly known and abbreviated as "apps") for smartphones today (in 2014) exist for all sorts of needs. These apps are created by companies, organizations, institutions, and smaller developers.

This thesis will describe the design, development, and evaluation of an app that uses LBS to help people meet to engage in some group sporting activity. This chapter describes the background, problem definition, purpose, goals, methodology, and the delimitations of this thesis, as well as presents the structure of this report.

## 1.1 Background

The City of Stockholm's budget for sports is 1.6% of the total city budget[1]. About 60,000, licensed (age 15 and up, constituting about 50%) and younger actively played soccer in Stockholm in 2005 (see [2]). That is roughly 24% of the total number of participants in sports activities in Stockholm's district. There is a need for a location-based smartphone app to help people spontaneously meet for different sports activities. This app will assist potential participants in organizing sports activities, deciding to participate in these activities, make friends, etc.

Moreover, people may want to participate in team sports (for example, (American) football, ice hockey, or soccer), but do not know who else (near where they live or work) would like to participate or know enough people to form team(s). The same may be true for other sports, such as tennis, that also requires a partner. This could mean that an individual would be unable to participate in a sport that they enjoy, hence they might not get as much exercise as they would like and need. This could negatively impact the person's health. In addition, not knowing people may lead to feelings of isolation, which could be detrimental to the individual's mental health. Such an application could help address these needs. This will be brought up again in Sections 1.3 and 6.4.

## 1.2 Problem definition

The company digiArts Entertainment AB[3] wants to create a smartphone app to allow people to easily meet and enjoy different sports based on location information. In most cases the location information the app uses is the user's coordinates, often accessed through the smartphone's GPS, but also general information about the place to do sports, such as weather for example can be shown. In this thesis we will focus on the design and development of this smartphone app using the Android Operating System (OS) that uses location information of both potential participants and of available places where one can engage in some kind of sport(s).

Consider a person who has moved to a new city, say Stockholm, Sweden, and wants to play soccer, but does not know (many) people in this city. There are about 330[4] different pitches outside and indoor halls where soccer players can play soccer in Stockholm (matches and/or training) as of 2014. The proposed app can help the user find others who would like to play soccer in these places in a quick and easy manner by using the location of these places and the location of the users as well. In addition, the user will get information about the place to play, whether there is a fee, etc.

Another problem that needs to be addressed is how to store information about users, sports facilities, etc. and to access this information in a way that the user finds acceptable. A web server

will be used to provide both the storage and major computations to support the app. The app will communicate using a secure means with this web server. We have chosen to use an Apache web server running on the Linux OS together with a MySQL database and scripts written in PHP. This type of setup is commonly known by the acronym LAMP.

In addition to the app's user interface enabling the user to indicate if they are interested in participating in a particular sport, the interface must also provide a way for the user to register with the service and to "log in" (some means of authenticating themselves and getting authorization to securely exchange information with the web server).

## 1.3 Purpose

My purpose for selecting this thesis project is to learn how to develop an Android app for smartphones, learn how such an app can communicate with a database on a web server, analyze how the app and server perform, and how this setup will scale. Additionally, I would like to understand what such an app could mean for people who could now meet new people and more easily engage in different sports, such as team sports.

The company's purpose is to make an app that many people will want to use and make money from the app indirectly from sponsors and others. The app will be freely available to users. The company also hopes people will find this app to be useful and fun.

How this app may be beneficial to people in general was brought up briefly in Section 1.1. Additionally, there is a need to consider the user's privacy, personal integrity, and other issues when providing location information to a server and when providing other personal information. Furthermore there are various societal issues and sustainability issues that need to be considered when creating an LBS. Other and similar aspects will also be brought up in Section 6.4.

This thesis may help other app developers and others interested in how to use location information in a smartphone app and those who want to understand the performance of such an app (together with LAMP).

## 1.4 Goals

To develop a mobile phone app that uses location and other information to dynamically organize *ad hoc* sports events requires acquiring location information, some personal information about the potential participants, and information about the available sites where the event could take place. The app also requires support from a server that can keep track of a user's preferences, the interests of all potential participants in a particular type of event, the number of participants needed in each type of event, the potential sites for each type of event, etc. In addition to designing and implementing this app and its supporting infrastructure, there is a need to analyze how the app performs and how it can be of benefit to both individual users and to society. Further, by applying my theoretical knowledge about general programming I will gain practical experience in Android programming in "the real world"[*]. The sub-goals of this project are:

1. Learning to acquire and use location information,
2. Learn how to design, develop, and deploy an Android app,
3. Learn how to best communicate between the app and the server infrastructure,

[*] This type of application programming and its environment are actually new to me, so a personal goal is to learn this skill – which is currently highly valuable in industry.

4. Understand how the web server and database works (i.e., how to provide a suitable server infrastructure),
5. Develop an app digiArts Entertainment AB is satisfied with,
6. Analyze the performance of app and the corresponding server infrastructure, and
7. Consider the benefits of the app to individuals and society.

## 1.5 Research Methodology

We will test the app both in an emulator (that simulates a smartphone on a computer) and on real smartphones and thus gather data about the app's performance with the web server. Based upon an analysis of this data, along with preferences both of myself and the company, I will consider what possible changes could/should be made in the app or infrastructure and I will identify conclusions that can be drawn.

The app will not be written from scratch. Instead, existing code for an app not released yet on Android will be developed, analyzed, and modified as needed. The existing app called "Kopplar" was developed by DigiArts Entertainment AB for the Android and iOS platforms. Actually, DigiArts hired a company in India I do not have the name of to do the programming of the app. Further details about this app can be found in Section 4.2. DigiArts Entertainment AB has not looked at this Android version of the app and I do not know beforehand what is needed to be done with it. The app version studied in this thesis will be the Android version.

If errors are found, they will be corrected. Warnings will be corrected if needed. I will not have access to those that developed this app before me (the company in India) and have to rely on myself to understand what the code does.

## 1.6 Delimitations

Due to the limited time available for this thesis project, the project will *not* analyze how the app affects people in real life, using polls or similar means of studying user opinions. Additionally, due to the limited time available for this project a complete website for the app will not be constructed (as was thought about beforehand). As mentioned in Section 1.5, this app is for Android rather than for Apple's iOS.

## 1.7 Structure of the thesis

In Chapter 2 relevant background information and previous works will be presented. Chapter 3 brings up the details about the methodology and methods used. Chapter 4 details what I did, phones and emulator used, the setup and so on. Chapter 5 presents the results and an analysis of these results. Chapter 6 presents the conclusions, limitations, future work and reflections on the possible impacts of the app on society.

# 2 Background

This chapter will present the background information about location-based apps. The background information is divided into 2 parts: types of localization and LBSs, and developing Android apps with a database. The chapter will also briefly describe six related works.

## 2.1 Types of Localization and Location-Based Services (LBS)

Localization is the process by which a mobile phone's location is estimated. This section presents some of the types of localization currently used with mobile phones.

Location-Based Services (LBS) are common today, mostly in smartphones (in the form of apps), but also in some web sites and applications for computers (for example, Windows 8 utilizes location in a number of applications). An LBS gathers information (the coordinates) of a user's device in different ways, depending on what kind of device it is and what hardware is in that device. For example, today most smartphones support Global Positioning System (GPS) (see section 2.1.1).

### 2.1.1 GPS

GPS is the most common form of localization in smartphones today. GPS is an American navigation system. It is the most famous kind of Global Navigation Satellite System (GNSS) [5]. GLObalnya NAvigatsionnaya Sputnikovaya Sistema (GLONASS) is a Russian GNSS. GLONASS covers northern latitudes (such as Sweden) better than GPS does. Both GPS and GLONASS currently support global positioning with high precision [6]. The Google Nexus 5 smartphone (shown in Figure 2-1) supports GLONASS, GPS, and A-GPS (described further below). Europe's GNSS is called Galileo and China's is called COMPASS or BeiDou-2, but as of yet - neither is widely deployed or used by large numbers of users.


**Figure 2-1: Google Nexus 5 smartphone**

GPS uses multilateration, a technique to determine the location based on knowledge about other locations that are already known. By solving a set of simultaneous linear equations, the GPS receiver can use the information transmitted from three or four satellites to determine its position in three dimensions and the current time. Smartphones today (2014) often implement Assisted-GPS (A-GPS) to reduce the time it takes to determine the receiver's initial location.

### 2.1.2 Cell-ID

If a device lacks GPS, another method to learn the device's location (coordinates - longitude and latitude) is to listen for a cellular base station's ID, then look in a database to see what the longitude and latitude of this base station is. A cellular phone that can hear multiple base stations can use this approach repeatedly to better estimate its location. This method is often called "Cell-ID". For further details see [7].

### 2.1.3 Wireless LAN localization

This approach is similar to using a cellular base station's ID, but instead uses the Service Set IDentifier (SSID) of a Wireless Local Area Network (WLAN) access point and a database of locations of known access points. These access points' locations are often provided by users themselves (so-called "crowdsourcing"), see for example WiGLE.net[8] and Navizon[9].

### 2.1.4 GSM Time advance offset

Global System for Mobile communications (GSM) base stations transmits information to tell the mobile device how to offset its transmissions in time so that they arrive at the correct time slot at the base station. This can be used to estimate the distance from a given base station. Again the method can be used multiple times to better estimate the location of the mobile device. For further details see publications such as [10].

### 2.1.5 Signal strength

Many researchers have utilized signal strength measurements, see for example [11]. Today, numerous commercial systems uses the Received Signal Strength Indication (RSSI) for cellular or WLAN interfaces to estimate the distance from several base stations or access points to estimate their location. For further details of combining this method with inertial measurements see [12].

### 2.1.6 Using mobile phones' hardware to do "dead reckoning"

As shown in Section 2.3.4, a mobile phone's own hardware can be used to estimate the location by using so-called "dead reckoning". Dead reckoning estimates a location based upon tracking movements, i.e., the direction and speed moved, from a given starting location.

### 2.1.7 User check-in

One more method to get the location, which has become quite common is called "check-in". In this method, a user looks at a map and choses his or her own location on it, from which the coordinates will be provided to an app.

Additionally, other technical means to gather location, especially indoors, exist, but they will not be mentioned here.

### 2.1.8 LBS-applications

There are many types of applications of LBS. Finding something close by that one is interested in, such as a specific type of store or restaurant. Finding people by viewing a map, advertisement, or notifications of different types. Examples of LBS apps include applications for emergencies, criminal investigations, and games using your location. There are many more examples of such apps.

The author's perception is that a location-based sports/fitness app that combines knowledge of locations of places to do sports and facilitates meeting new people has not been exploited in many apps available in the market; which, if true, indicates that the app may serve a need or fill a gap.

## 2.2 Developing Android apps with a database

Android provides developers with a Software Development Kit (SDK) that one can download for free at the Android website*[13].

### 2.2.1 Android OS

The Android OS is an open source project[14]. It comes in many different versions and have been in development since 23 September 2008[15]. The latest version, as of November 2014, is Android 5.0. Names of Android OS versions come from the world of candy and sweets, with Android 5.0 called Android L or "Lollipop". Version 4.4 was called KitKat. …

The Android OS has a nice look and allows installation of many different apps. These apps are installed from files whose names end with ".apk", which stands for Android Application Package.

The Android OS is not only used for smartphones, but also for tablets and wearable devices and more. Android 4.4 is a version specifically for wearable devices.

### 2.2.2 Developing apps for Android

The programming languages used for developing Android apps are Java and Extensible Markup Language (XML). XML is used by Android to create the graphical interface (i.e., the layout), while Java is the foundation for the app's logic. Android uses a specialized version of XML with its own terms. Java source-files are packaged as Java Archive (JAR)-files. The Eclipse integrated programming environment comes with the Android SDK.

To test the apps, one may use an emulator that comes with Eclipse. It emulates many different smartphones, but also other devices such as wearable devices for example. An example of the user interface when loading the emulator is shown in Figure 2-2. The emulator will be discussed in more detail in Chapter 4.

* http://developer.android.com/index.html

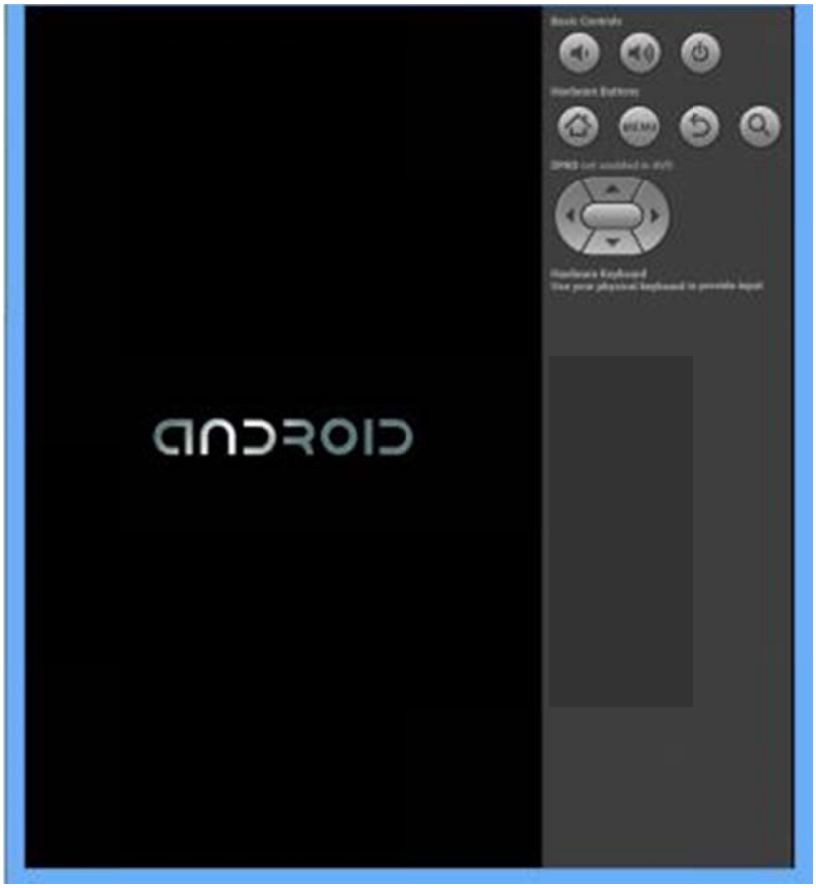



**Figure 2-2: Emulator loading Android OS version 4.4.2**

### 2.2.3 LAMP

LAMP is an acronym for a widely utilized combination of software that is used as a foundation to realize a web service. LAMP often consists of the Linux OS, an Apache web server, a MySQL database, and PHP scripting programming language, hence the acronym LAMP. As mentioned earlier, the app developed and studied in this thesis will use LAMP.

However, a LAMP web service solution can use other solutions for each of the 4 parts that LAMP consists of, so the web server rather than an Apache Web server could be NginX (pronounced "engine X"), Microsoft's IIS, Lighttpd, Tomcat, or Cherokee. The database could instead of MySQL use MariaDB or Drizzle, while the scripting language could be Python or Perl instead of PHP. Likewise, the OS could be Microsoft's Windows, Apple's Mac OS, … instead of Linux. The acronyms changes to reflect what is used, so there is WAMP, WIMP, MAMP, etc.

### 2.2.4 Android App Localization

To do localization in Android apps is pretty simple because Android itself provides good framework Application Programming Interface (API) that help a developer with this task. To do localization in Android a developer acquires a handle for a new instance of a "LocationManager". This LocationManager gives a developer access to different methods and information, which in turn allows an app to learn the location of the device.

Because the device's position is a sensitive personal matter, an app needs the user's permission to access the location. If this permission is granted, the app will be able to gather location information.

Localization in Android can be done in two ways (latest known location and registration) and used in one way (intent) as described here. An app may get the latest known location from some location provider (using the LocationManager). The app may also want updates from time to time about the device's current location. It receives these updates by registering for updates from a location provider. Localization can be used to trigger actions at a specified position. To do this the developer needs to register a so-called "intent". Information about all of these operations can be found at Android's "Location and Maps" at [16].

Another alternative to Android's localization APIs is Google Maps' Android API (see [17]), which according to some developers is more accurate and simpler to use[18]. Both APIs work well though and it is up to each developer to chose which API they want to use.

## 2.3 Related works

This section summarizes some related works on LBS, characterizing the accuracy of LBS, the performance of LAMP, and making performance measurements and tuning Android apps.

### 2.3.1 Building LBS Based On Social Network API: An Example of a Social Check-in App

This app uses the location of people to provide a user check-in function. Users use the Facebook API to gather information about the user (provided the user agrees to share this information). This app uses the Facebook Query Language (FQL) similar to SQL queries, to get the information needed for a user to perform a social check-in. For further details see [19].

### 2.3.2 Some experiments with the performance of LAMP architecture

In 2005, Ramana and Prabhakar[20] tested the performance of the LAMP architecture, specifically one using MySQL and PHP, in different setups. They measured CPU performance combined with connections per second and how many bytes are transferred over a long time period. They also compared programs written in PHP and the same program written in C for different calculations and tasks. Different CPU speeds were used for the execution of the software. The other operations were done on a single Pentium4 CPU.

### 2.3.3 Characterizing Mobile Open APIs in Smartphone Apps

Li Zhang, et al. studied many different mobile open APIs and their performances in smartphone apps. By "open APIs" they mean parts of apps that perform specific tasks, such as the Facebook login dialog[21] using the OAuth protocol 2.0 [22, 23]. They measure or gather info about CPU usage, latency, power usage and network traffic for different APIs. They also identify some tools to perform measurements and collect some of this information.

### 2.3.4 Investigation of location capabilities of four different smartphones for LBS navigation applications

Retscher and Hecht[24] studied the hardware (accelerometer, magnetometer and the digital compass) in four smartphones that makes it possible to gather the location data (coordinates - latitude and longitude) and also using GPS. They test 4 different smartphones in a city-environment

to see how they perform, both inside and outside, compared to known reference values. Provided that known locations were the starting point, they showed that using the accelerometer and the digital compass or magnetometer could be used to get the location data (often called "geolocation").

### 2.3.5 Tests of smartphone localization accuracy using W3C API and Cell-Id

Grzegorz Sabak [7] tested how well two common non-GPS techniques, World Wide Web Consortium (W3C) API and Cell-ID, worked for localization purposes and compared with GPS. The W3C API provides geolocation through web browsers using different techniques, some of which were mentioned earlier.

### 2.3.6 Selective cloaking: Need-to-know for location-based apps

In [25], Benjamin Henne, et al. describe how to keep the user's location private to apps by default, unless the user explicitly give an app permission to find and use the location of the mobile device. They introduce a method that obfuscates the location. Otherwise, many apps can access the location whether they need to have it or not and the user may not want this. What is especially relevant to my work is that they describe how to get the device's location in Android 4 (Android Ice Cream Sandwich).

## 2.4 Summary

Mobile phones are certainly not new; they have been around for decades. Smartphones and tablets with the capabilities of today have only been around for 4-5 years (since the iPad was launched in 2010 [26]) and about 6 years for the Android OS (since 2008 as mentioned above). Still, despite this relatively "short" time, there are already many apps and APIs for both smartphones and tablets and many apps have been developed.

Localization has been around a very long time. Using satellites and base stations for localizations is not new either, but has been popularized by the use of smartphones and smartphone apps. Additionally, localization has spread to computers, tablets, and web applications.

Table 2-1 summarizes the related works, highlight the advantages and disadvantages brought up in these works.

**Table 2-1: Advantages and disadvantages of relevant works**

| Relevant Work | Advantages | Disadvantages |
|---|---|---|
| **Building Location-Based Service Based On Social...** | To the point, clear, uses pictures. Good info about Facebook. | None |
| **Some experiments with the performance of LAMP architecture** | Fairly well thought through tests. Good they tested common setups like LAMP and WAMP. The tests of PHP vs C was perhaps most interesting and really clear. | Bit unclear how to interpret the result-graphs. |
| **Characterizing Mobile Open APIs in Smartphone Apps** | Thorough testing with many apps and tests. Pretty good info about APIs. | Some unclear terms, but not many. Few pictures. |
| **Investigation of location capabilities of four different smartphones...** | Interesting introduction about different localization techniques. | Some of the results seemed a bit unclear. |
| **Tests of smartphone localization accuracy using W3C API and Cell-ID** | Pretty clear tests. Good that they brought up such common localization techniques aside from GPS. | Fairly complicated formulas in the Data model. |
| **Selective cloaking: Need-to-know for location-based apps** | Fairly useful app they did these days. Several pictures. | None |

# 3 Methodology

The purpose of this chapter is to provide an overview of the research method used in this thesis. Section 3.1 describes the research process. Section 3.2 focuses on the data collection techniques used for this research. Section 3.3 describes the experimental design. Section 3.4 explains the techniques used to evaluate the reliability and validity of the data collected. Section 3.5 describes the method used for the data analysis. Finally, Section 3.6 describes the framework selected to evaluate the capacity of the web server using the Android app.

## 3.1 Research Process

1) Develop an Android app in Eclipse. 2) Set up and start an emulator and get access to smartphone(s) that use Android. 3) Install the app on the emulator and smartphone(s) to be used. 4) Run the app and do an HTTP POST request at the same time on mobile device(s) and the emulator while measuring the time simultaneously. Time measurement starts right before the POST request is done (before the connection is set up actually). This is step 1 in Figure 3-1. 5) Process the POST request at the web server (using PHP files). This is step 2 in Figure 3-1. 6) Receive a response from the web server and stop the time measurement. Take note of this time. This is step 3 in Figure 3-1. Repeat this process the desired number of times.

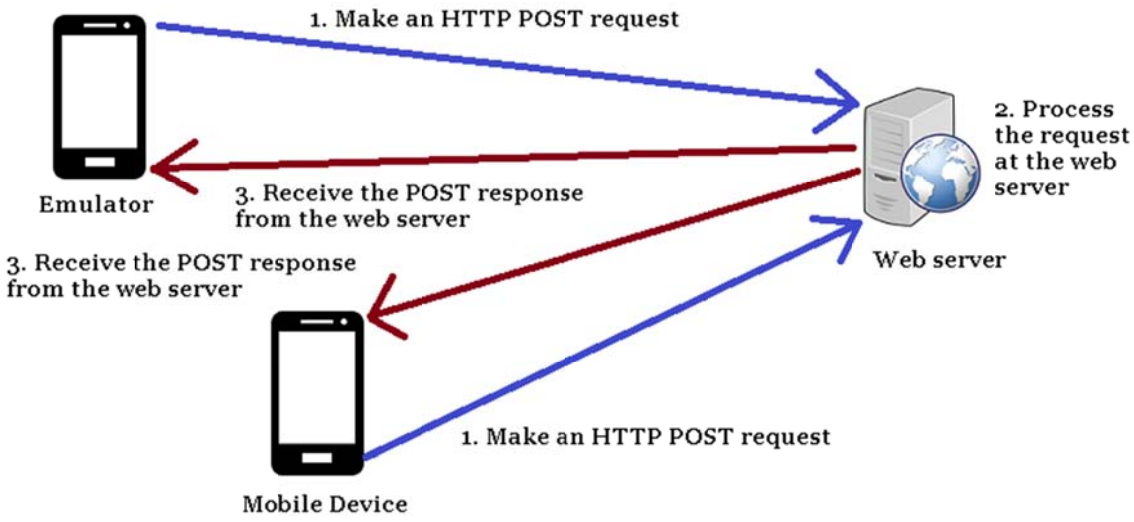



**Figure 3-1:** **The 3 steps of making the HTTP POST request in the app that I measure the time of. Measurements start right before step 1, and ends right after step 3.**

HTTP POST requests are sent to the web server to simulate the behavior of the app when a user uses it. A certain number of POST requests with a chosen delay will be equivalent of 1 "user". Several users will then be mimicked this way and response times recorded to evaluate how the web server can handle different loads of traffic.

## 3.2 Data Collection

In preparation for future testing of the app and web server with a number of users, this section examines how many users might need to be involved in such testing to have a representative test population. Note that testing of this app with more actual users than 2 was postponed and remains for future work.

### 3.2.1 Sampling

An emulator of a mobile phone, 3 real mobile phones (with emulated users in the app) and a python-based load testing tool will be used to test the below number of users (target population).

### 3.2.2 Sample Size

Based on the target population below (see 3.2.3 below), and estimations of how many Android phones there are in Stockholm, Sweden (the greater Stockholm area, not only the city of Stockholm) and other estimations and assumptions (see 3.2.3), the sample size for 95% confidence is **96**.

### 3.2.3 Target Population

People in Stockholm interested in doing sports with others, wanting to use this app to do sports together and that have Android smartphones with API 14 (or later) are the target population. To estimate the target population I will do some estimations and calculations and some guesses. It is not easy to get precise numbers on how many have Android smartphones, or a certain API level on those Android smartphones.

Measuring number of smartphones which use Android or are iPhones can be done in different ways. Numbers of owners or the amounts of traffic from different smartphones are 2 different ways to measure this. Another is to look at sales. It becomes a bit confusing so one has to do estimations.

An international comparison[27] put the number of Android owners in Sweden at 55% in 2013, vs 35% iPhone owners, but it did not say out of how many smartphones. Swedish statistics from sales showed that about 9 million smartphones were sold in 3 years up to 2013 in Sweden. In February of 2013, 73% of Sweden's population used a smartphone according to [28]. Stockholm's population in 2014 was 2,192,000 people (again, counting Greater Stockholm). 73% of 2.192 million is 1,600,160 people (number of people with smartphones). Again, 55% of smartphones in 2013 supposedly were Android, which would mean about 880,088 (55% of 1,600,160) Android smartphones. According to [29], Android users having Android 4.0 or later, is 50% on the 1st of April, but since it is a little "old", I believe it is 75% in Sweden today. This is not an exact number as such numbers are very difficult to obtain.

If you go by the 75% figure for Android smartphone users having Android 4.0 (lowest API level the app in this thesis supports) or higher though, you land at the figure 660,066. Of those, a guess is that maybe half do sports or such physical activities that are equivalent to sports. Further, another guess is that between 12.5 % to 25% of these 660,066 people could imagine trying out this app now when it is new and unknown. Again, this is only a guess. It could be more and it could be less. The number between 12.5 % and 25 % of 660.066 people is about 123,762 people.

To conclude: target population is **123,762** people in the Greater Stockholm area.

## 3.3 Experimental design/Planned Measurements

The testing that was actually done in the course of this thesis project is described in this section. The planned measurements are described, while the results of the testing and the results of the actual measurements are given in Chapter 4.

### 3.3.1 Test environment and test model

The tests will be conducted both on a laptop with a smartphone emulator and with real Android smartphones running the app on them and communicating with a webserver. The webserver contains several PHP-files that are used in conjunction with the java-code of the Android app. Further, the webserver has actually a domain-name reserved for the app: kopplar.com. For software- and hardware-requirements, see below.

As mentioned above in Section 3.1, HTTP POST-requests with some delays will be used to model user behavior and measure response times from the server.

### 3.3.2 Hardware/Software to be used

The laptop that has the emulator is an HP Pavilion Entertainment PC, DV-6. It has 4 GB RAM, an AMD dual core RM-75 CPU running at 2.2 GHz. It has a 500 GB hard drive running at 5,500 RPM. The OS is Windows 8.1, 64-bit.

The emulator used is a Genymotion (version 2.3.1). This emulator was chosen over the standard Android emulator that comes with Eclipse, because the standard emulator is terribly slow, especially on this laptop. The best ("fastest") virtual device (simulated smartphone) on the standard emulator took 15 minutes to start up. Almost any virtual device of those that can be “installed” (which means here downloaded from the cloud to be used as virtual devices) in Genymotion starts within about 3 minutes. According to some, Genymotion is even faster than a smartphone!

The Android version used on the emulator is 4.3, Jelly Bean, API level 18. This version was chosen because the app was developed with this Android version as the target version, but also because later versions of Android did not work well with the Genymotion emulator when using Google Apps (such as Google Maps for instance).

The smartphone that was emulated is an HTC Evo with 1 processor, 1 GB RAM and a 720x1280, 320 dpi (dots per inch) screen (all of which I could chose).

Smartphones to be used are a Samsung Galaxy Young 2, a Samsung Galaxy S5 and a Samsung Galaxy S4. All 3 use Android 4.4.2 (API 19), which is not a problem (I know that after having tested the app on them). S4 has a 1.9 GHz CPU and S5 has a 2.5 GHz CPU, both with 4 cores. These smartphones were meant to be used the most initially. They used 4G communication, the S5 using the S4’s mobile internet. Most smartphone tests were carried out on the Samsung Galaxy Young 2 though. It has a 1 GHz CPU (single core) uses WiFi instead. That could explain why it showed somewhat better results surprisingly (as seen in Chapter 5).

The web server is a shared server hosted by the company HostGator paid for by DigiArts Entertainment AB. The OS is Linux CentOS Enterprise 6.6 x86. It runs on a 32 Core AMD Opteron 6376 CPU with 64 or 32 GB RAM. It runs the Apache 2.2.26 web server with MySQL 5.5.40-36.1, (in phpmyadmin it is called "libmysql" 5.0.96) database provider and PHP 5.4.35.

The app is called Kopplar and is written in Java and XML together using Eclipse 4.2.1 (part of "Android Developer Tools"). For “heavy” (over 100 users) load testing Python-based “Locust” will be used. Its results will also be compared to results from the emulator as well as the smartphones.

## 3.4 Assessing reliability and validity of the data collected

This section addresses the question of how I ensured that the data collected would be reliable and valid.

### 3.4.1 Reliability

Ideally, one would have had access to as many smartphones as needed to test how well the app scaled (say 96 smartphones). However, smartphones cost a lot of money so that was not feasible here. Hence the simulation of users.

One could also discuss what normal user behavior of the app would be. Likely, a user would do more than simply log in, check the home page and log out (which is what I will do to simulate one user). This is the minimal user behavior of the app, which I felt enough to test with. See Section 4.2 for what can be done with the app. See further Section 5.2 for more on reliability.

### 3.4.2 Validity

Values from both real smartphones and an emulator will be used, and they will be gathered programmatically (not by humans). Given that smartphones were also used, “real” values were used of the app “in action”, not just on an emulated smartphone.

Time measurements in both the app itself as well as in Locust are exact, but may be susceptible to activity on the server, errors etc. that may not be immediately obvious. Several tests will be done to try to get results that are valid and reliable (using the average of these tests). For more, see Section 5.3.

## 3.5 Planned Data Analysis

This section presents my planned data analysis. The discussion is divided into two parts: the data analysis techniques and the software tools that I planned to use.

### 3.5.1 Data Analysis Technique

Several function calls (to the same function, one I made) will be done. The function calls will each contain startups of threads (5) that will make HTTP POST requests to the web server. Timing of these requests will be done inside the treads. These calls will be done at the same time (using a timer in the app) on the emulator and a smartphone. On one occasion 2 smartphones will be used with the emulator. The time will be measured from right before the POST requests are made to when the responses from the server are received and read. I did these tests both with and without reading the responses from the server, but most were read.

These test results from the emulator and smartphone will be used to calculate an average of the 5 tests I’ll do per number of users I test with (100-500 in the case of the emulator and smartphones).

Locust (mentioned below) will also use POST-requests. The results from Locust will then, like above, be used to calculate an average from 4 of its statistics: average (of the POST requests), fail rate, number of requests and RPS (Requests Per Second). Like above, 5 tests will be done per number of users, but Locust will simulate 100-700 simultaneous users.

### 3.5.2 Software Tools

Python-based load testing tool Locust (in Python called “locustio”) will be used to simulate number of users from 25 to 800.

## 3.6 Evaluation framework

My plan is to make programming changes to the app's java code and measure how long some functions take to finish. This will then be presented to the user and can thus be noted. It will be tried on the emulator as well as on 3 smartphones (but not all of them at once, see Section 4.2.1).

As mentioned above, Python-based Locust will also be used to simulate several users and statistics from it will be used to help evaluate the web server’s performance in this setup.

# 4 The application: development, testing, and evaluation

This chapter describes the app itself & its function, how the app was developed, and how I set up the test environment with an emulator to launch, test, and modify the app. These tests were done to both understand the behavior of the app and to change the app in order to measure response-times from the webserver. The chapter also describes the hardware and software used to do these tasks.

## 4.1 Developing the code for the app

The app came with a lot of code from the start, designed for API level 14 as the minimum SDK and API level 18 as the target SDK. The app should be able to run on those Android versions that are equivalent to API 14 (4.0, Ice Cream Sandwich) through API 18 (4.3).

There were some minor errors and many warnings, but overall it is a basically finished app with many files. DigiArts Entertainment AB had not yet had time to review this version though.

The basic tasks in developing the app was to understand the code and make changes to make the measurements I needed as well as fixing errors I found. What did what in the code took some time to figure out and I did not have contact with those that developed the app. It was a lot of testing with printouts etc. that was needed.

As Eclipse builds the project it creates a number of files. One of these files has the file extension ".apk". This file is used for the installation of the app on the virtual device or on a smartphone. As part of the "Run"-process in Eclipse, the project (the app), in addition to being built and installed on the virtual device or phone, is also launched (started on the emulator/smartphone - this step fails at times). Should the launch fail, but the installation worked, then one simply finds the icon for the app on the virtual device or smartphone and starts the app "manually". In Eclipse they also have something else they call "launch" as part of the process to run the app. It begins before one gets to chose device to launch the app on. It may sound confusing, but it is more that the word launch is used for both attempting to install the app on a device and for starting the app itself on the device once it is installed. So, the steps are basically launch -> select device -> install-> launch (start on the device).

When the emulator is used, one has an automatic debugging of the app and can see printouts from the app in the debugging-window if there is any printout (which helps in troubleshooting). If the USB-cable used to install the app on the smartphone is still attached, it will be debugged too.

The emulator used to test the app on is called Genymotion (from Genymotion) and was downloaded separately. To use the Genymobile's emulator, Eclipse needs a plugin for Genymotion*. An example of selecting this emulator is shown in Figure 4-1. More about the emulator in Section 4.3.

* Download-instructions: https://www.genymotion.com/#!/support?chapter=collapse-eclipse#faq

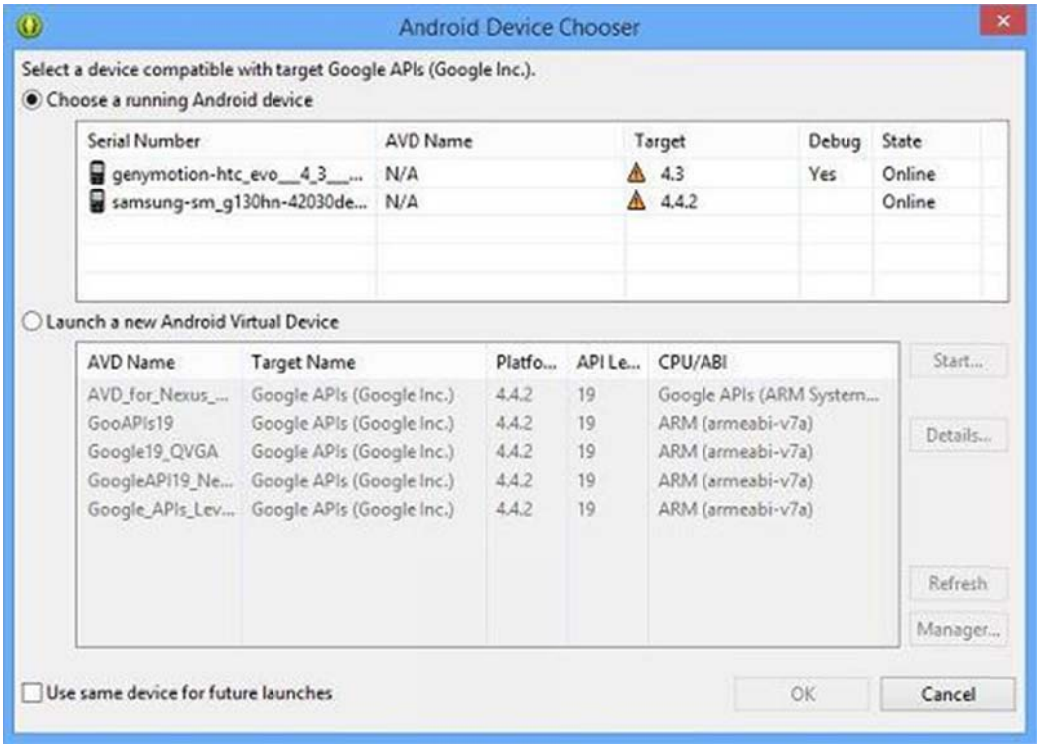



**Figure 4-1: Android Device Chooser with Genymotion Emulator and Samsung Galaxy Young 2 in Eclipse**

## 4.2 The App, Kopplar, and its functions

The first thing a user of the app Kopplar will see is the Login-screen (see Figure 4-2).

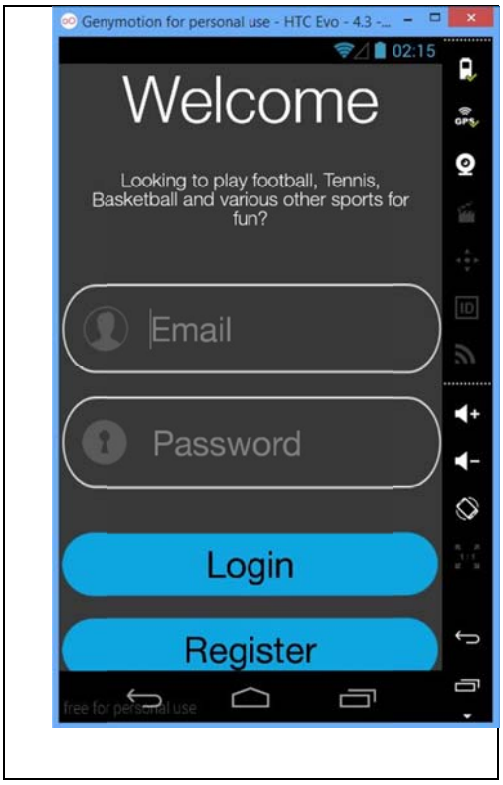



**Figure 4-2: Login-screen**

The user has 2 text-fields for e-mail and password and 2 buttons for login and registration respectively. In addition, under the Register-button, there are 2 more buttons, not seen in Figure 4-2, to allow users to login via Facebook and Twitter respectively. If the user has not used the app before, he or she needs to register by pressing the Register-button and the writing name, e-mail and password in the new screen that shows up. If the user is already registered, he or she fills in the e-mail and password and presses the Login-button.

Once the user has logged in or registered, he or she will be taken to the Home-screen (shown later in Figure 4-5). On the Home-screen, the user can see upcoming events marked with pins on the map, which takes up the main portion of the screen if there are any. In the bottom center, there is a blue button which finds the position on the map for the upcoming events. The top right has a button that allows the user to chat with a friend they have previously added and on the left is the menu-button. Tapping on that will bring up the menu seen in Figure 4-3 below.

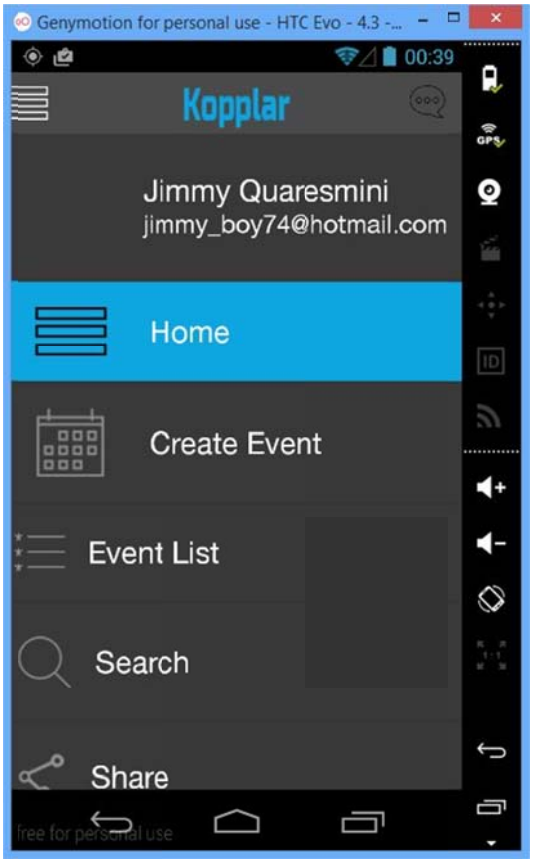



**Figure 4-3: Menu-options in the app Kopplar**

The menu-options are to update one's profile (can add a picture for example), which is the top option where my name is in the picture. One can go to the home-screen described above, create a new sports-event, go to the event list, search, share on Facebook or Twitter and also some options not in the picture. These options come below "share" in the picture and are to add a friend, check one's messages (from friends) and to logout.

## 4.3 Testing and setting up an emulator to use the app-code on

At first this task seemed very straightforward and very appealing - as it is quite enjoyable to try out smartphone apps, even when only running them in emulators. However, it turned out that the Android emulator that Eclipse came with was extremely slow to start. It might take an hour or more before even the start screen was visible. Pressing the "all apps"-button and waiting for a response can take 20-30 minutes or more. 7 of 8 emulated virtual devices I tried failed to start in a "reasonable" time (under an hour), if they started at all. If you abort the startup attempt before the app was loaded, there was a risk that the virtual device becomes corrupt. You chose which Android-API these emulators will run, what CPU, RAM etc. it will have, even what screen it will have. It is not so easy to decide what emulated hardware to chose for optimal performance, especially if the computer they are tested on are a bit old like mine from 2008-2009.

The reason for the sluggishness is that this emulator actually emulates the smartphone's ARM-processor (see further down). If the computer that the emulator runs on has an Intel-CPU, one can get a faster emulator, especially with an Intel x86 atom (can chose 64 too) system image of Google APIs and especially one can chose the Intel-specific x86 Emulator Accellerator option, also called HAXM. This is chosen in Eclipse, for example in "Android SDK Manager" where you can download the HAXM installer (see Figure 4-4 below). Unfortunately, this is not possible on AMD-CPUs. Another CPU-acceleration option was not available because it was only available on Linux.

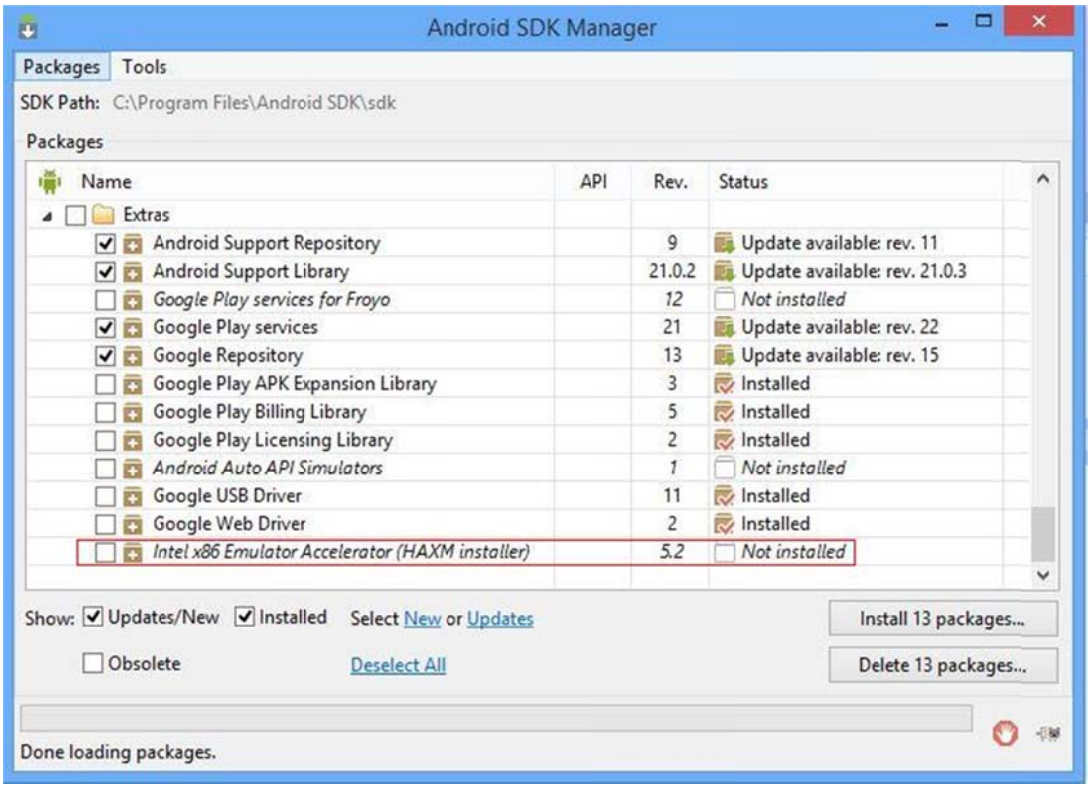



**Figure 4-4: Android SDK Manager with Intel's HAXM-installer marked**

Another option was to find another emulator. Although a solution to the problem with the sluggish "normal" Android emulator was what was sought, what was found was an alternative emulator from Genymobile[30] called "Genymotion"*. Several users had written web postings about its speed I saw when trying to find information about the performance of the "normal" emulator. Genymotion was found to be much faster and more responsive, even if it is somewhat more limited in terms of choices of APIs. For example, it lacks Google APIs (Google apps), which are very important to the app that is being developed. Genymobile chose not to include Google APIs in their virtual devices, but it was still possible to install code that uses these APIs on the virtual device (despite the fact that Genymobile makes no promises or guarantees about these APIs working).

The reason that Genymotion is so fast is because it does **not** emulate the processor of a smartphone, but rather use x86 (32 bit) architecture virtualization. This means that Genymotion uses instructions for a 32-bit CPU directly. The Android emulator that comes with Eclipse simulates the CPU-architecture of a smartphone, which often is an ARM-CPU. So, the Android emulator uses instructions for an ARM-CPU that then gets translated to x86 (or x64 for 64-bit) instructions, which slows it down, a lot on older computers. Further, Genymotion uses the OpenGL hardware acceleration of the computer's graphic card, speeding it up even more.

Many attempts to install Google apps for API 19 on Genymotion failed. This API was the preferred API when using Eclipse and the "normal" emulator because this API was not the absolutely latest, but the next latest version of the Google APIs for non-wearable devices (unlike API 20 which is for wearable devices). API 21 (Android 5.0) is the latest API as of today in January 2015. It was not apparent why this API (19) could not be installed. Finally, an attempt was made with API 18. The emulator said "Failed...", but it turned out that this API version worked!

The process to install the Google apps API was to start with an emulator running a virtual device of either a phone or a tablet with API 18. Then simply drag & drop an executable file (that one had downloaded to the device) to translate the ARM-calls for this virtual device. Next, restart the virtual device and then drag and drop the Google apps API (which you previously downloaded) onto the virtual device for this particular API. Note that there were several Google app API's to choose from. They each have different sizes and some sources for download were not very reliable. Once again you restart the device. Unfortunately, it can take several restarts before the device works. This is because files etc. get updated on the virtual device with each restart. Users have found this through trial-and-error.

When you drag and drop the files [zip-files with the ".zip" extension] to the virtual device Genymotion converts the file to its own internal form and logically places it in the flash memory of the virtual device and installs this application. While this is a very neat feature, it can fail at times for no apparent reason.

If all has gone well, you are now ready to open the "Play store" app on your virtual device. If you can see this app (the play store), then you should be fine. However, during my testing it took many attempts to get it to show and there was no apparent reason when it did not show. Once the Google Play Store app has been opened, one needs to fill in the mail address (used as your user name) and password of one's Google account. Then one can download an app from Google's Play store. Once this is done, one can update the Google apps that have already been installed on the virtual device and set up the app. At this point, the app we developed could be installed, launched, and run on the virtual device.

* http://www.genymotion.com

## 4.4 Installing and setting up Locust

Locust is as mentioned in Chapter 3 based on Python, so to be able to use Locust, one needs to install Python first. Further, a particular version of Python needs to be used and it is not as simple as downloading the latest version unfortunately. The reason is that Locust depends on 2 other small programs called “gevent” and “greenlet”, which requires an earlier version of Python. Sometimes the information was a bit confusing and sometimes one lacked information to help one understand what needed to be done or how to correct an error. Almost all of the problems related to which version of Python, gevent and greenlet one used though. Much trial-and-error was needed and consultation of Google to finally get the versions to match. 4 different versions of Python were tested in this process and 3 uninstallations before getting it right.

The result was that Python needed to be version 2.7.5. Gevent should be version 1.0.1 (not 0.13 as Locust mentions in its installation instructions) and Greenlet should be 0.4.5. Python 2.7 and 2.7.9 will not work with Locust, even if gevent and greenlet installed on Python 2.7.

Another note is that a program used in Python to install other programs called “pip” had to be installed after Python was installed. This was done by first visiting a web site that had a link that led to displaying (as text on a website) a Python-file needed for “pip” so one had to copy all of the contents of this (marking it and copy the text) and store in a Python-file (ending .py). Then, by typing in a Windows command prompt “python get-pip.py” one installed “pip”. Here, “get-pip.py” is the file that one saved before. One had to first make sure the Windows PATH environment variable pointed to the Python-catalog or be in that folder.

Once all of this was done, Locust itself could be installed using “pip” by simply typing “pip install locustio”.

## 4.5 Simulation of app users on mobile devices

Two different methods were used for testing the effects of multiple app users: testing using the emulator and testing using Locust.

### 4.5.1 Measurements using the emulator and devices

To simulate users communicating with the web server, several (specifically 5) HTTP POST requests (the same POST request) were performed. The reason for making 5 requests and not 10 or 1 is because the app used 5 HTTP POST requests if a user chose to log in, look at the home screen (see Figure 4-5 below), and then log out. That whole process on the emulator took about 1 minute to do. The app chose to do HTTP POST requests and not HTTP GET or other HTTP-commands when one only logged in, went to the home-screen and logged out so that was chosen.

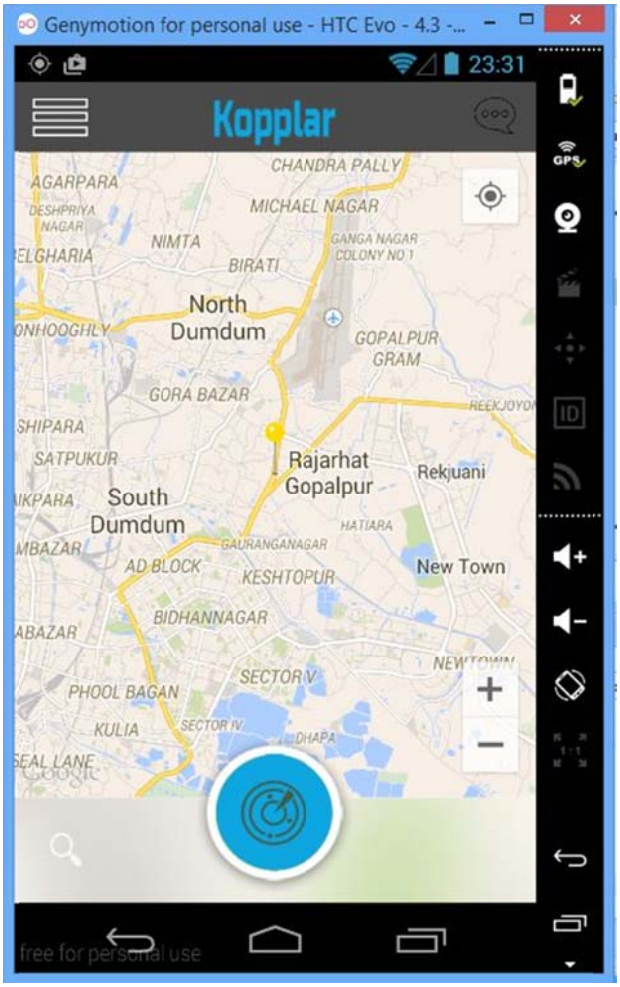



**Figure 4-5: Home screen on the app (called Kopplar) on the Genymotion emulator**

It took time and experimentation with the Java-code to find what to measure and to get the measurements correct. In the course of doing so, some interesting things were discovered. For the emulator, it took approximately 600–700 milliseconds (varying from roughly 300 to 1,000 milliseconds) to make an "httpClient.execute(httpPost)" (see Figure 4-6 below). Since this represents the actual communication with the server, this duration was measured.

```
startTime = System.currentTimeMills();
response = httpClient.execute(httpPost);
endTime= = System.currentTimeMills();
duration = (endTime – startTime);
```

**Figure 4-6: HttpClient doing a HTTP POST request in the normal way while I measure the time**

Additionally, it was found that network-related tasks had to be done in threads or in the background (that is, not in the main thread) or an exception, "NetworkOnMainThreadException", would be thrown. The reason is that networking may take some time to complete and that may make an app unnecessarily slow.

Another discovery was that the "HttpClient" that was used in this code used a single TCP connection. Hence, if another function also called "HttpClient", even creating a new instance of it, it could (but not always) interfere, as the connection was still be open (i.e., had not been released – see [31]. In that case an exception would be thrown and no measurement could be made. This seemed

to happen rather randomly, but the problem only came once I tried to make separate POST requests to measure the time of, as the app were still doing its own requests. The timing of these exceptions was not measured.

The answer to these problems was to create a new function to be used only for testing that did not use "HttpClient", but instead used "HttpURLConnection". "HttpURLConnection" does not use "HttpClient.execute", but instead opens a connection, creates an outstream (Figure 4-7), and writes on that stream. After which it will disconnect. To receive and read the response, an InStream needs to be created and a BufferedReader used (see Figure 4-8). As neither out- nor instreams are buffered, the Buffered Writer and Reader are needed. One should close out- and instreams afterwards.

One might expect this second approach to take longer, given that more commands are used. However, it was actually found to be faster. From the opening of the connection until the function had written on the outstream, it took about 1-200 ms compared to roughly (not calculated – average) 600-700 ms as written above just for "httpCllent.execute(httpPost)". As a parenthesis it can be mentioned that Android recommends using "HttpURLConnection" instead of "HttpClient".

```
OutputStream out = urlConnection.GetOutputStream();
BufferedWriter writer = new BufferedWriter(
          new OutputStreamWriter(out, "UTF-8"));
```

**Figure 4-7:** **Using an URLConnection in my function to get an outstream and then creating a BufferedWriter to later do the HTTP POST request with. Have started measuring the time before this.**

```
// Get Response
InputStream is = urlConnection.getInputStream();
BufferedReader rd = new BufferedReader(new InputStreamReader(is));
String line;
StringBuffer response = new StringBuffer();
while ((line = rd.readLine()) != null) {
          response.append(line),
          response.append('\r');
}
// rd.close()
endTime= = System.currentTimeMills();
duration = (endTime – startTime);
timeValues.add(duration);
```

**Figure 4-8:** **Reading the response from the HTTP POST request using an InputStream and a BufferedReader while still being in my function. After reading, I stop measuring the time and store the value (in ms).**

To make it a little more similar to a normal user using the app (which uses the slower httpClient POST request as mentioned above), delays (Java's Thread.sleep-command) of 500 milliseconds in these threads that ran my function with a faster POST request were implemented. To clarify, my function ran as a thread and in this function the sleep-command was used. The app's code limited it to max 5 connections at the same time. For that reason, I chose to also have 5 threads running at the same time, where each thread ran one connection doing a POST request. This gave a limit to how much the web server was tested, but it was also the same way the app was programmed so it seemed reasonable.

Each device (and the emulator) was decided to first simulate 5 "users". As described above, 1 "user" is equal to 5 HTTP POST requests. Of course, the app's functionality gets affected by this in how it works, but as the measurements were of the web server communicating with the app and not

the app itself, this was deemed acceptable. As long as the device could send out requests in about the same rate that I estimate several normal users would, it would be fairly similar. As written above, a normal user logging in, going to the home screen and logging out takes 5 POST requests and 1 minute. If 5 users are logging in at the exact same time, their POST requests would come close together in time (25 POST requests in 1 minute) and I tried to make the app simulate that.

Further, the app was also tested just at different times without being synchronized. However, a timer was introduced in the java-code for the app that enabled it to be run at a specific time on all devices it was installed on. This was to simulate several users communicating at the same time with the server. So, the app could run a test with 100 users on the emulator and a smartphone at exactly 9 pm (21:00) for example.

After the first test with 5 users worked well, the number of users was increased to 10, 15, 20, 25, 50, and 100. Since little difference was noticed with these numbers of users, the number of users were increased to 1,000 in steps of 100 (200, 300 and so on) on the individual mobile device (in the app). In total, the number of users went up in steps of 200 when I increased by 100 users as both a smartphone and the emulator tested 100 more users when I ran them simultaneously. The first tests were on the emulator alone, but later tests were on the emulator and smartphone at the same time.

Initially, tests were conducted only measuring the time it took to send out the POST requests with no consideration to the response from the server. However, tests were later conducted that also included the time to read the response from the server for 100-1,000 users in steps of 100 (see Chapter 5). 5 measurements were taken on both the emulator and the smartphone at the same time using the timer mentioned earlier per number of users simulated. So, 5 tests for 100 users reading responses from the server, 5 tests for 200 users and so on up to 500 users. 500 users on both the emulator and the smartphone at the same time can be seen as 1,000 simultaneous users. The measurements that did not read the responses were not always 5 per number of users. Those tests were also undertaken on the smartphone alone or the emulator alone at times for comparison.

### 4.5.2 Measurements using Locust

Locust was used to both compare with the results from the emulator and devices and to test with more users than 100 (it was thought that Locust would be more effective at simulating more than 100 users). Tests were performed using 25, 50, 75, 100, 500, and 1,000 users to get a “sense” of whether the web server’s performance was affected or not. There were no real changes up to 100 users, but large changes were evident as the number of users approached and reached 700 (over 300 really). As a result I focused on testing with 100-700 users. I had initially planned to test from 100-1,000 users, but tests failed when testing with 800 users – when the server stopped responding. As written above, I once tested with 1,000 users *without* the server stopping to respond, but that was during a test of Locust and the server, hence I do not consider that test in my evaluation of the app.

To start a load test with Locust, one has to open a command prompt on Windows and set the current directory to where the Python-file (simply called “locustfile.py”) is that has the information about the tasks for the Locusts about to “swarm”. It is usually in Python’s “Scripts”-folder. In the file I used I had 2 POST requests, but one was only done at the start of Locust to login. Further, I wrote in the file that it should do its POST request and then wait between 5 and 15 seconds. The start-command in the command-prompt (have to have fixed the PATH-environment variable first) to start Locust is in my case “locust –H http://kopplar.com”. Where –H tells Locust which host to communicate with (http://kopplar.com here).

After that is done and one checks that Locust is working without problems, one opens a web browser and goes to the address http://127.0.0.1:8089/, which is Localhost, port 8089. The first screen one then sees is the below selection-window in Figure 4-9.

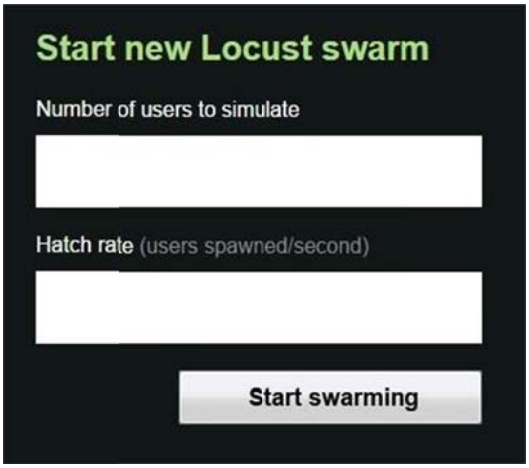



**Figure 4-9: Start options for Locust. Selection of number of users and users / second.**

Locust can be set up in different ways so that it mimics different user behaviors, such as making different GET- or POST-requests, waiting times between each call, logins, and more. In the specific test done here, a user would initially do a POST request to login, but after that only do one task: make a POST request to get event data. The event contains information about an upcoming sports event that had been put into the system for testing. This event had been stored in the database on the web server by using the app. To get that event, I programmatically created a message to send as a POST request to the server containing three things: latitude, longitude, and user-ID. This is exactly the same request that would be made by the app in my function there.

In real life locusts are insects. In this load testing tool, one can decide the "hatch rate", which means number of users created ("hatched") each second. Number of users can also be set to a fixed number. Both of these parameters are normally set via a web interface connecting to "localhost". Each of these users then sends a POST request once every 5-15 seconds, chosen according to a uniform random distribution.

# 5 Analysis

In this chapter, I present the results of the testing described in the previous chapter and discuss these results.

## 5.1 Major results

The major results fall into three categories: measurements of the web server response times, GPS location measurements, and Locust testing of the web server. Each of these will be described in the subsections below.

### 5.1.1 Smartphone/emulator measurements of web server response times

The results of the tests can vary quite a lot from time to time. Overall, a big impact upon the performance of the communication with the web server was not noticed, even with up to 1,000 simultaneous simulated users.

In Table 5-1 under the column for 100 users we can see that some requests failed of the 100 * 5 HTTP POST requests sent to and received from the web server from the app. As the number of users increased, some requests failed so not exactly 500 (or 1000, 1500 etc.) requests succeeded. To send these 500 requests took about 2 minutes. In the case of 200 users means 5*200 = 1,000 POST requests were sent, etc.. In the case of 100 users these 500 requests were sent in about 120 seconds (2 minutes), which gives an average rate of 4.17 requests/second. This value will later be compared to the rate when testing with Locust. The values in the two tables below are in milliseconds. Remember that 1 second = 1,000 ms. The values are the averages of the 5 measurements for each test load (i.e., for a given number of users).

**Table 5-1:** **Average response times (in ms) on smartphone and emulator with 100-500 users at the same time. The number of requests made is shown. Note that there is a programmatic 500 ms delay included in these measurements.**

| Users | 100 | 200 | 300 | 400 | 500 |
|---|---|---|---|---|---|
| Galaxy Young2 | 1401 ms | 1306 ms | 1246 ms | 1214 ms | 1275 ms |
| Genymotion | 1541 ms | 1441 ms | 1367 ms | 1344 ms | 1421 ms |
| Requests | 500 | 1000 | 1500 | 2000 | 2500 |

Results where the response of the HTTP POST-message from the web server was not read (or printed for evaluation) is shown in Table 5-2 for comparison. This was not done at first because the way the app did the POST-requests was a bit different from how I did it in my function. The app's POST-requests were done in one line without a buffered reader, unlike in my function which used both a buffered reader and writer for reading responses and sending requests respectively. The reason is that they used the HttpClient in the app while I in my function in the app used HttpURLConnection. HttpClient's POST-request is simpler to write, but HttpClient is not recommended to use by Android (as they do not actively develop it to avoid breaking compatibility). When HttpClient's POST-request is done, the response is gotten too on the same line in the code as far as I can tell and stored in a variable called "response". To read the POST-message, they later in the code use part of that variable by calling "response.getEntity()" and have a function to interpret what the response says.

In my function, because of using HttpURLConnection, I have to first create a connection, then make a POST request and then read the response. These are all separate parts in the same function.

**Table 5-2:** **Average time (in ms) to send a POST request *without* reading the POST request 's response from the server (see below) measured on smartphone and emulator with 100-500 users at the same time. Note that there is a programmatic 500 ms delay included in these measurements. Equal number of requests as in Table 5-1.**

| Users | 100 | 200 | 300 | 400 | 500 |
|---|---|---|---|---|---|
| Galaxy Young2 | 703 ms | 718 ms | 749 ms | 739 ms | 739 ms |
| Genymotion | 665 ms | 676 ms | 690 ms | 668 ms | 681 ms |

The same number of requests was made for the tests results shown in Table 5-2 and Table 5-1. I did these tests that did not read the responses first as my first thought was to see if the server's ability to handle the POST request was affected by the number of users and I was not very interested in the response (as long as it was a normal POST response without error). This goes back to what I wrote above about how the POST-request is done with one line in the app normally and how everything from that request is stored in one variable (from which the response can sort of be "extracted"). Reading the POST response to me seemed more like processing of the information after the POST request was done and thus not something I felt interested in measuring at first. I did read the POST responses after that and those measurements are in table 5-1 above.

Again, as mentioned in Chapter 4, I let the threads making the calls sleep 500 ms, in order to make it similar to the app's own POST requests (which are sent at a slower rate). This programmatic delay of 500 ms is included in the measurements, thus the actual response times shown in both Table 5-1 and Table 5-2 are 500 ms shorter than the values shown.

The Galaxy Young2 smartphone was also tested on its own (without the emulator running simultaneously) several times. It was tested 3 times with 100, 200, and 300 users. It was also tested 2 times with 400 users and 5 times with 500 users. The average response times for 100-500 users were as follows: 692, 712, 726, 722, and 720 ms. These are roughly 10-20 ms faster than the results in the table above, but most results in the table above come only from one run (apart from where 500 users were simulated).

Further, tests without reading the responses were done on two other Samsung smartphones (both running with 100 users) (as described in Chapter 3) while the emulator also ran with 100 users. Only twice did both smartphones run at the same time (and with the emulator) due to mistakes in the emulator or by the smartphone users. The average response time for the Samsung S5 running with 100 users was 776 ms. While the Samsung S4's average response time was 740 ms and the emulator's average response time for these two tests were 656 ms.

### 5.1.2 GPS-measurements on smartphone/emulator

As an aside, the time it took to get the GPS coordinates was measured, but it was not easy to exactly see where programmatically this reading was done. Regardless, it on average took **165,890 ns**, or 0.17 ms to do this, with larger variations. For example, one reading took over 1.8 million ns, while another took about 65,000 ns. Most took around 85,000 ns though. These readings are from the Samsung Galaxy Young2 while testing with users where the responses were read. The *median* value was **76,695 ns**.

For comparison, the emulator's average time to get GPS coordinates during the same test was **1,652,961 ns**! However, there was one extreme value of over 34 million nanoseconds, thus affecting that average a lot. If one excludes this one extreme result, the average becomes 231,976 ns. *Median* turned out to be **69,705 ns**, so a huge difference.

### 5.1.3 Locust's test of web server response times

Locust's results for 100-700 users, in steps of 100 are shown in Table 5-3.In this table, RPS means Requests Per Second. All tests with 100-500 users were done with 1 user created every second, except for one test with 400 users, when 2 users were created every second. There were five tests for each of the different numbers of users. For the cases of 600 and 700 users, 2 users were spawned per second to shorten the duration of the tests.

**Table 5-3:** **Locust-tests with 100-700 users and statistics. "Dur." = Duration. Average and Median are response times in milliseconds (ms).**

| Users | 100 | 200 | 300 | 400 | 500 | 600 | 700 |
|---|---|---|---|---|---|---|---|
| **Average** | 866 | 4245 | 5759 | 12125 | 12599 | 13810 | 53383 |
| **Fails(%)** | 2.8 | 17 | 31.6 | 40.6 | 30.4 | 22.8 | 30.83 |
| **Requests** | 706 | 1259 | 2374 | 2304 | 2692 | 3523 | 1731 |
| **RPS** | 9.8 | 6.6 | 13.08 | 6.2 | 14.98 | 20.78 | 7.96 |
| **Dururation (minutes)** | 3 | 5 | 9 | 13-14 | 13-14 | 10 | 9 |
| **Median** | 470 | 770 | 5100 | 11000 | 11000 | 13000 | 38000 |

The RPS value shown in the above table is the value at the time when the test was stopped and may thus be somewhat misleading. The number of requests is also a bit misleading because it was not possible to run the test for exactly the same amount of time in every test. However, these results can be used to compare with the tests of the app and the number of requests it made depending on the number of users that were simulated. This comparison is given in Section 5.1.1. One can also note the low number of requests with 700 users due to the slow response times and high failure rate.

The duration of the test runs varied a bit. It is worth to know that it took quite some time to create all of the users. Once the desired number of users was created and running, the rest of the test run went pretty fast considering the number of requests that I wanted. As an example: when 300 users were simulated, creating the 300 users took 7 minutes, while getting about 2,000 requests as I wanted took only 2 minutes. The total duration was then 7 + 2 = 9 minutes. The test duration is the time from the creation of the first user (start of the test) until the test stopped, including "user-creation-time". The test duration for 100 users was 3 minutes, for 200 users 5 minutes, for 300 users 9 minutes, for 400 users about 13-14 minutes (one longer), for 500 users roughly 13-14 minutes (one longer), for 600 users 10 minutes, and for 700 users 9 minutes. Note that for both 600 and 700 users the rate of creating new users was twice that of the other tests.

Lastly, there is the median value. For all measurements a median value was gotten. What is shown in the table is the median value of the median values. Median is the middle value when one ranks results from lowest to highest.

## 5.2 Reliability Analysis

Given that one millisecond is a short amount of time, a thousandth of a second, relying on exact values is probably not recommended. However, what we want to look for is a trend, specifically if the server struggles (i.e., has increasing delays to requests) as the number of users increases. Furthermore, fluctuations in response times could be due to a number of factors, such as load on the web server when tests are done, load on computer or mobile device when the tests are done, speed of the internet connection, and the CPU speeds and capacity of the specific web server used – just to name a few potential factors.

Locust's measurements introduced quite a heavy load on the server and the server was not meant to handle these loads, so I think one should look at the "bigger picture" of the results from Locust: above 300 users the results are very poor for this type of server with this POST request at this frequency. However, the exact numbers are perhaps less reliable as a dedicated server would be more likely to show better performance than this shared server as will be discussed in Section 5.4.

## 5.3 Validity Analysis

Numbers are collected programmatically and presented on screen after each run (as shown in Figure 5-1). At times some requests got no response from the server, but that was to be expected given the high loads and the performance of the server. There was very little risk of human error in reading the results.


**Figure 5-1:** **Results in a popup I did in the app after a test of 100 users. The result to focus on comes after "5)" ("1611 ms…").**

Locust's data are gathered programmatically so they are valid readings, but as the discussion below shows, there is some difficulty in interpreting these results.

The statistics screen (shown in Figure 5-2) is open under the entire test with a button to stop the test where it says “Stopped” in the picture. After “POST” to the left, it says “/phone/api.php”, which is the location of the PHP-file to do the HTTP POST requests to. Unfortunately, the text above the stats is not clear when making the picture smaller. From left it says in order: Type, Name, “# requests”, “# fails”, Median, Average, Min, Max, “Content Size” and “# reqs/sec”.

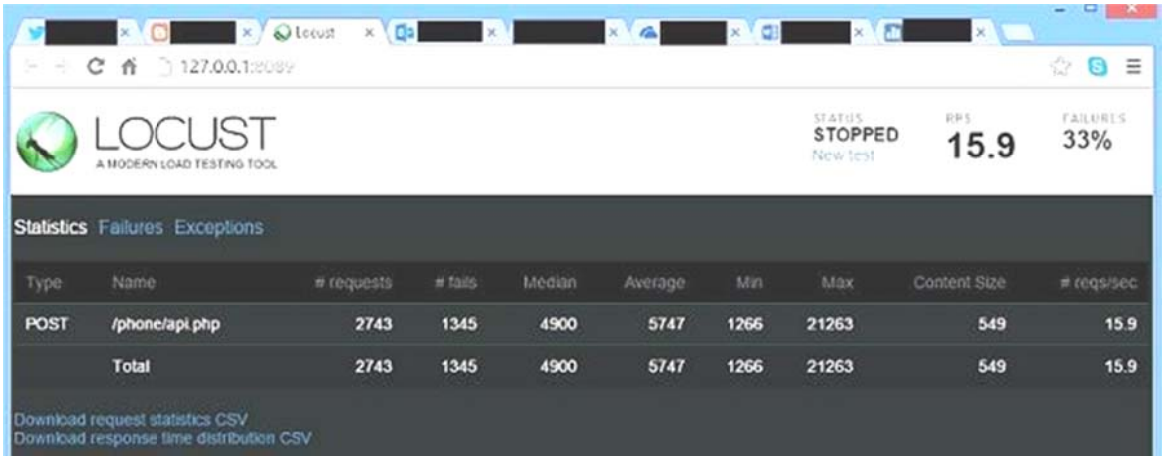



**Figure 5-2: Test results after having run Locust with 300 simulated users.**

In the top right besides “Stopped”, it says “RPS 15.9” (Requests Per Second, same as the number farthest to the right in the statistics). To the right of that it says “Failures 33%”. In other words: not very good.

## 5.4 Discussion

No clear trend found related to the number of users using the app itself with the additional programming. Differences between emulator and the mobile device are in this case less interesting. Locust showed large effects. Actually, the Locust-results showed such poor performance that it was a bit hard to believe at times. After attempting to test with 800 users with Locust, the server actually stopped responding to the tests so no requests succeeded. I subsequently learned that the server needed to be a dedicated server in order to be able to handle the attempted traffic loads. Sending requests every 5-15 seconds was too fast for the server in question according to the company hosting the server (HostGator). This explains the very poor performance as the number of users increases. See the end of this section for more discussion of the Locust-results.

Deciding exactly what to measure in the app was a bit tricky. Both with regard to which request to measure, and whether to include the response or not. Further, it is of course open for discussion whether a certain number of HTTP requests can accurately simulate a user, but in this case the behavior of the user that was interesting were those that interacted with the web server, rather than those interactions with the app itself. Locust, the load-testing tool uses HTTP requests, such as POST, which makes it seem reasonable for me to also use them to simulate users. Locust can also do other HTTP requests such as GET, PUT etc.. I could have done that also if I had wanted, but wanted to do POST requests as the app does so. Locust can also read web pages, watch movies, read a thread on a forum, reply, print information in the command prompt and more.

I initially used a 9 second delay in my function in the app to try to simulate user behavior, but after some time the delays felt a bit superfluous and were removed in my function, although a 500 ms delay was used in my function when doing the POST request. I wanted my function’s POST request to take about as long time as the POST request normally done by the app, thus I kept this

delay of 500 ms. As noted in Section 4.5.1, my own function in the app did the POST request 500 ms faster than the POST request used by the app.

Deciding upon the number of users to simulate is another open question. Simulating up to 10,000 users might have been interesting to really be thorough. However, for the time being, testing with less than 1,000 users seems adequate.

I felt less confident about Locust's results at first, especially given that I had less control over it, even though I wrote the python-file telling it what to do. My experience is that it is definitely a good load-testing tool though and I trust its results now. Given that the web server used in this thesis project was not dedicated, the results of Locust testing may seem much worse than the situation really is. I expect that a dedicated web server with the same hardware may show better results, because the server was running other processes and tasks, it may drop some of the requests due to restrictions on number of connections and other resources available to service the requests that I was generating. My test results show that this non-dedicated server cannot really handle many simultaneous users at all. This is confirmed by the big increase in average response times between 300 and 400 users in the Locust-statistics shown previously in Table 5-3. When 400 users were simulated it took about 12 seconds on average to get a response, which is pretty terrible. It got even worse when the number of users was increased from 600 to 700 as can be seen in Figure 5-3 below. Average was about 53 seconds, making the user experience very poor. Values in Figure 5-3 have been taken from Table 5-3.

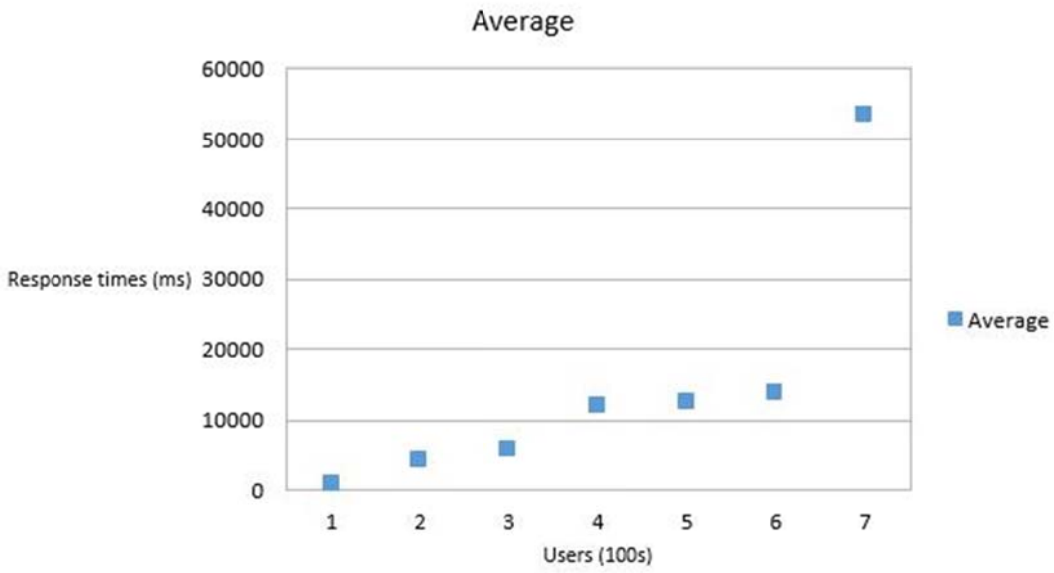



**Figure 5-3:** **Average response times on Locust. 100-700 users.**

# 6 Conclusions and Future work

In this chapter I present my conclusions after having worked on this thesis project and describe what limitations I had. This is followed by some suggestions for future work on both testing the performance of the web server and on the app itself. The chapter concludes with my reflections on how the app can affect people and society.

## 6.1 Conclusions

I was able to make the changes needed in the app to test what I wanted and got results for the tests that I wanted to make. The results were a little surprising to me, but not too much so. I also learned how to create an app for Android smartphones. Although I have only scratched the surface of what there is to know about this subject, I learned a lot about Android in general. I had expected to see some trend in the performance of the web server as the number of users increased, but no clear trends are apparent (in my own test). Furthermore, I was surprised to see the slowest smartphone do so well in these tests. Locust's results also surprised.

I have learned that smartphones really are "smart" in the sense that they have lots of new technology in them that can be used in many fascinating ways. I also learned more about localization, even though this was not the focus of the tests conducted for this thesis project. Further, I could see the benefit of planning my time over the course of this thesis project. This planning was important as time went by quickly and a lot needed to be done in this limited time.

I learned something about what information is useful to store about users within apps such as this. I also learned how to work with Google maps, specifically how to get a developer's key to develop such applications. However, only a limited amount of work was done with the actual map itself.

The Eclipse integrated programming environment can definitely be scary to the developer at times and give you the impression that something is seriously wrong, when perhaps all you need to do is to clean the project. I also learned several new different things with regard to programming.

Had I not already done so much app programming before I began the thesis, I would probably not have completed the project on time. There really was limited time to develop the complete app in this project, while still having sufficient time to conduct the tests described earlier. Fortunately, it was possible to make temporary changes to the app to facilitate my testing.

Along the way I encountered a lot of the errors, many of which took a fairly long time to find a solution to. This reinforced the value of planning my time.

I had good help along the way, which was very important. This guidance enabled me to avoid getting stuck along the way.

Ensuring that everything is ready before beginning the thesis project proved to be important. Halfway through this thesis project, it was unclear whether I would get access to a web server, which was very important as I needed the web server to provide many files which the app needed to work properly.

Looking at previous works early on was a good way to get a sense of what it means to do a thesis, and it also enabled me to compare what I was doing with what others have done as well as to learn more generally how to carry out a thesis project.

If I would have to do it all over again, perhaps I would have tried the app on a larger number of different types of smartphones, this would have been especially useful at certain times during my app development process. Planning the tests was important and actually carrying out the testing

took longer time than I had expected. Perhaps I should have focused more on using load-testing tools, but I am not sure. I enjoyed trying to do the load testing myself programmatically.

I should have taken more time to understand the pre-requisites of the app, such as what type of web server was needed, and how to have appropriate access to a suitable web server in advance, thus avoiding the nasty surprise that I faced after having done a lot of testing already – when I found that the web server was significantly underpowered. A suitable server needed to be able to handle 100-1,000 users, if that is the number of potential users that you want to support. Further, prior to the start of the thesis I also should have had more information about the PHP-files on the web server that were necessary. There are 7 PHP files that are necessary for this app and they are not all in the same location (folder) on the server. Some of them contain a lot of code, while others are shorter.

A clear conclusion is that you should not assume that others know what you need. However, sometimes it is hard to know what one needs in advance.

## 6.2 Limitations

There were three major limitations encountered during this thesis project:

- Not having access to more Android smartphones.
- Not having a smartphone of my own during most of the thesis project was a major limitation. Finally, I got one towards the end of the thesis project and could run tests on it.
- Not having a dedicated web server when testing.

## 6.3 Future work

My suggestions for future work are:

- Scale the app to support even more users, for example up to 10,000 users.
- Create a website for the app.
- Launch the app at the Google Play Store.
- Analyze how the app affects people in real life, using polls or similar means of studying user opinions. See Section 3.2.
- Test the app at times of the day when internet activity peaks, for example according to [32] would be at 5 in the morning and 10 pm at night in USA.
- Expand the app to other types of events where people meet.*
- Test web servers with different hardware configurations and compare the results.

## 6.4 Reflections

An app such as this can help people meet and do sports, which can be good both for mental and physical health. If health would improve, it would not only be good for the individual, but for society as a whole.

The sports meant to be exercised with this app have been imagined to be team sports to a big extent, but one could see it being used for people running together perhaps and similar sports that you do in pairs (tennis is an example in the app). Many of these sports, if not most, are indeed

* Currently I am working on such an app myself for Windows 8.1, but this app is not related to my thesis work for this company.

outdoor sports, but several of these can be done indoors as well. So, one could certainly say there may be an environmental impact, even if not big provided people don't litter, which cannot be guaranteed of course. Apart from littering, the environmental impact is basically limited to wear and tear of the ground and possibly disturbing wildlife if sports outdoors have such an impact (running in the nature may have this effect).

Should the app become popular so that many uses it, it could mean increased income for those that have for example football pitches or other places where sports are exercised that are exercised together as the ones this app tries to help with. Although in Stockholm there is a shortage of pitches and halls for indoor sports I think so therefore maybe they will not earn so much more.

Another possible economic benefit is that the app could benefit the app owner of course if the app becomes a success and ads in the app for example could be used to generate income.

Of course, the company renting the web server would make money off of the web traffic to and from the server.

As brought up in Sections 1.1 and 1.3 (and above in this section) an app like this could help improve people's health [see [33], [34], and [35] for the benefits of sports to health]. This would likely benefit those with fewer friends the most, but by no means only these people. People with fewer friends benefit more from meeting new friends and perhaps even gives them a chance to do something that they could not do otherwise, such as exercise with others, avoiding the need to pay to go to an aerobics class or similar, which may cost quite a lot. However, in the end the use of the app depends on the individual app-user and his/her situation.

There is of course also a risk of people possibility using this app to abuse others, hence those making and distributing the app must at least be aware of this risk. The users also need to be aware of this. Hopefully such problems will be very rare.

Another advantage of this app is of course that it can bring the user a lot of fun, as it can make it easier to get to meet others and exercise and have fun. Location-based services affect how we live our lives today as many of us have smartphones with apps that check for our location or maybe we "check in" on Facebook and other social websites. Thereby we see more about our surroundings, but we also let others know where we are more. This can be good, but also possibly a risk factor when a person may intend to do us harm. Case in point is the sad case of the man who used an app to see where police officers were near him in New York City and then sought them out and shot them unprovoked at the end of 2014. In other less extreme cases, location can be an annoyance due to vendors sending us ads via our smartphones depending on our location, but this can also be beneficial to help us out or help us have more fun. Pew did a poll on how location-based services are used by people and related questions as can be seen at [36].

## References


[1] Pontus Gyllensten, 'Het debatt om anläggningskrisen i Stockholms stad', *Stockholmsidrotten*, Updated: 2014-09-29 08:37. [Online]. Available: http://www.rf.se/Distrikt/StockholmsIdrottsforbund/Varanyheter/2014/HetdebattomanlaggningskriseniStockholmsstad/. [Accessed: 18-Jan-2015]

[2] Stockholms Fotbollförbund, 'Fakta', 18-Jan-2015. [Online]. Available: http://www.stff.se/om-stff/fakta/. [Accessed: 18-Jan-2015]

[3] 'digiarts.wuzzel.com |', 18-Jan-2015. [Online]. Available: http://www.digiarts.se/. [Accessed: 18-Jan-2015]

[4] 'Stockholms anläggningssituation — Stockholms FF', 18-Jan-2015. [Online]. Available: http://www.stff.se/kommun-anlaggning/stockholms-anlaggningssituation/. [Accessed: 18-Jan-2015]

[5] Lantmäteriet, 'GPS och andra GNSS', 18-Jan-2015. [Online]. Available: http://www.lantmateriet.se/Kartor-och-geografisk-information/GPS-och-geodetisk-matning/GPS-och-satellitpositionering/GPS-och-andra-GNSS/. [Accessed: 18-Jan-2015]

[6] Lantmäteriet, 'Glonass', 18-Jan-2015. [Online]. Available: http://www.lantmateriet.se/sv/Kartor-och-geografisk-information/GPS-och-geodetisk-matning/GPS-och-satellitpositionering/GPS-och-andra-GNSS/Glonass/. [Accessed: 18-Jan-2015]

[7] G. Sabak, 'Tests of smartphone localization accuracy using W3C API and Cell-Id', in *Computer Science and Information Systems (FedCSIS), 2013 Federated Conference on*, Krakow, Poland, 2013, pp. 845–849.

[8] WiGLE.net, 'WiGLE: Wireless Network Mapping'. [Online]. Available: https://wigle.net/. [Accessed: 16-Nov-2014]

[9] Navizon Inc., 'How it works', 18-Jan-2015. [Online]. Available: http://www.navizon.com/navizon-how-it-works. [Accessed: 18-Jan-2015]

[10] Denis Foo Kune, John Koelndorfer, Nicholas Hopper, and Yongdae Kim, 'Location leaks on the GSM Air Interface', in *19th Network and Distributed System Security Symposium*, 2012 [Online]. Available: http://www-users.cs.umn.edu/~foo/research/docs/fookune_ndss_gsm.pdf

[11] P. Bahl and V. N. Padmanabhan, 'RADAR: an in-building RF-based user location and tracking system', presented at the Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2000), Tel Aviv, Israel, 2000, vol. 2, pp. 775–784 [Online]. DOI: 10.1109/INFCOM.2000.832252

[12] J. Schmid, T. Gadeke, W. Stork, and K. D. Müller-Glaser, 'On the fusion of inertial data for signal strength localization', presented at the 8th Workshop on Positioning Navigation and Communication (WPNC), 2011, 2011, pp. 7–12 [Online]. DOI: 10.1109/WPNC.2011.5961006

[13] Google Inc. and the Open Handset Alliance, 'Android Developers'. [Online]. Available: http://developer.android.com/index.html. [Accessed: 16-Nov-2014]

[14] Google Inc. and the Open Handset Alliance, 'Welcome to the Android Open Source Project!'. [Online]. Available: http://source.android.com/. [Accessed: 16-Nov-2014]

[15] Dan Morrill, 'Announcing the Android 1.0 SDK, release 1', *Android Developers Blog*, 23-Sep-2008. [Online]. Available: http://android-developers.blogspot.se/2008/09/announcing-android-10-sdk-release-1.html. [Accessed: 16-Nov-2014]

[16] Google Inc. and the Open Handset Alliance, 'Location and Maps', *Android Developers*. [Online]. Available: http://developer.android.com/guide/topics/location/index.html. [Accessed: 16-Nov-2014]

[17] Google Inc. and the Open Handset Alliance, 'Location APIs', | *Android Developers*. [Online]. Available: http://developer.android.com/google/play-services/location.html. [Accessed: 16-Nov-2014]

[18] Stack Exchange Inc., 'geolocation - How do I get the current GPS location programmatically in Android? - Stack Overflow'. [Online]. Available: http://stackoverflow.com/questions/1513485/how-do-i-get-the-current-gps-location-programmatically-in-android. [Accessed: 16-Nov-2014]

[19] Pin-Fan Lee and Shuchih Ernest Chang, 'Building Location-Based Service Based on Social Network API: An Example of Check-In App', presented at the IEEE International Conference on Green Computing and Communications (GreenCom) and 2013 IEEE and Internet of Things (iThings/CPSCom), Beijing, 2013, pp. 1904–1909 [Online]. DOI: 10.1109/GreenCom-iThings-CPSCom.2013.354

[20] U.V. Ramana and T.V. Prabhakar, 'Some experiments with the performance of LAMP architecture', presented at the The Fifth International Conference on Computer and Information Technology, 2005. CIT 2005., 2005, pp. 916–920 [Online]. DOI: 10.1109/CIT.2005.169

[21] Facebook, 'Login Dialog', *Developers Docs*. [Online]. Available: https://developers.facebook.com/docs/reference/dialogs/oauth/. [Accessed: 16-Nov-2014]

[22] IETF OAuth WG, 'OAuth 2.0 — OAuth'. [Online]. Available: http://oauth.net/2/. [Accessed: 16-Nov-2014]

[23] D. Hardt, 'The OAuth 2.0 Authorization Framework', *Internet Req. Comments*, vol. RFC 6749 (Proposed Standard), Oct. 2012 [Online]. Available: http://www.rfc-editor.org/rfc/rfc6749.txt

[24] G. Retscher and T. Hecht, 'Investigation of location capabilities of four different smartphones for LBS navigation applications', presented at the 2012 International Conference on Indoor Positioning and Indoor Navigation (IPIN), 2012, pp. 1–6. DOI: 10.1109/IPIN.2012.6418892

[25] Benjamin Henne, Christian Kater, Matthew Smith, and Michael Brenner, 'Selective cloaking: Need-to-know for location-based apps', presented at the Eleventh Annual International Conference on Privacy, Security and Trust (PST), 2013, 2013, pp. 19–26 [Online]. DOI: 10.1109/PST.2013.6596032

[26] Rene Ritchie, 'History of iPad (original): Apple makes the tablet magical and revolutionary | iMore', 06-Oct-2014. [Online]. Available: http://www.imore.com/history-ipad-2010. [Accessed: 16-Nov-2014]

[27] 'Android vanligaste smartphone i Sverige – men vi surfar med Iphone', *Ajour*. 20-Jan-2015 [Online]. Available: http://www.ajour.se/android-vanligaste-smartphone-i-sverige-men-vi-surfar-med-iphone/. [Accessed: 20-Jan-2015]

[28] 'Svenskar använder smartphones flitigast', *Dagens Media*, 20-Jan-2015. [Online]. Available: http://www.dagensmedia.se/nyheter/mobilt/article3634818.ece. [Accessed: 20-Jan-2015]

[29] 'Google ändrar hur man mäter Android-statistik', *Feber / Android*, 20-Jan-2015. [Online]. Available: http://feber.se/android/art/268842/google_ndrar_hur_man_mter_andr/. [Accessed: 20-Jan-2015]

[30] 'Genymobile | Let IT be Mobile'. 17-Jan-2015 [Online]. Available: http://www.genymobile.com/. [Accessed: 17-Jan-2015]

[31] 'SingleClientConnManager | Android Developers', 17-Jan-2015. [Online]. Available: http://developer.android.com/reference/org/apache/http/impl/conn/SingleClientConnManager.html. [Accessed: 17-Jan-2015]

[32] 'How time of day affects content performance - iMediaConnection.com', 17-Jan-2015. [Online]. Available: http://www.imediaconnection.com/content/31577.asp. [Accessed: 17-Jan-2015]

[33] 'ESSA', 18-Jan-2015. [Online]. Available: https://www.essa.org.au/. [Accessed: 18-Jan-2015]

[34] 'ACSM | Medicine & Science in Sports & Exercise', 18-Jan-2015. [Online]. Available: http://www.acsm.org/access-public-information/acsm-journals/medicine-science-in-sports-exercise. [Accessed: 18-Jan-2015]

[35] *Psychology of Sport and Exercise*. 2015 [Online]. Available: http://www.journals.elsevier.com/psychology-of-sport-and-exercise/. [Accessed: 18-Jan-2015]

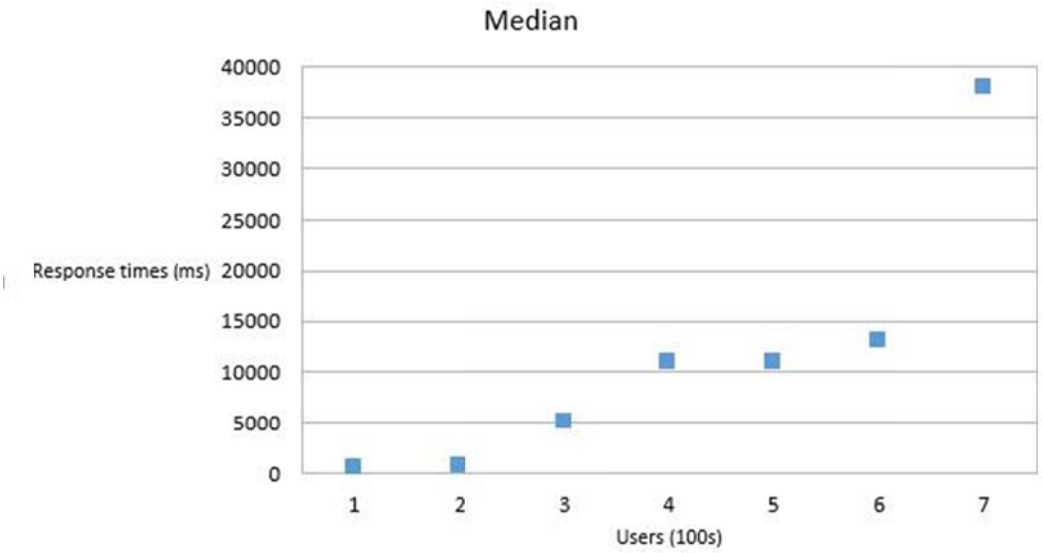
[36] K. Zickuhr, 'Location-Based Services', *Pew Research Center's Internet & American Life Project*. 18-Jan-2015 [Online]. Available: http://www.pewinternet.org/2013/09/12/location-based-services/. [Accessed: 18-Jan-2015]

## Appendix A: Detailed results

The table below presents all the results (i.e., including both from the tests with the smartphone and the emulator and Locust). Only tests with responses are shown. All of the results and median values are in ms.

| Smartphone | Result | Emulator | Result | Locust | Average | Locust | Median |
|---|---|---|---|---|---|---|---|
| 100 | 1210 | 100 | 1302 | 100 | 1647 | 100 | 420 |
| | 1451 | | 1380 | | 956 | | 470 |
| | 1534 | | 1648 | | 438 | | 410 |
| | 1627 | | 1611 | | 654 | | 590 |
| | 1182 | | 1583 | | 635 | | 510 |
| 200 | 1320 | 200 | 1422 | 200 | 1378 | 200 | 1100 |
| | 1269 | | 1499 | | 2867 | | 670 |
| | 1289 | | 1419 | | 843 | | 550 |
| | 1288 | | 1420 | | 13036 | | 10000 |
| | 1364 | | 1443 | | 3100 | | 770 |
| 300 | 1189 | 300 | 1310 | 300 | 6544 | 300 | 6100 |
| | 1165 | | 1281 | | 5747 | | 4900 |
| | 1338 | | 1431 | | 5464 | | 5100 |
| | 1326 | | 1471 | | 7294 | | 6800 |
| | 1210 | | 1342 | | 3745 | | 3500 |
| 400 | 1160 | 400 | 1277 | 400 | 14615 | 400 | 13000 |
| | 1217 | | 1324 | | 11152 | | 11000 |
| | 1204 | | 1343 | | 10340 | | 9800 |
| | 1310 | | 1411 | | 12761 | | 12000 |
| | 1177 | | 1363 | | 11759 | | 11000 |
| 500 | 1303 | 500 | 1449 | 500 | 9441 | 500 | 8900 |
| | 1296 | | 1442 | | 11472 | | 11000 |
| | 1220 | | 1411 | | 18253 | | 15000 |
| | 1278 | | 1407 | | 10844 | | 9500 |
| | 1314 | | 1418 | | 12983 | | 11000 |
| | 1240 | | 1396 | 600 | 16621 | 600 | 17000 |
| | | | | | 12870 | | 14000 |
| | | | | | 14468 | | 12000 |
| | | | | | 12648 | | 13000 |
| | | | | | 12444 | | 13000 |
| | | | | 700 | 40754 | 700 | 28000 |
| | | | | | 29903 | | 28000 |
| | | | | | 29039 | | 96000 |
| | | | | | 99502 | | 63000 |
| | | | | | 67719 | | 38000 |
| | | | | | | | |
| Users | 100 | 200 | 300 | 400 | 500 | 600 | 700 |
| Median (smartphone) | 1451 | 1289 | 1210 | 1204 | 1287 | | |
| Median (emulator) | 1583 | 1422 | 1342 | 1343 | 1414.5 | | |
| Median (median, Locust) | 470 | 770 | 5100 | 11000 | 11000 | 13000 | 38000 |

The median value of the median values that Locust gave are shown below. These values are in the last row of the table above.



TRITA-ICT-EX-2015:10