Optimizing Enterprise Resource Planning systems for mobile applications

KONSTANTINOS VAGGELAKOS

KTH Information and Communication Technology

Degree project in
Communication Systems
Second level, 30.0 HEC
Stockholm, Sweden

Optimizing Enterprise Resource Planning systems for mobile applications

Konstantinos Vaggelakos

Master of Science Thesis

Communication Systems
School of Information and Communication Technology
KTH Royal Institute of Technology
Stockholm, Sweden

May 30, 2013

Examiner: Professor Gerald Q. "Chip" Maguire Jr.

Abstract

Growing enterprises have growing amounts of information. Making sure that this information is easily accessible to its employees is not an easy task. For this task Enterprise Resource Planning (ERP) systems come in handy. However, the ERP systems contain lots of information and may be too complex to handle this information or may be too slow in sharing this information within the organisation. This thesis introduces a new solution for how mobile applications can connect to an ERP system by exploiting database synchronisation, which allows the employees to get the information wherever they are without the need to directly access the ERP system.

There are three important questions in this thesis: how the ERP works, how to build a mobile application and, how to build a back end? By combining answers to these questions a whole system is built which is production ready and can copy data between the ERP system and the back end and show the information in a mobile application. The overarching goals were to build a system that could extract data from the ERP system into a proprietary back end and a mobile application that could synchronise with the back end. However, synchronisation between the mobile application and the back end was not implemented due to lack of time. The back end had to be able to scale up to 500 concurrent users and respond within 3 seconds, both of these goals were achieved. The mobile application that was built to display the information to the end user was built with usability in mind, since Netlight wanted a straightforward application that anyone could use. The mobile application was shown to have good usability.

The results of this thesis project show that building systems around ERPs, instead of inside them, gives these systems the ability to scale, improved the implementation time, and reduced the company's maintenance efforts.

Sammanfattning

Växande företag får mer och mer information. Att kunna se till att den informationen blir enkelt tillgänglig för alla inom företaget är inte nödvändigtvis lätt. Det är ofta det som affärssystem kan användas till, dock innehåller affärssystem väldigt mycket information och kan vara för komplexa för att enkelt kunna hantera information man är intresserad av. Det kan även uppstå problem i prestanda i och med storleken på affärssystemet. I det här examensarbetet föreslås ett nytt sätt för hur mobila applikationer kan integreras med affärssystem genom att synkronisera mot dess databas, vilket tillåter anställda att komma åt informationen vart de än befinner sig

I det här examensarbetet finns det tre olika delar som är intressanta, hur affärssystemet fungerar, hur man bygger en mobilapplikation och hur man bygger ett back end. Genom att kombinera kunskapen från ovan nämnda delar, byggdes systemet som är redo för produktion och kan synkronisera data från affärssystemet till back endet, samt att visa upp informationen i mobilapplikationen. De översiktliga målen var att bygga ett system som kunde extrahera data från affärssystemet till ett eget byggt back end och en mobilapplikation som kunde synkronisera med detta back end. Dock blev synkroniseringen mellan mobilapplikationen och back endet aldrig implementerat. Back endet skulle även kunna skala upp till 500 samtidiga användare och då kunna svara inom 3 sekunder, vilket man lyckades med. Mobilapplikationen som byggdes för att visa information byggdes med användbarhet i tankarna, eftersom Netlight ville ha en enkel mobilapplikation som vem som helst skulle kunna använda. Mobilapplikationen analyserades fram till att vara användarvänlig.

Detta examensarbete visar på att det går att bygga system runt affärssystem istället för att bygga dem i affärssystemen, vilket möjliggör att systemet kan skala upp bättre, mindre tid för implementation samt mindre underhåll.

Acknowledgements

First of all I would like to thank all the people that have been helpful in the making of this thesis project.

I would like to especially thank my two supervisors at Netlight, Håkan Andersson and Per Carlsson for putting a lot of effort into helping me and making sure that the project was successful.

Last but not least I would like to thank Professor Gerald Q. "Chip" Maguire Jr. for helping me form my report and giving me great feedback.

Contents

List of Figures

List of Tables

List of Acronyms and Abbreviations

ACID	Atomicity, Consistency, Isolation, Durability
API	Application Programming Interface
CPU	Central Processing Unit
CRUD	Create, Read, Update, Delete
ERP	Enterprise Resource Planner
HTTP	Hypertext Transfer Protocol
IT	Information Technology
JSON	JavaScript Object Notation
LDAP	Lightweight Directory Access Protocol
MSSQL	Microsoft SQL
MVC	Model View Controller
NPM	Node Packaged Modules
ORM	Object-Relational Mapping
RAM	Random Access Memory
RDBMS	Relational Database Management System
SQL	Structured Query Language
SSIS	SQL Server Integration Services

SWOT	Strengths, Weaknesses, Opportunities, and Threats
UI	User Interface
URI	Uniform Resource Identifier

Chapter 1

Introduction

This chapter begins in section 1.1 with an introduction to the problem considered in this thesis and gives an overview of the context in which this project was done in section 1.2. Section 1.3 introduces the specific problem that is addressed in this thesis project. The goals for the thesis project are described in section 1.4. In section 1.5, the scope of the thesis project is delineated. Section 1.6 describes the structure of this thesis.

1.1 Overview

This thesis project was conducted at Netlight, a consulting firm. Netlight has their roots in information technology (IT) consulting and provides consultants for different kinds of problems within IT. Netlight has grown as a firm and has expanded globally. Growing as a company has its advantages and disadvantages. Given Netlight's way of managing their consultants, one can clearly see that it is always important to help each consultant and keep the company (logically) together, rather than scattered. Netlight values these two objectives very highly. However, as a growing company, it is getting harder and harder to keep track of the employees and to know what coworkers are doing.

Consulting firms tend to send consultants to do external jobs and then have problems backing them up with the competence of the whole firm. As companies grow, it is known to be harder to keep the company manageable and to keep the company's identity. This is due to the fact that there are multiple entities to keep track of. Keeping better track of these entities or a subset of them is something that was desired by Netlight.

To accomplish this Netlight wanted a mobile application to help their employees keep track of each other and to enable employees to know what other employees are up to. To achieve this and to implement it in practice, there are three things that need to be taken into consideration. The first is the mobile application itself, which will be used by the employees to keep track of their coworkers. The second is the back end, which will server the application with the data. The third is the information source where information is gathered in order to later be sent to the mobile application. This thesis project will go into the details of each of these three parts, but will focus primarily on the back end (more about this in section 2.4) and the connection between this back end and the actual information source.

1.2 Problem Context

Current trends and technologies, include increasing adoption of cloud computing, smartphones, and a combination of both. Mobility has become a much higher priority than it was earlier, as can be seen in a recent Cisco report[1], where it is shown that there has been an increase in mobile internet usage over time. In combination with this trend there is also an increase in the size and adoption of cloud computing infrastructures [2], which implies that mobility and intelligence, along with access to information on the fly are needed *wherever* employees are and *whenever* these employees need access to this information. Scalability is also an issue in rapidly growing enterprises, hence this is another property of the cloud that we desire.

To build something for the future it is important to examine the existing trends and consider how these trends will affect a specific project over time. This implies that the project as a whole needs to be cognisant of the relevant trends. However, this is not necessarily a major problem, since the mobile application itself already has mobility we need to design and implement a system keeps up with the predicted trends. Therefore the most important focus in this thesis project is how the information source and back end can scale together.

The information source in this case is an Enterprise Resource Planning system (also called ERP). However, ERP systems as built today do not necessarily meet the expected future demands, especially when it comes to supporting mobility and scalability.

1.3 Problem description

ERP software handles information, which spans across different functions within an enterprise [3]. However, utilising an ERP system to handle all information and communication may not be the best way forward for an enterprise. As mentioned previously there is a paradigm shift in the way we want to work today compared with earlier years, were we see more mobility and more of cloud concepts. Therefore, it is interesting to understand how an ERP should operate with respect to this paradigm shift and how the ERP system impact on the business can be improved.

ERP systems often consist of huge databases filled with all sorts of information. This large collection of information has its advantages and disadvantages. One of the most obvious advantages is that business intelligence is stored in one place, therefore integration of multiple different systems is not needed when handling intelligence which would otherwise span across different systems. However, keeping everything in one huge system makes it much harder to manage this information and providing the relevant information to an employee within the company is increasingly difficult. This implies that even though the ERP systems stores everything in one database, only a small fraction of the information is needed at any point in time. To query an ERP for a small piece of information is probably going to be increasingly difficult due to the fact that the ERP's database is normalised (more about this in Chapter 2). Due to the fact that the ERP database is normalised and consistent, there may be problems with scalability and minimising response times. Another important factor when it comes to the ERP system's databases is that many ERP systems are pretty old and do not have or do not use techniques to optimising information searching.

1.4 Goals

The overall goal of this thesis project is to build a system to be used by Netlight's employees in order to get information from the ERP system. Although the project has many different goals, the most important goals will be clearly identified. The focus of this thesis project will be directed towards these important goals.

The goals are divided into different subgoals which associate to each of the three major questions. Below is a short description of each goal

and its subgoals:

1. **Core functionality** The system should fulfil the requirements set by Netlight. The core functionality is defined in the requirements specifications that can be found in Appendix A. The requirements specified in this requirements document have different priorities. All of the the level 1 priority requirements are essential for this thesis project and will be implemented.

2. **Build for the future:** The system should be built for the future, assuming that the current trends continue. For this goal it is interesting to understand what the current trends are and consider how these trends can be addressed in this thesis project.

3. Mobile application

   (a) **Easy to use:** This goal is very important to achieve in order for the actual result to be useful. The idea is that the prototype developed during this thesis project will be good enough to deploy into production. This goal includes understanding what ease of use implies and how it can be measured.

   (b) **Synchronise the application with the back end:** Implement a synchronisation mechanism that will be used by the application to talk to the back end. Since the information source (the ERP system), holds all the data, it is expected that most of the communications traffic will flow from the information source to the mobile application. Therefore, the proposed synchronisation mechanism will cache data in the mobile application. The subgoal includes understanding how synchronisation can support the expected usage scenarios and how to evaluate if the user's performance is increased or decreased as a result of the use of their mobile application.

4. Back end goals A back end should be implemented with the following properties:

   (a) **Scalability:** Since the results of this thesis project might be interesting to corporations larger than Netlight and since the whole project has a goal to build for the future, scalability is essential. To build a scalable system it is important to

first define what this implies and to understand how scalable system are built today.

(b) **Speed:** The back end will serve a mobile application used by many users, each of which have to be served within a bounded time frame. The exact time duration will be investigated further. It is important to understand how long a user will wait before quitting the application.

5. Information source goals

(a) **Get the desired information:** The information source (in this case an ERP system) will have all information that is needed for the system to be built in this thesis project. Therefore, it is important to investigate what alternative methods can be used to extract the relevant data from this ERP system. This subgoal includes understanding the differences between these alternatives and measuring them in order to determine which is most effective for the expected use cases. It is also important to investigate, what different architectures could be utilised when extracting the data, especially those that can do so quickly.

(b) **Integration with the ERP system:** ERP systems are used in different ways within enterprises, thus it is interesting to understand how the proposed synchronisation application can be integrated [1] with the ERP system since these systems hold all of the relevant information. This will require understanding how an ERP system works and what the integration options are. Investigating and solving these issues will give us a clear picture of how the system could be built in the most appropriate way.

1.5 Scope

The overall scope of this thesis project includes each part of the project as described above, therefore there is a different scope for each part of the project. We go into details of how we can extract information from an ERP

[1] The term integration in this thesis will imply that two systems are unified even though information does not necessarily flow in both directions.

system, including the different ways of copying data to other systems. Measurements will be done of the different ways of extracting the desired information from the ERP system and how to transfer information to the back end. The result of these measurements will indicate how the prototype should be integrated into the particular ERP system used at Netlight, by identifying the best alternative for the particular case of creating a back end to serve the prototype mobile application. This thesis will go into depth concerning each of the different methods and provide information about the different methods, especially their advantages and disadvantages. The scope of this thesis includes suggesting a way to integrate with ERP systems in order to extract information for these mobile applications, while accommodating the expected current trends. This thesis project will not go into details about different kinds of ERP systems, but rather will give a practical view of how one specific ERP system works in this case examined within this thesis project.

1.6 Structure of this thesis

Chapter 1 describes the problem and its context. Chapter 2 provides the background necessary to understand the problem and the specific knowledge that the reader will need to understand the rest of this thesis. Chapter 3 describes the goals, metrics, and solution proposed in this thesis project. Chapter 4 depicts the results from this thesis project. The solution is analysed and evaluated in Chapter 5. Finally, Chapter 6 offers some conclusions and suggests future work.

Chapter 2

Background

This chapter will introduce the underlying theories which this thesis project is based on. Section 2.1 briefly introduces related work which this thesis project touches upon. Section 2.2 describes some relevant current trends. Section 2.3 presents some important information about how to build a successful mobile application by understanding user behaviours. Section 2.4 presents a number of different technologies and paradigms that are essential to understand in order to know how the back end should be built. Section 2.5 explains what an ERP system is and how synchronisation can be done between an ERP system and another system. Section 2.6 explains some important programming paradigms that are used in the creation of the prototype solution proposed in this thesis project.

2.1 Related work

This section provides information about two related works, one master thesis made by Shitian Long in section 2.1.2 and one product called OpenMobster in section 2.1.1.

2.1.1 OpenMobster integrations platform

OpenMobster is an open source framework that provides means for building a back end to connect mobile platforms directly to an ERP system [4]. This framework minimises the development costs for how to get the data in the mobile phones, since it provides methods for the most common operations such as create, read, update and delete

(CRUD). OpenMobster also provides a framework for the clients side, which include the most common mobile platforms. There are a couple of advantages that come with the use of a framework such as OpenMobster:

Saves development time. The time spent on developing a back end and the communication between back end and the mobile application can be reduced, since most of the needed development already exists in the framework.

Includes synchronisation. The OpenMobster frameworks provide means for synchronising the data from the back end to the mobile application. This makes the mobile application usable offline, by utilising its local store.

Push notifications. By telling the mobile application when new data exists, this framework makes sure that no unnecessary time is spent on polling for new information.

2.1.2 Synchronisation mechanisms

Shitian Long wrote a master's thesis about database synchronisation between devices[5]. In his thesis the focus was on designing a repository based SQLite database synchronisation mechanism. The proposed synchronisation mechanism supported rollbacks since it was repository based and required minimal computational power since the synchronisation was not done in real time. His conclusions can be useful, depending on the requirements set for the synchronisation mechanism in the context of our thesis project. Nevertheless, his thesis was an inspiration with regard to understanding and defining *synchronisation patterns* (as seen in section 2.3.3.1).

2.2 Current trends

Current trends are used to understand how the future might look and what can be done to exploit these trends. Accurately predicting the future is very important from a market analysis perspective. Building a product that will actually generate a positive return requires knowledge of the current trends and how these trends might be used to predict the future. When one understands where the market is heading, then a plan can be

made to follow the expected trends and thus maximise your return. The details of doing so will depend on what return is required or desired. This section will introduce two trends that are important for this thesis project, in order to better grasp where these current trends might be heading. These two trends are mobility and scalability.

2.2.1 Mobility

Mobility is something that has always been appreciated, however until recently wireless connectivity was not sufficient in many areas for users to expect connectivity where ever they are. However, today users expect that they can be mobile and still use their applications. Mobile smartphones have given mobility a new face, as almost anything can be done from such a mobile device nowadays. This has paid off for many users find that their mobile device is useful. Internet usage via mobile phones, as shown in KPCB's analysis of internet trends [6], continues to increase. There is also a trend toward more and more mobile broadband subscriptions, which suggests that people who have a mobile phone want to use this phone to access the Internet, rather than simply to talk or utilise short messages. Between 2010 and 2011 there was an increase in the number of active mobile broadband subscriptions from 11.2% to 15.7% [7]. The trend in mobile phone broadband subscriptions suggests where we are heading in the near future.

2.2.2 Scalability

According to the statistics in section 2.2.1, the number of active mobile phone subscriptions and the number of mobile broadband subscriptions are both increasing. Based on these statistics, future systems might want to exploit this trend, hence it is important for these future systems to scale up in order to cope with all of the new mobile devices that are connecting to the Internet. It is important to provide scalability in a service so that more users can utilise the service and the service can continue to support the users. However, to understand scalability we must first define it. There are actually two common uses of the word scalability. One use of the term is that the system can handle more workload without adding any extra resources to the system [8]. The other use of the word concerns the ability to add resources to the system to handle the increased workload.

Having scalability ensures that the system will not fail as the system continues to grow, independent of the way this load grows. Having an assurance of scalability, can reduce the risk of a system, product, or service. Therefore developers must consider how their system, product, or service might behave when it is scaled (up or down). It is important to keep in mind that scalability cannot be calculated, unless the system actually scaled (up or down), which implies that you already know how it will behave. Below are some guidelines that can be followed in order to get a hint of how well a system scales [8]:

1. Looking at the performance of different resources as the load varies. Characterise the performance as O(n), O(n^2), O(n log n), etc. We can learn how the system scales by checking how adding (or removing) resources changes the system's response time and if the systems scales with addition (removal) of resources as required.

2. Identify any bottlenecks in the mechanisms utilised by the system. Also identify if there is any scaling design architecture violations in the system.

3. Perform a Strengths, Weaknesses, Opportunities, Threats (SWOT) analysis on the system from a scalability point of view. Strengths are that growth would be handled well by the system. Weaknesses are that growth that would not be handled well by the system. Opportunities are how well the system can adapt to new technologies or handle changes in workload. Threats are how the system does not adapt to new technologies or handle changes in workload.

2.2.3 Database scalability

System administration has become a bigger problem than it was earlier. This is due to the fact that the number of users and their expectations are growing which can lead to an unpredictable spike in server utilisation due to correlations in the demands of these users. This problem was addressed in the case of serving static files such as music, video, and web pages by introducing Content Delivery Networks (CDNs) [9]. Sharing static content by distributing it across different CDNs ensures that spikes in requests for this content are handled better as there are multiple servers which can server these requests. CDNs can also be distributed geographically to better handle static content delivery across continents.

However, the web is becoming increasingly dynamic and provides increasingly personalised web pages for each user [9]. As a result there is an increasing need to serve dynamic content to these users. These personalised pages are frequently based on previous experience and user preferences. CDNs are not a good way to serve dynamic content since they simply cache static content. Therefore there is trend to scale databases to support applications which provide dynamic content.

In combination with the above, data volumes are growing faster than ever, with a 40% growth rate per annum (measured during 2011) [10]. In order for databases to meet such demands, these databases have to face new challenges such as those described in[10], specifically

1. **Scaling write operations.** Write operations are harder to handle than reading because reading does not modify the contents of the database, while write operations either have to be synchronised across instances of the database or one has to relax the Atomicity, Consistency, Isolation, and Durability (ACID) database properties - for example by allowing eventual consistence.

2. **Real-time user experience.** Scaling up databases implies challenges in how the servers communicate with each other. That is why the real-time user experience and expectations have to be considered when building systems.

3. **Continuous service uptime.** Availability is really important in today's web, without it huge losses can occur (where these losses can be financial or information losses).

4. **Reduce barriers to entry.** Increasing complexity in the database may result in the need for more complex application solutions. It is therefore important to keep in mind that the barriers to entry should still be kept low in order to attract application developers, this requires using more scalable database solutions.

2.3 Mobile application

This section explains the technologies that could be used for the mobile application that will be developed in this thesis project and to further investigate how the different parts contribute to a complete solution. For each major part (usability, back end connection, and local cache) there

is a description which gives a deeper understanding of the problems or challenges that may occur. This section will focus on mobile application development utilising Apple's iOS [11] for the mobile specific parts of the application. This operating system was chosen because Netlight specifically requested a mobile application running on iOS.

2.3.1 Usability

Usability is important as we wish to have the users feel comfortable accessing their information via a mobile phone, otherwise they will not make use of the application. There were 11.1% more mobile devices sold in 2011 compared to 2010 [12]. Since users have more and more information to take in they need to improve how they do so. In addition, more and more people are using their mobile phones to browse for information. However, a mobile phone introduces a lot of challenges in terms of showing this information in a suitable way. As a result usability has become a major requirement for mobile applications.

The following paragraphs will go into greater depth concerning some of the guidelines described by Krug in [13]. These guidelines are based only on Krug's view of usability. Some argue that Krug's view of usability is narrow and does not consider the fact that different people with different backgrounds (such as culture and language) might think or see things differently [14].

2.3.1.1 Do not make the user think

The first and most important guideline mentioned in [13] is to not make users think too much. Here it is important to understand that everything has to feel almost obvious, even to someone that has not seen anything like it before. The question is then, what makes us think? Or what makes us think for too long? The selection of words that are used in an application are very important. It is important to allow the user to understand the information that the application want to convey to the user. However, at the same time it is also important that the message is concise and does not make the user think twice before understanding. A good example presented in [13], is a button for job listings. Where there are three button labeled: Jobs, Employment opportunities, and Job-o-Rama. From left to right these increase in requirement for thoughts, which might not be necessary. It is important to reduce the thinking

required for each piece of information that is presented. This applies not only to labels, but to the design elements themselves, such as user interface elements (UI elements). UI elements need to have a clear distinction between them so that the user is not confused with how to navigate within the application.

Taking into account the things mentioned above it is important to understand that it might be impossible to have everything be self-evident. For instance if the application introduces new things it is important that those things are at least self-explanatory [13]. This means that the information should be nearly instantaneous recognisable.

2.3.1.2 How we really use the web

According to a study made by Nielsen Norman Group [15] 79% of users always scan any new page of information instead of reading it word by word. The question is why do users skim through information, rather than read the whole of it? One explanation is that users are often in a hurry, which implies that they try to save time. Krug has said: "Web users tend to act like sharks: They have to keep moving, or they will die" [13]. This behavior might be something that most designers do not realise, however it seems to be important as previous research has shown. There is another optimisation, users only read what they actually need to read. This is based upon the assumption that there is no or little value in reading something that the user does not consider to be interesting. As a result users scan information that they find interesting. This scanning and prioritising is something that the users are actually good at according to Krug, however this is not proven in his work [13].

According to Krug users choose the first reasonable choice, and not the most optimal choice. This is also known as a *satisficing* strategy, a combination between satisfying and sufficing [16]. The satisficing strategy in the case of usability implies that the user tries the first reasonable choice, rather than the optimal choice. This is due to the fact that making the wrong choice generally does not have a severe penalty to the user, in the event of a wrong choice the user simply goes back and tries again. A possible explanation for this behavior could be that users do not really care to understand how things works as long as they work. When they do work, then the user tends to stick with their earlier choice.

2.3.1.3 Designing pages for scanning not reading

Having a clear visual hierarchy implies that relationships are clear between the elements in the UI. It is important to understand what elements belong and what element do not. Below are three important points Krug suggest in order to have a clear visual hierarchy [13]:

1. **The more important the more prominent.** The more important headings should for instance either be larger, bolder, in some distinctive colour, or set off by more white space.

2. **Logical relationships are also visual ones.** Group things that are similar by putting them together under a heading or in a well defined area.

3. **Nesting visual components to show hierarchy.** Just as in newspapers, in a UI there are headings with subsections, which are all part of a bigger hierarchy making it easier for the user to find what they need.

Without considering the above mentioned points, all text would be the same and would have the same spacing around it. What this means in practice is that when a user scans for information, they need to be able to identify what is important, hence try to understand how things are organised.

2.3.1.4 Omit needless words

According to William Strunk Jr. in his book "The Elements of Style": “Vigorous writing is concise. A sentence should contain no unnecessary words, a paragraph no unnecessary sentences, for the same reason that a drawing should have no unnecessary lines and a machine no unnecessary parts” [17]. There are often more words than needed in most user interfaces. Getting rid of the needless words can give the following effect:

- Reducing the noise level of the UI.

- Makes the useful content more prominent.

- Makes the UI more compact, which allows the users to see more useful information at the same time.

One way to recognise needless words is to look for something called happy talk [13], which is a paragraph of text that starts with something like "Welcome to ...". Happy talk is just like small talk, in other words it is content free, the main purpose is to fill up space or time. However, users do not have time for either small talk or happy talk, therefore these kinds of text should be avoided.

Instructions are also needless if the application itself is self-explanatory. Therefore one should always strive towards instruction free applications, however if there is a need to explain something, then the explanation should be kept to a bare minimum.

2.3.2 Back end connection

In order for the mobile application to be able to get the information that the application will display to the user, the application needs to get that data from somewhere. This is where the back end connection module comes in. There are different ways of getting data into a mobile phone. No matter which method is actually used there are a couple of important issues that have to be considered. Since the mobile phone has limited battery power and limited connectivity it is extra important to optimise the communication with the back end. A study made by Aaron Carroll and Gernot Heiser, shows that the most power consumption in all phones tested, occurred during network connectivity or phone calls, and that the amount of power consumed depended on the phone [18]. Therefore it is important that the mobile application's back end connection is optimised in terms of the time used and the amount of data that is transferred. The back end will remove all unnecessary spaces (also called minifying) and will only send relevant data, but may include pointers to where new or additional data can be fetched.

As mentioned previously there are different ways of getting the information into the phone, such as representational state transfer (REST) web service (more about REST services in section 2.4.5) The two different communication patterns that will be introduced are request-response and synchronisation patterns. The subsections below give a description of both, together with their advantages and disadvantages according to [19].

2.3.2.1 Request-response pattern

The request-response pattern is used in an on demand fashion. When information is needed it is requested and the requested information is provided in a response.

Advantages. The main advantage of the request-response pattern is that the mobile application only fetches the information which is needed for the present view. In other words, the data is optimised to the current view and whenever the user wants new information, i.e. when switching views, the mobile application will send a new request and get the corresponding data back in a response. Another big advantage is that the mobile application can always be served the latest information from the back end.

Disadvantages. One of the biggest disadvantages is that the request-response pattern requires both the mobile application and the back end to have connectivity. Therefore any offline action needs to be cached in order to work, however such caching belongs more to the synchronisation pattern.

2.3.2.2 Synchronisation pattern

In a synchronisation pattern while the mobile application has connectivity to the back end it will synchronise its local copy of all information needed by the mobile application. This implies that the mobile application keeps all information locally. To be able to manage this information locally there needs to be a good structure of the data internally in the application.

Advantages. The main advantage of utilising a synchronisation pattern is that the application can be used offline, since all information is stored locally. Keeping all information locally implies that connectivity is somewhat optimised, which is an important factor for mobile applications as previously mentioned.

Disadvantages. Since the information is stored by the mobile application itself, there is a need to locally manage and interpret this information. The mobile application must take some of the responsibility for handling this information, which could have been done by the server when utilising a request-response pattern. Another big disadvantage is the fact that the synchronised information might be

out of date, depending on the information. Having a synchronisation pattern complicates the back end communication, but in some cases might have a greater advantages than disadvantages.

2.3.3 Local cache

Keeping a local cache in the phone would utilise the synchronisation pattern described in section 2.3.2. A local cache implies keeping all the data locally in the storage of the mobile phone. As mentioned in the description of the synchronisation pattern, there are downsides to keeping a local cache. This section will try to describe these problems further and investigate how they can be solved, for the mobile application. Most of the information for these following sections is based upon the master's thesis of Shitan Long [5].

2.3.3.1 Synchronisation fundamentals

The synchronisation process maintains data consistency among sources and targets. To do this there are different paradigms and methods. However, we are only interested in one of these paradigms, specifically synchronisation between the mobile application and the back end, when many sources have one target with which they wish to remain synchronised, as described in [5]. This paradigm is depicted in figure 2.1.

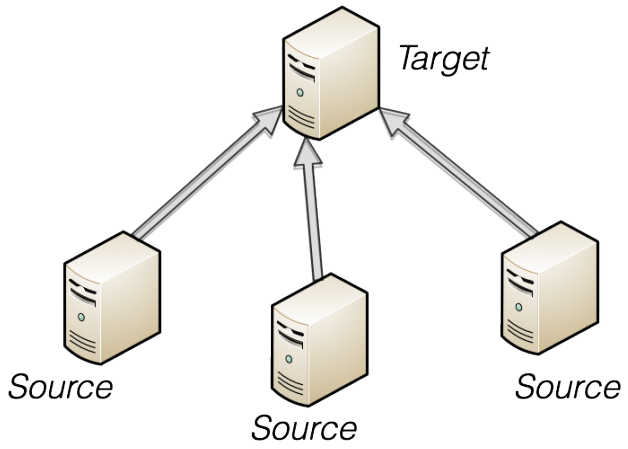

Figure 2.1: Fundamental view of data synchronisation model

The paradigm of one target and multiple sources is very common when it comes to code versioning. In the case of a source code control system most developers have a very clear picture of what the source is and what the target is. The sources can be seen as clients and the target is the server. This implies that the clients synchronise with the server rather than the opposite.

2.3.3.2 Fast synchronisation

Fast synchronisation is often used when there is only one source. In fast synchronisation the target uses synchronisation flags or time stamps, in order to mark what has been edited since the last synchronisation with the source. When a source performs a fast synchronisation against its target, it starts by comparing the synchronisation flags or time stamps to see what differs. Depending on what kind of changes there are (additions, modifications, deletions, etc) the source will behave accordingly. After the synchronisation is complete, the synchronisation flags will be reset, in order to prepare for the next synchronisation. In order for this method of synchronising to work there needs to be extra features added to the target to be able to handle multiple sources. This is necessary because the fast synchronisation mechanism needs to keep track of the synchronisation flags for all sources.

2.3.3.3 Slow synchronisation

Slow synchronisation occurs when the sources copy the target's database to compute the differences. This method is of course not the fastest, as the name suggests. However, this method might be useful in some cases, especially where there are multiple targets and/or sources. Also this is more useful when many changes where made in between each synchronisation. To make this as efficient as possible the device with the most CPU power or CPU time should calculate the difference, thus making the computation as fast as possible. This method of synchronisation might need optimisation in order to be feasible as it is not optimal with regard to network traffic and processor utilisation.

2.3.3.4 Database transaction log

Synchronisation based on a database transaction log can be done by sending the sources the transaction log since the last time they synchronised

with the target. This way only changes will be sent across the network and only the updates will be applied to the source or target. However, this requires additional storage for storing the transaction logs on the source. If there are a few changes, then this approach has many advantages.

2.3.3.5 Repository based synchronisation

In repository based synchronisation the most common solution is to have one dedicated target that holds all the information about the synchronisation. When a device wants to synchronise it will contact the dedicated target (which might also be the central repository) get the changes and update its copy of the data. This method is primarily used for source code version controlling. It can be used with binary files as well, but often this is handled by updating the whole file. This method is not very suitable for a big database, i.e., treating the whole database as a single binary file.

2.3.3.6 Keeping a local database

In order to store all the data in the mobile application there is a need to keep the data in a very manageable way. Operations such as searching, filtering, and sorting are common when managing even local data, which implies that there needs to be some management involved to simplify the process. Core Data [20] is a framework provided inside the iOS Software Development Kit (SDK). This framework allows storing and management of data in a manageable way. Core Data allows for object oriented (more about the object oriented coding pattern in section 2.6.2) control of data. This means that all objects are handled as classes or models (more about models in the Model-View-Controller pattern in section 2.6.3).

Core Data handles all communication with a lower level persistent store, making this lower level storage invisible to the mobile application itself. Core Data is middleware between the models in the mobile application and the storage, which simplifies the mobile application as the application does not need to care about the details of the storage. Core Data can use a database, such as Sqlite3 [21], as a persistent store. In other words Core Data will set up and manage the lower level database, for example an sqlite3 database, including tables, indexing, associations, etc. Figure 2.2, shows how Core Data sits in between the application and the storage.

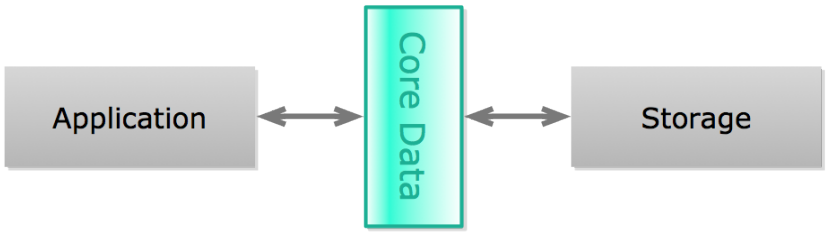

Figure 2.2: Core Data is middleware between the (persistent) storage and the application to make it simpler to handle data

2.4 Back end

This section explains and further investigates the technologies used in the back end. These sections will explain how the back end works and why this particular technology was chosen, as well as how this technology has been used in this thesis project.

2.4.1 Back end application server

The back end application server is the component that will provide the mobile application with information. The back end server will communicate with the actual database via a connection within the company's internal network. The back end server will be located behind a fire wall, but will be available to the users through the internet. None of the internal information sources used by the back end application server will be directly exposed to internet access.

The back end application collects the information that it needs and generates a response to the mobile application's request(s). The back end can be seen as a central element in the system. Figure 2.3 shows where the back end server is positioned within the whole system architecture.

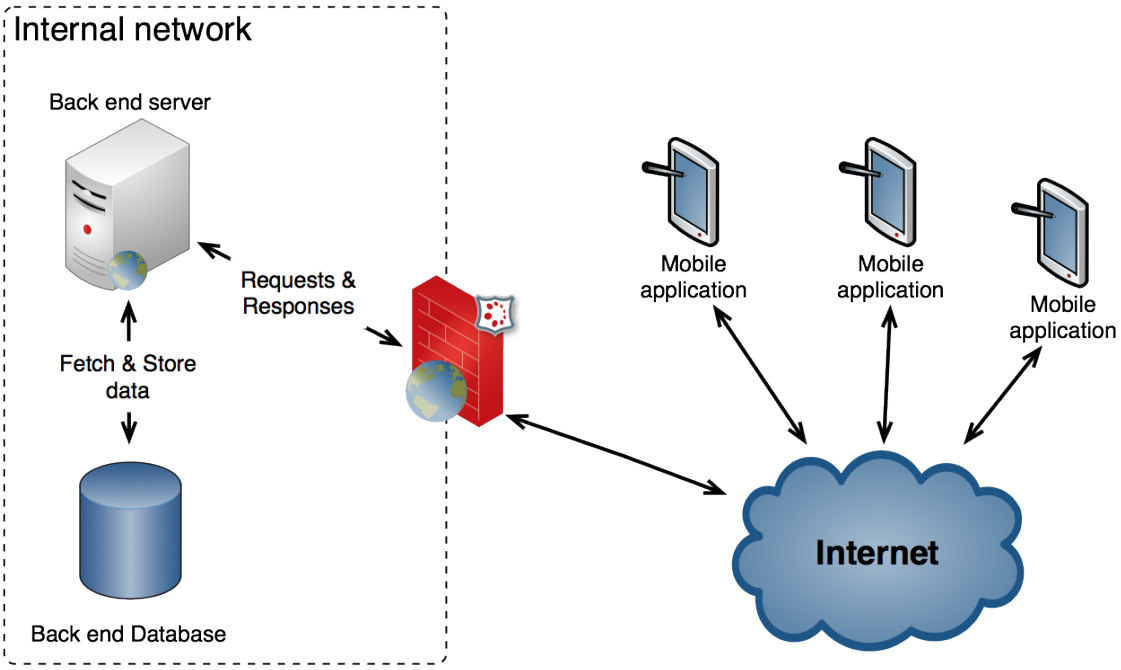

Figure 2.3: Back end architecture, inside the internal network.

Since the back end server will be responsible for responding to all clients it is important, that this server has scalability, thus the back end may need to be scaled up and if so this should be able to be done without problems. However, as mentioned in section 2.2.3 this may mean that the database supporting the back end has to be scalable as well depending on where the load of the system is. Figure 2.4 shows a back end which has been scaled up in order to meet higher load requirements, by either adding more back end servers, more database servers or both. What is important to understand here is that the application running in the back end server and the databases supporting this back end application with dynamic content, also needs to be scaled.

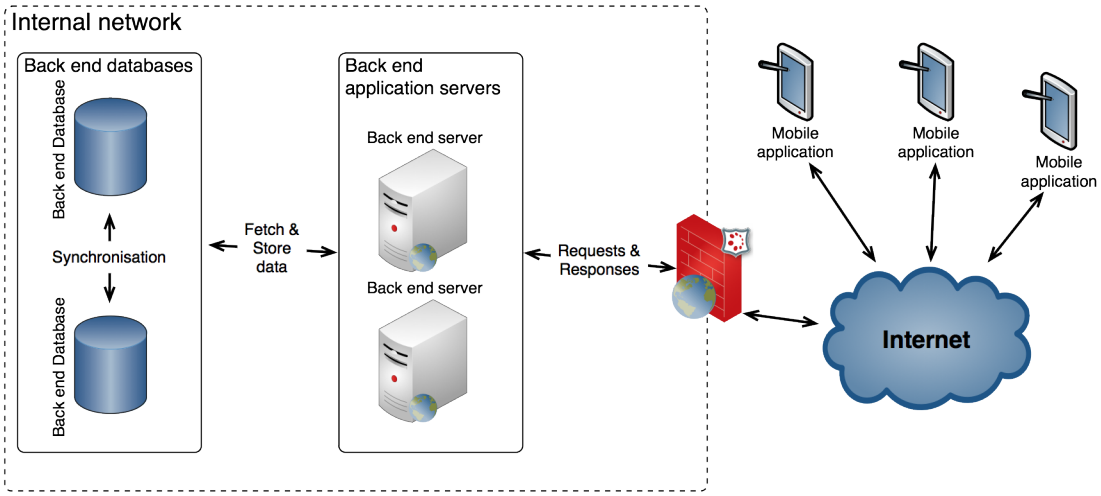

Figure 2.4: Scaled up back end.

2.4.2 Node.js application server

There are many different ways of writing an application back end. However, in this thesis Node.js [22] was chosen. Node.js is a server side scripting language used by Yahoo!, Microsoft, and many more. Below are some of the advantages with Node.js:

Simplicity. Node.js is based on javascript, which is the 11th most common programming language [23]. This implies that more documentation, more example code, and more people know javascript and they may be able to help during the development process or continue the project further.

Fast development time. Node.js comes with its own package handler called Node Packaged Modules (NPM). NPM provides an easy way to integrate open source components. With NPM and some javascript knowledge any developer can quickly start to develop with Node.js.

Performs well with many concurrent requests. Node.js is based on the event-driven programming paradigm and is by far the most successful language using the event-driven programming paradigm[24]. Event-driven computing has its advantages and disadvantages (more about this in section 2.6.5), however it is very

efficient for handling many concurrent requests which is required for our back end application.

Using javascript for the server side use is not very common, since javascript needs a javascript interpreter to run. Node.js provides a javascript interpreter (Google's V8 javascript engine) as well as various C libraries to be able to do system calls [25]. Node.js is essentially a wrapper which is written based on Google's V8 javascript engine. This way Javascript can be used in the back end as it can do the necessary system calls that the back end will need to do, such as opening a socket, writing to a file, reading from a database, etc.

2.4.3 Databases

There are different types of databases. Each of which have their own advantages and disadvantages, therefore it is important to understand the different types of databases that are available in order to choose the one that is the best for a given situation. In this section an introduction to a number of different databases will be given in order to give a better understanding of how they work and to identify when they should be used.

2.4.3.1 Traditional SQL databases

Traditional SQL databases or relational databases work in a very structured way and often go under the name Relational Database Management System (RDBMS). There are many different RDBMSs, however in this section the focus will be on the fundamental concepts that they all have in common. Below are some fundamental concepts that a RDBMS has compared to a file system[26]:

1. **Database schemas are not file sets**. A database schema defines relationships between entities, compared to a normal file set. SQL in it self does not define how the underlying storage is to be handled, but simply assumes that the physical storage is contiguous. Before SQL or RDBMSs the applications themselves had to handle relations between files, however today that is usually done in a database.

2. **Tables are not files.** Tables are not files, however they have similarities. A table is defined by a database schema and multiple

tables cannot have the same names just as in file systems, however tables are not physically stored in the same way that physical files are stored in a physical file system. This means that tables can be virtual (such as a database view, which is a virtual representation of a selection of information).

3. **Rows in a database are not necessarily records.** Rows in a database are not necessarily records, meaning that only the application can make sense of the information stored in a database row, thus making it a database record.

4. **Database columns are not necessarily fields.** The database has columns which only together with an application makes them into fields. Columns can have a computed value when accessed by an application.

2.4.3.2 NoSQL databases

Due to the need for scalable databases there have been a number of systems designed to provide horizontal scalability (i.e. scaling the database over multiple servers), unlike traditional databases which have little or no support for horizontal scalability. Most Not Only SQL or Not relational SQL (NoSQL) databases were designed with scalability in mind, which makes them interesting for this thesis project. Below are six key features of a NoSQL database [27]:

1. **Ability to scale horizontally.** The ability to scale horizontally, which essentially means distributing the database over multiple servers, is a key feature.

2. **Replication and distribution.** Able to replicate and distribute (also called partition) data over multiple servers.

3. **Call level interface.** NoSQL databases need another way of managing the data, which is not SQL.

4. **Weaker concurrency model.** Most NoSQL databases have a weaker concurrency model than the ACID transactions of a relational database. In other words there is no ACID guarantee with most NoSQL databases since updates only *eventually* propagate across different instances of the database.

5. **Efficient use of distributed indexes and RAM.** NoSQL databases need to have an efficient model for a distributed index and Random Access Memory (RAM) management.

6. **Dynamically add new attributes.** NoSQL databases should be able to dynamically add new attributes. This breaks the traditional relational model which is very structured compared to a NoSQL model.

There are different NoSQL databases with different ways of storing their data. Below is a list of three different types of NoSQL databases along with their advantages and disadvantages [27] [28]:

Key-value databases. Key-value databases store values together with an indexed key to be able to easily find every object that is persistly stored the database. Generally key-value databases provide functionality for replication, versioning, locking, transactions, and sorting. However, they only support one index or key and only one type of object.

Document databases. Document databases, store documents as defined by the application. Document databases provide indexing of all documents and a simple query mechanism. Unlike key-value databases, document databases provide support for secondary indexes and multiple types of objects within one database.

Graph databases. In graph databases the data is stored as a graph. Each node represents an entity (or object) and each vector represents a relationship. Both nodes and vectors have properties associated with them to define the node itself or the relationship between the nodes. Graph databases are advantageous to use when there are many relations between nodes, such as a social network or the network of all links that connect pages on the web. Graph databases allow an application to traverse from node to node, which means that all connecting information can easily be found. Compared to a relational database, where all relations between two entities are defined in different tables.

2.4.3.3 Sharding or horizontal splits

Sharding or also know as horizontal splitting of data is a technique to spread data across instances of databases. [29] Sharding is done

automatically by some databases such as MySQL cluster and MongoDB, where the data will spread across different instances of the cluster. The reason why one would want to shard their data onto different instances is the performance gain that might come with it. Scaling database servers is not an easy task, sometimes it is even impossible to keep the performance at a desired level. An important factor concerning sharding is how the data is partitioned. Since the data by definition is stateful, the partitioning logic has to be defined. Without suitable partitioning logic the database cluster will have problems adding nodes, as the data will need to be repartitioned.

2.4.4 Authentication

Today's enterprises make use of many internal systems. Most of these system require authentication since the information might be confidential. Authentication is quite a problem since it makes it hard to integrate all systems together. A solution to this is to have all authentication information in the same place, which is how authentication is handled in many enterprises today [30]. An example scenario of how this might look is depicted in figure 2.5. The back end server which is to be added to the internal system needs to utilise the existing authentication store, rather than keeping information about the users locally in its own database. The authentication store usually is a Microsoft Active Directory, which is based on the Lightweight Directory Access Protocol (LDAP). It is therefore important that the authentication component of the back end application supports integration with an LDAP store. Having a setup with one central authentication store, makes it easier for users to authenticate, since their authentication information is the same across all systems.

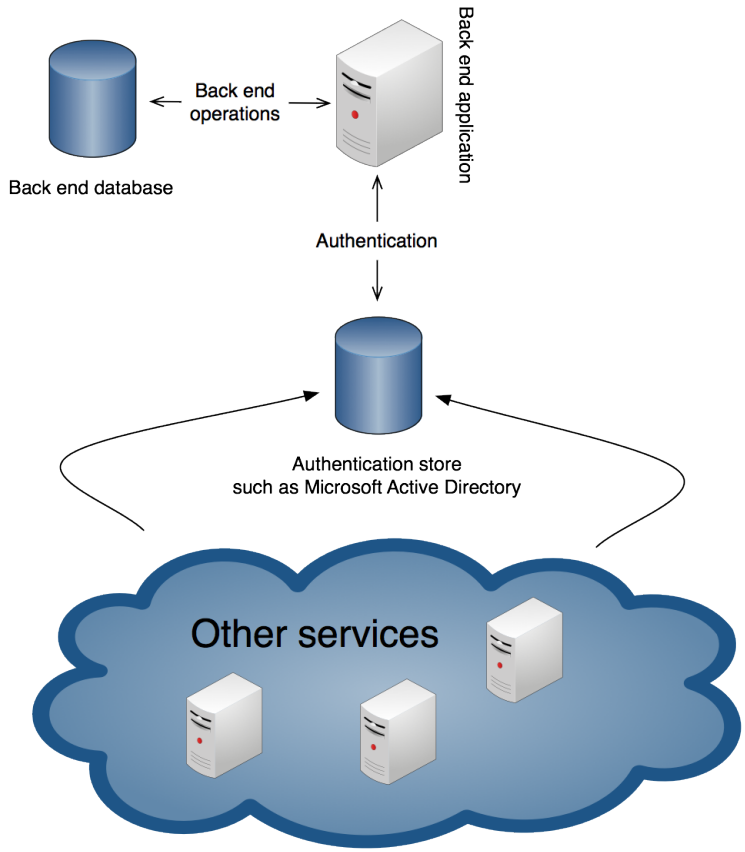

Figure 2.5: Figure of how local services depend on an authentication store to authenticate users.

2.4.5 Representational state transfer

Representational state transfer (REST) is a software architecture that outlines how different components of a distributed system interact with each other [31]. Two important terms in REST are the resources and their representations. Resources are objects of information that are requested by a client, whether the client is a machine or a human makes no difference. However, the representation may be changed depending on whether a machine or human is making the request, since machines interpret resources in a different way than humans. A resources in REST needs to be identifiable with an universal descriptors, therefore each resource has a universal resource identifier (URI). When a client wants to operate on a resource the client sends a request to the server with

a specific URI, which corresponds to a resource. The server will then, depending on the client, send back a representation of the resource.

An important aspect of web services that implement RESTful architectures is that upon receiving a resource, the client should have enough information from a response, to perform different actions of that resource on the server. It is important that the service provides hyperlinks in order to allow the client to move to the next stage.

Hypertext Transfer Protocol (HTTP) is a protocol which is often used in combination with RESTful web services. Using HTTP the client can specify the URI for the resource that it wants to operate on. HTTP supports different kinds of operations, which are then interpreted by the server to perform different actions. Below is a list of different methods, which are interesting for RESTful services and how they are described in RFC 2616 [32]:

GET. This method implies that the client wants to fetch information which is identified by the URI in the HTTP request. The information returned should be the value from the data-producing process as an entity.

POST. The POST method is used when the server is to accept a new subordinate entity as a new entity with the resource specified in the URI. However, it is up to the server to decide what to do with the information posted in the request to the server. If the server accepts the new subordinate entity, it is also assigned a URI.

PUT. When a client request contains a PUT message it is asking the server to store an entity under the specified URI. If the specified URI already exists this operation should be handled as an update to that entity.

DELETE. The DELETE method is sent by a client to requests that the server remove the entity located at URI.

The REST protocol will specify the communication between the components in the architecture. However, it does not specify how the data should be presented. When two machines need to communicate, the only important thing is the information itself. There are plenty of different data formats to specify how the data is formatted and interpreted by both sides. In this thesis the focus will be on a format called JavaScript Object Notation (more about JSON in section 2.4.5.1).

2.4.5.1 JavaScript Object Notation

JavaScript Object Notation (JSON) is a lightweight data format that is language independent, however it is a subset of the JavaScript language [33]. JSON has two structures: collections and ordered lists. A structure in JSON is basically a key-value pair, it is similar to an object in most programming languages. An ordered list is similar to an array in most programming languages and is essentially a list with collections.

The key-value pairs in a collections are separated by a colon and the pairs are separated by commas. Each collection is encapsulated within a left and right brace, whereas the lists are encapsulated in left and right brackets. Listing 1 shows an example with one array of items at the top level (using the square brackets) and two objects within that array using braces.

[H]

```
[
    {
        name: "Object 1",
        value1: "abc",
        value2: "def"
    },
    {
        name: "Object 2",
        value1: "123",
        value2: "456"
    },
]
```

Table 2.1: JSON example, which represents an array with two objects

2.5 Information source

This section explains the technologies and terms related to what is referred to in this thesis as the information source. Each section will provide deeper technological background information of topic that might be interesting or important for this thesis project.

2.5.1 Enterprise resource planning systems

What we know and what we do with knowledge is essentially what a business consists of. Intelligence is really important to keep secure and manageable, if an enterprise can not manage these two essential activities it is hard to provide value for their owners, which in turn might lead to a failure of the business. As companies grow in the normal course of operations one might expect that the amount of intelligence grows, which could imply that manageability becomes harder. If manageability becomes harder, inefficiency increases, hence the business is less competitive towards its competitors. Until recently many companies had many separate information systems that were not integrated with each other [34]. These systems typically focused on one specific business area only. For a company to efficiently achieve its goals it must share data among all of its business areas [34]. Leaving the task of sharing information between these separate systems to humans can be very inefficient, as it not only takes more time but also increases the risk of data errors. Therefore systems called Enterprise Resource Planning systems (ERP). These systems provide enterprises with a tool to manage their resources. Today it might seem obvious that businesses have software that manages all business areas in an integrated fashion. However, an integrated ERP system was not feasible until the 1990s because of its complexity in both hardware and software [34]. ERP systems evolved as a result of the following three things [34]:

1. Advancements in hardware and software technology;
2. The development of a vision where all information system are integrated; and
3. Companies structural shift from an functional focus to business process focus.

2.5.2 Exporting data from an information source

There are several different ways to export data from an information source such as an ERP system. This sections explains several of these different ways of exporting information and their advantages and disadvantages.

Since ERP systems store their information in a database, exporting data from an ERP system requires extracting the data from a database.

The choice of database might differ from ERP to ERP, however in this thesis the focus will be on an ERP named Agresso [35]. Agresso stores its information in a Microsoft SQL server, hence the focus will be on how to export data from a relational database such as Microsoft SQL Server [36]. Agresso stores its information in 1900 different tables and uses 300 views to be able to show the information that is usually requested. The database utilises no referential constraints and most of the identifying fields are characters rather than numbers. The structure of the database is not optimised for performance, as strings are longer than a numeric identifier could be, resulting in more time spent to comparing values when searching.

There are basically two ways of exporting data automatically: using a custom built application or using predefined tools for system integration. The next question is when the exporting of data needs to occur, based on the answer of this question, the exportation strategy might change. Below is a list explaining the advantages and disadvantages with the two approaches and how they might apply to different needs regarding when the export needs to occur:

1. **Custom built application.** Building a custom application implies building the common bridge between two systems. The application should understand how the source relates to the destination. In other words, the application needs to know every destination for every source. The more structured the mapping from source to destination, the easier it is to build the application. The advantage of a custom built application for extracting the data is that the application has full control over the process, however full control may imply more work must be done. A custom application approach might be better to run between updates to the information source, depending on how the information source works.

2. **Predefined tools.** Predefined tools can extract data from an information source, however they may not work as well as a custom application. The problem with these tools is that they were designed to fit a general need. In reality the needs may vary and the more the needs vary the more general tools become less useful. The advantage of a predefined tool is that the development time is a lot less than for a custom application. One such tool is Microsoft's SQL Server Integration Services (SSIS), which is used to build integration services between systems. SSIS works by defining a flow of tasks to

execute, where you define the parameters for these tasks and you can package this definition into a complete package which can be deployed and run. The advantage with an integration tool like SSIS is the ability to manage the integration process such as filtering data, sorting data, handle errors and successful runs.

2.6 Programming paradigms

This section presents some information about the programming paradigms used in this thesis project. This section will introduce why these specific programming paradigms are important, when they are used, and how.

2.6.1 Overview

To solve a programming problem in the best way there is a need to understand what paradigms exist in order to choose the correct one for each specific problem [37]. Peter Van Roy states: "A programming paradigm is an approach to programming a computer based on a mathematical theory or a coherent set of principles" [37]. A paradigm can support a set of concepts and it is the combination of these concepts that determines whether it is efficient or not for a specific set of problems. However, the choice of paradigm can be limited due to the programming language that is chosen. A programming language might be chosen for many different reasons, but it should support the programming paradigm that is most suitable to the solve the problem. That is why it is good to use a programming language that supports many different programming paradigms, which makes it better all around [37].

2.6.2 Object oriented programming

There are four basic tenets of object oriented programming: abstraction, encapsulation, inheritance, and polymorphism[38]. The following sections will go deeper into each of these basic tenets and give an explanation as to what they are and how they work.

2.6.2.1 Abstraction

When problems become complex, there is a need to divide them up to make them easier to solve, which is exactly what abstraction helps to do. For instance if you look in a radio, it has a tuner, an antenna, a volume control, etc. In order to use this radio the user does not need to know or understand exactly how the antenna captures radio waves or how the radio then transforms that into electrical signals through a process of filtering and converting it into sound[38]. The user of the radio often wants to be able to use the radio without being a radio expert, hence the radio presents a user interface which abstracts the complexity behind its components by providing the user with only a volume knob and a tuning dial, in order for the user to pick the station. When using abstraction it is important to understand what to show the user and what to hide.

2.6.2.2 Encapsulation

In non-object oriented programming languages such as C, there are data objects called structs. Structs are basically defined sets of data, which in practice means that variables can be bundled together to form an object or an entity. However, C structs do not have built-in encapsulation. Encapsulation or information hiding, is similar to abstraction, but encapsulation is the mechanism by which abstraction is implemented [38]. In other words encapsulation is the mechanism that controls or restricts access to internal data within an object. This allows for the object to control its internals and protect them from being corrupted by other classes using them.

2.6.2.3 Inheritance

Inheritance is a concept where a child (also called a derived) class inherits behaviours of a parent (also called a base) class. Figure 2.6 illustrates how base classes are related to their derived classes.

The reasons why inheritance is desirable or needed in object oriented programming are extensibility and code reuse[38]. As shown in figure 2.6, the relationships between child and parent objects can be explained with "is-a" or "kind-of". However, there are other kinds of relationships such as a composition, where the object is not necessarily "kind-of" but rather "has-a". This means that an object can include or refer to another object in order to be complete. In this way the code is reused and is

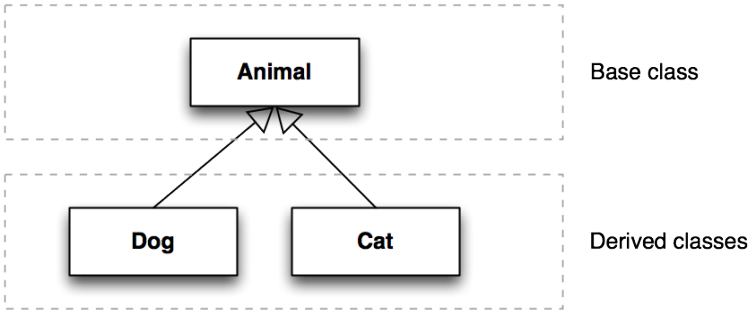

Figure 2.6: Simple overview of inheritance

more extensible. C++ supports multiple inheritance, which implies that a child class has multiple parents. However, it is unusual to use multiple inheritance, since it can cause a lot of trouble with for instance identical function names[38].

2.6.2.4 Polymorphism

Polymorphism allows objects to assume different forms. When calling a method on an object, we pass in parameters and get a response back. With a polymorphic object the important thing is that a specific behaviour exists, disregarding the object's class, as this object can be any arbitrary class [38]. There are two different polymorphism types: parametric polymorphism and overloading. Parametric polymorphism allows the user to choose the parameter type when calling a method [39]. The parameters can be of a certain type, depending on how they are defined in the method definition. This implies that parametric polymorphism is generic in the sense that it only needs to be compiled once for all implicit types. Overloading occurs when an object has two methods of the same name, but they are distinguished by the types of the parameters. [38]

2.6.3 Model View Controller

The Model View Controller (MVC) pattern is today a widely used pattern within object oriented programming [40]. The pattern consists of three parts: the model, the view, and the controller. The idea behind the MVC pattern is to split up presentation logic, business logic, and business models. Below we go deeper into each part of this pattern and the

communication between them.

2.6.3.1 Model

The model domain in the MVC pattern represents the application data and business rules. According to Liu in [41], the model domain should theoretically handle a real-world process. The model itself will consist of a set of classes which support to implement the process. Deacon states in [40] that a model should live as long as the process lives.

2.6.3.2 View

The view domain in the MVC pattern represents a view of a model. In object oriented programming each model will have a set of view classes to show the model itself, which are often graphical. The view is the model's graphical representation and it is therefore important that the view knows at least the structure of the model itself in order to represent it visually [40].

2.6.3.3 Controller

The controller is the domain in the MVC pattern which will handle user input to manipulate the view[40]. Even though it might not be completely true, the controller can be seen as the input whilst the view can be seen as the output. Just as the model does not know about its view, the views do not know about any of the controllers. A controller is a managerial part of the application which causes the model to perform some computation and then show the computation's output using a view[41].

2.6.3.4 Communcation between the domains

The division between the domains in the MVC pattern makes it quite clear what each domain should do, which abstracts the problem of handling all or parts of them at the same time. However with this division, there is now a need to have some kind of communication between the domains. If the models changes, then the view needs to be updated in order to represent the model in the correct way. Similarly if the view has been triggered by the controller to do something, then the model might need to compute something in order to update its internal information. The communication between the domains is event driven,

which implies that every time one domain wants to notify another it will generate an event [40].

Below is an example based on figure 2.7 showing communication flowing in both directions[41] [40]:

1. **View to model.** A user clicks on a button to reload a table with information. The view will generate an event which invokes a method in the controller. The controller accesses the correct model and executes the reload method.

2. **Model to view.** A model has been invoked to reload its information. It will do what ever is necessary for this particular model and generating a new view. When the model is done and has updated its information it will generate an event which lets all interested views know of the change.

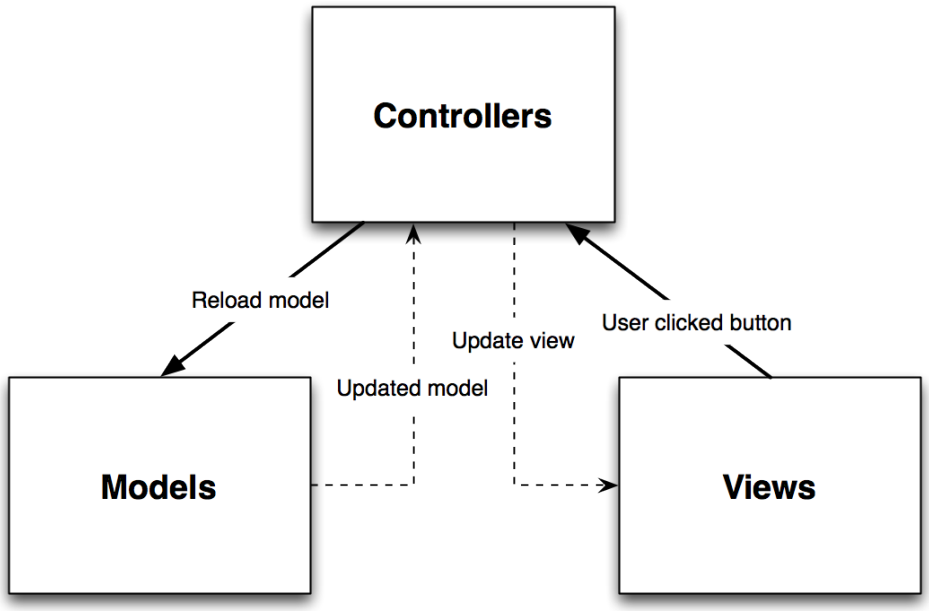

Figure 2.7: Model View Controller communication example

2.6.4 Object relational mapping

Many applications today store their data that needs to be persistent in some sort of persistent store. This means that the application itself needs

a way to communicate with that store in order to store its objects. Section 2.3.3.6 described how Core Data can be utilised in a mobile application to provide persistent (or non-persistent) storage of objects. A persistent store is often a database, where the data can be stored and managed. Object relational mapping (ORM) is an automated way of placing persistent objects in a relational database using information about these objects and their relationships. Basically what ORM does is handle the connection between the application and the store. This implies handling the structed query language (SQL) for the application, thus abstracting away the problem of writing proprietary SQL.

ORM consists of four components [42]:

1. An Application Programming Interface (API) for basic creation, reading, updating, and deleting of objects that are persistently stored.

2. An API or a language that can create queries to operate on the classes and properties of objects.

3. A way of defining each object and mapping them together.

4. A way for the ORM to perform dirty checking, lazy association (more about these in 2.6.4.2), or other such optimisations.

2.6.4.1 ORM implementations

An ORM can be implemented in different ways depending on how hard the coupling should be between the object and the persistent store. Below are four different paradigms used in ORM implementations [42]:

1. **Pure relational.** The application is built around fully relational models and SQL-based relational models. A pure relation ORM implementation is often slower, less maintainable, and not as portable, compared to writing queries in SQL optimised for the specific situation. However, a pure relational ORM can be really useful for a smaller system, because all mapping and handling of objects is handled by the ORM.

2. **Light object mapping.** With light object mapping the objects are manually mapped to a relational table. This implies that there is a need to manually write code to map each object to its corresponding database entity.

3. **Medium object mapping.** In the medium object mapping paradigm the SQL code needed to handle the objects is built at compile time or at run time. This means that there is a SQL code generator running that uses the defined mappings, however this code generator is only invoked when needed. Medium object mapping is a basic way of implementing an ORM and many ORM implementations support this paradigm. This kind of mapping is good when portability is needed, especially between different persistent stores.

4. **Full object mapping.** Full object mapping means that the persistence layer supports composition, inheritance, and polymorphism (see section 2.6.2 for a better explanation of these terms). It also means that the implementation has transparent support for optimised strategies.

2.6.4.2 ORM optimisation strategies

This section describes two different ORM optimisation strategies that are commonly used in ORM implementations. These optimisation strategies are important to understand in order to have a good picture of how ORM actually works. Optimisation is often important since one of the biggest problems with ORMs is that associations need to be handle efficiently. These two optimisation strategies are explained [42]:

1. **Dirty checking.** Dirty checking occurs when the ORM implementation checks for updates in an object in order to know when to update the persistent store. There are two ways of doing this, one way is *interception* and the other way is *inspection*. The interception method intercepts the assignment of an object's properties, hence interception knows when an update happens. The inspecting method will check the object's value at the end of a transaction and compare it to a local snapshot of the object to see if the object needs updating.

2. **Lazy association.** An object might have an associated object. One problem that ORM implementations have to solve is how to load the associated objects. Lazy association is a strategy which fetches the associated objects only when they are first accessed. This strategy has its advantages and disadvantages, while optimising the fetching of the object itself it takes longer time for fetching associated objects.

2.6.5 Event-driven computing from a Node.js perspective

Node.js which was described in section 2.4.2 is based on JavaScript. In JavaScript events have always been in the core of the language, rather than a side effect [24]. JavaScript has always dealt with user interaction, such as "onclick" or "onmouseover" events, which are all triggered by the user. In other words JavaScript reacts to an act. Node.js utilises this feature of the JavaScript language, to make the execution non-blocking. Non-blocking implies that the code will never wait for the method to return, but rather the specific operation runs independently and either emits an event when it is done or calls a callback. Since the non-blocking code will run separately, we need a way for the flow of the execution to find its way back to the next instruction in the flow. The use of callbacks is very common in Node.js, because of the support for non-blocking calls in the system libraries.

Essentially what happens when Node.js is executing its code is to first run through the code and set up the events that will listen for changes. When this is done, Node.js does not exit, unlike a non-event-driven program. A non-event-driven program will execute the code until it is finished, while Node.js waits and listens for an event to occur and continue the execution from the callback or event listener.

Event-driven programming is beneficial when there are lots of slow operations (such as I/O) to be handled. Node.js will handle these operations in the background and asynchronously call the callback when the operation is completed. A good example is to think of a restaurant, where you tell the waiter what you want and the waiter tells the chef what to start cooking. In the meantime while the food is being prepared, the waiter goes to the next customer and solicits their order. When the chef is done cooking an event will be generated, telling the waiter that it is time to serve the customer.

An important concept in all this is that the code that is written for Node.js is run in a single thread. Taking the example into account, the restaurant only has one waiter. This implies that if the Node.js code (or the waiter) gets stuck at one customer, then the other customers will have to wait. [24]

Chapter 3

Method

This chapter presents the methodology used as well as the method and execution of the thesis project. Section 3.1 describes the methodology and the measuring framework developed for this thesis project. Section 3.2 presents the design of the system and the responsibilities each part has of the overall design. Section 3.3 describes how the system was actually implemented, considering the goals and the design. Section 3.4 shows the test set up and describes how the performance of the system was measured.

3.1 Methodology

This section describes the framework for how the performance measurements and evaluation were done. First the methodology used is described in section 3.1.1. The framework is not only used to describe the metrics for performance, but this framework is also used to define the goals of this project - as described in section 3.1.2.

3.1.1 Methodology

The methodology used in this thesis project is a combination of two different methods: quantitative and qualitative. The question that is asked in this thesis requires answers and measurements that combine both quantitative and qualitative analysis.

The qualitative analyses will be done based on the background information gathered, whereas the quantitative analyses will be done by setting up a test system and measuring the performance. Using the

framework as support for the analyses, conclusions will be drawn as to whether the defined goals were achieved or not (the actual analyses are in Chapter 5).

Having qualitative and quantitative analysis together, covers both sides of the assessment of the proposed system. Together with the conclusions drawn, we will clarify what has been done and what remains to be done (see section 6.2). The suggested future work (section 6.2) will give future researchers a better ability to build upon this thesis work.

3.1.2 Measurement framework

In order to build a measurement framework it is important to first understand our goals, because the measurement framework's only purpose is to measure whether we have achieved our goals. Below a short description is given of each goal and how it should be measured:

1. **Core functionality (goal 1.a)**. The core functionality concerns the correct implementation of features as requested by Netlight. The core functionality of the system was defined together with the product owner in order to make sure that the end system provides the *required functionality*. This goal will be measured in a qualitative way, i.e., as to whether the required functionality exists or not.

2. **Build a system for the future (goal 1.b)**. Building a system for the future is hard to measure, since predicting the future is hard, hence, the measurement will be simplified. Given that the system is following the current trends specified in 2.2, the system will be measured as if it was built for the future (i.e., by applying a load that might be expected in the future).

3. **Mobile application is easy to use (goal 2.a)**. This goal will be measured by performing a usability test with a group of five people. According to Jakob Nielsen, five users is sufficient and more users will not give a much better result [43]. Based on the results of this usability test, an analysis will be done to conclude how easy the mobile application is to use.

4. **The mobile application should maintain synchronisation with the back end (goal 2.c)**. This goal requires less analysis, than the goals above. To achieve this goal synchronisation should be maintained

between the local cache in the mobile application and the back end. The criteria for this synchronisation mechanism is that it synchronises periodically *faster than the back end synchronises with the ERP*.

5. **Scalability in the back end (goal 3.a)**. A qualitative analysis (based on the definition of scalable systems in section 2.2.2) of the back end should be done in order to ensure that all components are scalable as needed. This goal will be considered to be achieved if all the components in the back end can scale.

6. **Sufficient speed in the back end (goal 3.b)**. Having a fast enough back end to serve the mobile application will help users not to lose interest, as otherwise they might not use the application. In order to achieve and analyse this qualitative goal, there needs to be a limit to which we can compare the performance. The limit has been set at 3 seconds, because of the requirements given by Netlight. However, according to Jakob Nielsen, everything that takes between 2 and 10 seconds is sufficient to avoid the need to show a loading screen [44]. It is also important that the system can deliver this performance under a load of 500 concurrent users, as this is also a requirement set by Netlight (appendix A).

7. **Get the information from the ERP (goal 4.a)**. In order to measure this qualitative goal it is important to understand what qualities make a synchronisation module better. Better in this sense concerns two factors: time to implement and performance. Performance concerns how long it takes to synchronise the data and the downtime of the system, as both are important factors in order to understand how well the synchronisation performs.

8. **Integration with ERP (goal 4.b)**. If the thesis includes options for different forms of integration and describes some of the issues of integrating the back end with an ERP, this goal will be considered to be fulfilled.

3.2 Design

This section introduces the design of the system as a whole and then describes each of its parts. The design will conceptually describe the

entire system, in order to give a better understanding of how all the parts of the system connect to each another. This section also introduces and describes each part's responsibility and functionality.

3.2.1 System architecture

Figure 3.1 shows the conceptual system architecture, which will be the focus of this thesis project. All of the parts that are defined in the conceptual system architecture, play a specific role. The leftmost part of the architecture is the mobile application, which is described in section 3.2.2. The back end itself is in the middle of the architecture. The back end application is described in section 3.2.3. The database drivers and what role they play are discussed in section 3.2.4. The database cluster's role in the architecture and its performance are described in section 3.2.5. The rightmost module in the back end is the synchronisation mechanism, which is described in section 3.2.6. The ERP section of the system architecture describes how the synchronisation module in the back end is connected to the ERP's back end repository, which is used for storing information.

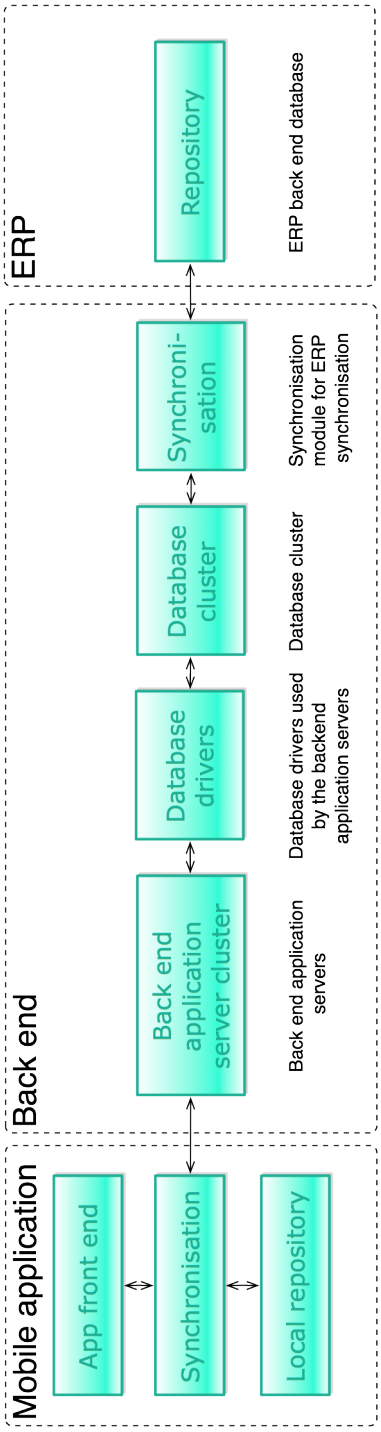

Figure 3.1: Conceptual system architecture. The figure shows how the parts connect to each other.

3.2.2 Mobile application

The mobile application will act as a client to the system. The role of the mobile application, is to help the users navigate within the information in order to find the information that they are looking for. The mobile application must meet both functional goals (i.e., providing specific required functions) and certain performance goals. How the mobile application performs with regard to fetching the relevant data is also important, because the mobile application must fetch the data and present this data to the user. There are three major parts of the mobile application. The first part is the module called "app front end". This is an essential part of the application as it provides the user interface (UI). The app front end needs to fulfil requirements regarding the UI and show the information in such a way that the user can easily navigate and find the information that they need. The second part of the mobile application is the synchronisation module. The synchronisation module is responsible for determining how and when the information should be stored or fetched and from where the application should get the information. In other words the synchronisation module is responsible for handling pre-fetching of information that *might* be used. When the user then wants to access the information it is up to the synchronisation module to determine from which source to fetch the information (i.e., the local cache - implemented by the local repository - or the remote back end). An important thing to note for the synchronisation module is the problem of handling revisions (more about synchronisation fundamentals in section 2.3.3.1). Lastly the local repository stores information persistently. The local repository is fed by the synchronisation module. The local repository needs to store the information, so that when the synchronisation module asks for it the local repository can deliver. However, the local repository does not have an obligation to deliver information, but rather it has to be able to tell the synchronisation module that it does not have the information, which was searched for and therefore that the synchronisation module should get this information from the back end. As long as the modules follow this logic the information should be correctly synchronised between the mobile application and the back end, under the assumption that correct revision logic is implemented (to ensure that changes to the data in the

back end propagate to the local repository as described in 3.3.3).

3.2.3 Back end application

The role of the back end application is to serve the mobile application or any other devices that might be used in the future. It does this by providing a REST API, which is a common standard and therefore allows easy integration for different clients based upon different platforms (more about the REST protocol in section 2.4.5). This API is well defined and known to the mobile application and any other clients. A client uses this API to fetch specific information by crafting suitable URIs.

Since the role of the back end application is to extract information from the underlying storage subsystem and delivering it to any (authenticated and authorised) client, it is essential that the back end application can do this with good performance in order to achieve the goals stated earlier. The back end application needs to be able to scale up to handle incoming requests during peak times. Also since the main client in this system will be a mobile application, there is a need to make the system responsive enough to avoid users losing interest. This implies that the back end application needs to keep the response below a bounded amount of time (as stated in the requirements this is 3 seconds). Therefore the back end application, as a provider of information for incoming requests needs to meet the desired response times.

3.2.4 Database drivers

The database drivers are connected to the back end application and are responsible for helping the back end application get the actual data from the underlying storage, such as a database cluster. Depending on the underlying database cluster the database drivers will differ. However, all database drivers provide an interface for the back end application to talk to the database cluster in a simple way. In this thesis, the focus will be on database drivers with built in ORM functions (see section 2.6.4.2 for more information on how ORMs work). This implies that the database drivers will be responsible for most of the querying to fetch and store information. However, not all of the responsibility is given to the ORM, since there will be problems with normalised tables, where custom queries need to be written by hand.

3.2.5 Database cluster

The database cluster acts as the repository storage for the system's back end. In order to be able to support a larger number of users the database back end cluster will have to be able to be replicated or sharded across multiple servers, thus we propose using a database cluster (more about this in section 2.4.3.3). Having a database cluster might actually decrease performance depending on the structure of the cluster. The ability of the cluster to scale is essential.

The role of the database cluster is not only to store the information that the system will use, but also to deliver it in the fashion that is required. The performance of the database is therefore crucial for this, as the database must be able to deliver information to the mobile application within the specified bounded time. Understanding the database cluster's performance is important as well, in order to be able to draw conclusions and see that the database does not become a bottleneck in the delivery of data to the mobile clients.

3.2.6 Synchronisation module

In this thesis project the system is supposed to extract a subset of information from the ERP system to the back end, which means that the synchronisation module will only do copying (reading) from the ERP system. This information is to be accessible via the mobile application. The mobile application communicates with the back end application, which in turn gets the information from the back end database cluster. In order for the correct subset of information from the ERP repository to reach the back end database cluster and later be provided to the mobile application, there is a need for a synchronisation module in the back end. The role of the synchronisation module in the back end is to synchronise the information from the ERP to the back end database cluster. Depending on the delay requirements for updating the information this synchronisation module needs to have varying performance. It is important that the information updates from this synchronisation module meets these requirements, otherwise there is no point in extracting the information. The role of the synchronisation module may differ, depending on the system's requirements. However, in this thesis the focus will not be in how synchronisation delays affect the value of the system, but rather to explain how the synchronisation is

done.

3.3 Implementation

This section will describe how the system was implemented. The goals stated earlier affected the design decisions and the functionality and performance of the implementation has to meet Netlight's requirements. The implementation itself consists of different parts. Each part is further described in one of the following sections. The back end application implementation is described in section 3.3.1. How the databases were set up is described in section 3.3.2. The mobile application synchronisation mechanism is described in the section 3.3.3. The database drivers layer is described in section 3.3.4. The synchronisation between the back end database of the ERP and the back end application server is described in section 3.3.5.

3.3.1 Back end application

Figure 3.2 shows how the back end is designed and how this design guided the implementation. The figure depicts how the MVC concept applies to the implementation of the back end application (more about MVC in section 2.6.3). The flow of the back end begins with the mobile application or any other REST capable client requesting a resource using a specific URI. The request is delivered to a HTTP server. The back end application running in this HTTP server passes the request on to the (request) router, which is responsible for selecting the correct controller to handle the request. The selection is based on the request URI, in this case the decision is based on the type of resource that is requested. In this implementation the view is not very visual, but the view is still the part of the application that triggers the application to respond based on an interaction with a user or machine.

In practice a request to *http://host:port/employees/1*, will go to the controller which handles "employees" requests. The application is built in such a way that each resource type has its own controller. The controller is registered in the (request) router. By implementing the application in this way, an arbitrary number of different types of resources can easily be added and the application will simply scale up. The controller is responsible for handling the request and making sure that the requested

information reaches the user. The controller will utilise the ORM to fetch the data based on the type of resource that was requested.

The ORM consists of models and a database connection (more about ORM in section 2.6.4.2). The controller calls the model, which contains the query methods needed to fetch a specific instance of a resource. However, this is not always possible, since the ORM cannot provide all the custom queries that the application might want to run. Therefore the ORM provides methods for executing SQL queries. The ORM will maintain a connection to the underlying back end database to execute the queries in order to fetch data and return it to the controller.

Finally the controller will format the data in a structured and consistent way in the responder module and send it back to the client that initiated the request.

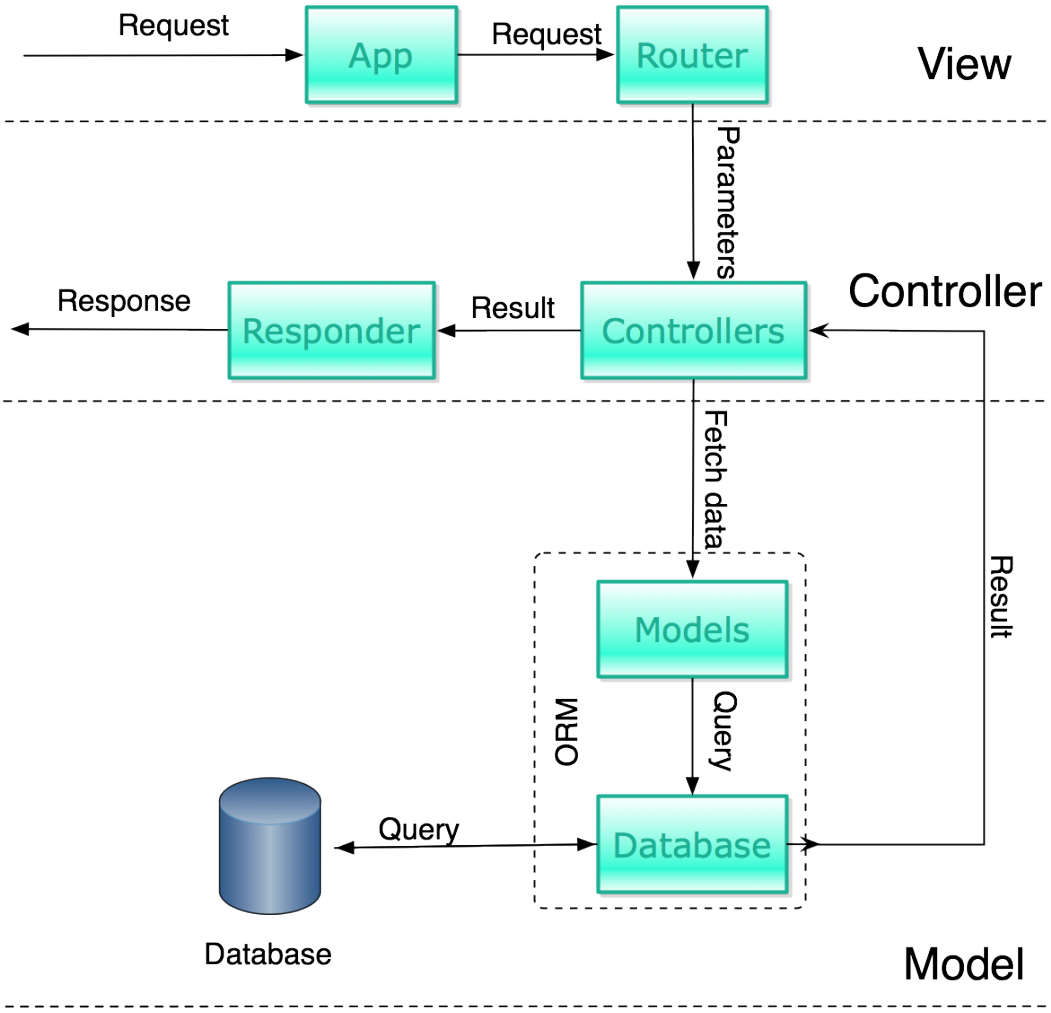

Figure 3.2: Back end application structure, from an MVC point of view.

3.3.2 Database setup

The back end database holds the extracted subset of data that was earlier extracted from the ERP system's database. One important aim is that the database should scale up easily, because in this way the ERP database does not have to scale up in order to serve this subset of data. For this reason the back end database should be chosen so that it can scale easily. In this initial implementation only a single MySQL server is used. MySQL already allows for clustering databases, which implies that if the system needs to scale in the future, additional database nodes can be added.

3.3.3 Mobile application synchronisation

Due to the limited time available for this project the mobile application synchronisation part of the system has not been implemented, but a solution will be proposed below. Synchronising data within the mobile application is not a simple task and the details of this will be left to future work. In this section we present one way of doing so which would fit the needs of this project.

Figure 3.3 shows the overall flow of the synchronisation. When the application starts, an initial request will be sent to the server. This request will contain an indication of what data has already been synchronised, preferably using ids. The response will contain all ids that have been added after the ids specified in the request. The application will then sort the ids in this response, placing the ids for information that will be needed sooner first [1]. This way the application can focus on getting the latest information that is mostly like to be needed. The next step is for the application to decide if it is crucial that a piece of data is updated or not. Based on the above mentioned decision, the data will either be returned to the caller based on what is in the local store or wait until the data has been synchronised into the local store. It is important to note that no data has yet been returned to the user. Now the background synchronisation starts, based on the sorted queue of ids. The background synchronisation should request only one or a couple of resource(s) at a time, because of the following reasons:

1. **Do not lose connectivity**. We do not want to lose network connectivity when synchronising bigger chunks of data.

2. **Scales is size**. If the size of the data to be synchronised is big, the risk of encountering a transmission failure is higher.

3. **Measure progress**. It is easier to manage and understand how much work has been done and how much is left. A progress bar can also be shown to the user to show how much work that is left before they can start using the application or until the synchronisation is finished.

Once a resource has been synchronised to the local store, it should be taken off the queue.

[1] The application needs to know which information that needs to be requested first.

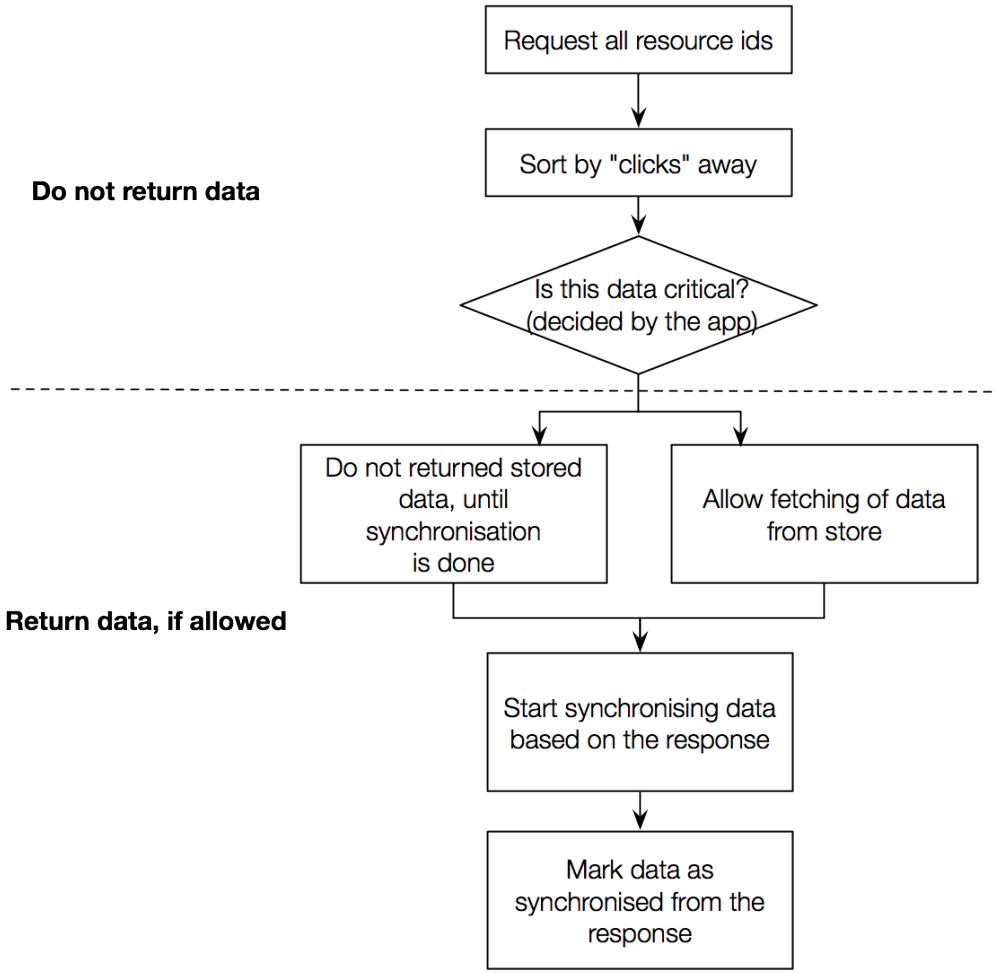

Figure 3.3: Mobile app synchronisation flow scheme.

3.3.4 Database drivers

The database driver is a module from NPM (see section 2.4.2 for more information) and is called Sequelize.js [45]. The database driver is not only a network connection to the database server, but it is also a complete ORM. Sequelize.js is open source and is still under development, and therefore some work had to be done in order to modify it to better fit the needs of this thesis project better.

3.3.5 Synchronisation mechanism

There is a need to copy the information from the ERP with the system's back end database (in this case the MySQL server). The copying does not have any real time requirements, as we assume that nightly copying is sufficient.

There are different ways of doing this copying, but Netlight requested the use of SSIS (described in section 2.5.2), since they already have other systems running SSIS packages and thus the integration would be easy. The SSIS package does two things:

1. Truncates the database tables. This truncation is done in order to clear all data from the tables, but to keep their structure.

2. Copies data from the ERP. The ERP has custom built database views of the data that are needed in the back end database. Every time the SSIS package runs, the data which has already been formatted correctly is copied into the back end database.

The whole process takes about 10 seconds, which is an acceptable downtime for the back end system. The predicted usage of the mobile application during night time, when the SSIS package is run is almost non-existent. However, a problem that might occur is that as the data grows in the ERP, then it is going to take longer to refresh the back end database. If reduced downtime becomes essential and more data is to be copied, then a solution could be to do partial updates, thus minimising the risk of information not being available to the end users.

3.4 Test setup

This section focuses on how the test system was set up, in order to measure the performance of the system. The information generated will be used in the analysis and from the analysis conclusions will be drawn. Section 3.4.1 describes what tools were used in order to run the tests and how each tool works. Section 3.4.2 describes the system set up for the testing and how the tools are connected to each other in order to build a complete testing system.

3.4.1 Measurement tools

In order to be able to measure the different aspects of the system to ensure that it meets the requirements, there is a need for a number of different tools. These tools will be used primarily for quantitative measurements, but can also be applied in the qualitative analysis. Below is a list of different measurements and the tools that will be used for measuring the different parameters (these parameters are needed for later analysis):

top Top is a standard unix application to measure CPU and memory utilisation. Top will be used to make sure that the hardware of the system is not the bottleneck. It is important to have a good understanding of how the hardware behaves, especially when making a qualitative analysis of the system's behaviour. Top will be used to collect data about how the system performs in practice.

free Free is a linux application which measures how much memory the system is currently using. For our performance tests, free enables us to measure the amount of memory that is needed when running the application. However, one important thing to keep in mind is that free will show the system memory usage and is not application specific. When using free in system tests, recording the starting values is important in order to understand how much memory the application is actually using.

Apache benchmark Apache benchmark is a tool for benchmarking a web server [46]. You can specify how many connections and how many requests the stress testing should include. The apache benchmark will output the minimum, mean, median, and maximum time it took process a request. Using these results should give us an idea of how the system can handle a given amount of load. It is important to measure the response times as a function of load so that we can ensure that the system will keep the user's interest. If the response time is too long, then the user will quit using the application.

3.4.2 Set up

The test system was set up as depicted in figure 3.4. There are two servers used in this test setup to be able to measure the bandwidth coming in and

out of the back end server without adding the load of the testing software to this server. The tools described in section 3.4.1, were mostly run in the back end server, where the measurements need to be run. In the back end server a measurement script was created, which uses top and free. The script is in Appendix B.

The test server will be running Apache benchmark in order to stress the back end server. Different tests will be done to measure performance when fetching different resources with different parameters, to see how the back end handles different loads. The results will be gathered in comma separated files and will be analysed by looking at graphs generated from this data.

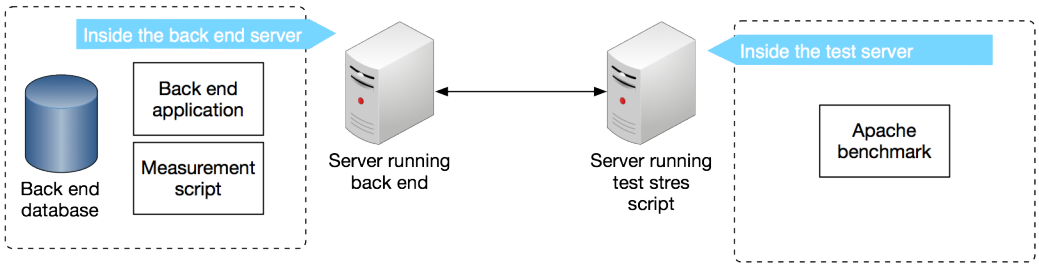

Figure 3.4: A model of the test setup, with internal components

Chapter 4

Results

The results presented in this chapter are results from the quantitative tests. The results are described briefly in this chapter and are then further analysed Chapter 5. The measurements shown here were done on a virtual server with the specifications listed in listing 4.1. The measurements are done using different parameters to see how the system handles different loads. The below list explains these parameters:

1. **No analysis**. No analysis indicates that the server has turned off its analysis, such as statistics gathering, to optimise performance.
2. **Socket fix**. Socket fix indicates that the operating system of the test server has been configured to recycle and reuse sockets. This implies that the system will allow the use of sockets that are in the TIME_WAIT state. A socket enters the TIME_WAIT state after closing its connection [47]. The system sets the state of the socket to TIME_WAIT in case delayed packets might reach the system later. After a while the socket is removed.
3. **Cluster**. Cluster is a Node.js module which enables the back end to initiate multiple processes when using multiple processor cores.

Table 4.1: Test server specifications

Operating system	Ubuntu server 12.04
CPU	4 cores @ 2.66 Ghz
Memory (RAM)	2048 MB

The following sections focus on what resources were being utilised. All of the following results are based on *500* concurrent connections [1] and in total *50,000* requests.

4.1 Employee with id 404

In this section the results for fetching information about employee with the ID number 404 will be shown. The response size for this resource was 68 bytes.

4.1.1 Default settings

These are the results from running the benchmark with default settings, implying that analysis module is enabled, socket fix is not applied and no clustering is enabled. Table 4.2 shows the response times when running the benchmarks with the settings mentioned above. As we can see both in table 4.2 and in figure 4.1, some of the responses take more than 3 seconds. However the fastest response is 12 milliseconds. Figure 4.2 shows the CPU and memory consumption during this benchmark.

Table 4.2: Results from apache bench, with default parameters

	Min (ms)	Mean (ms)	Median (ms)	Max (ms)
Connect	0	167	30	15071
Processing	12	599	356	27987
Waiting	11	578	356	27986
Total	12	766	415	27993

[1] The number of concurrent connections are increased over time, which might give the back end a slight advantage in the results.

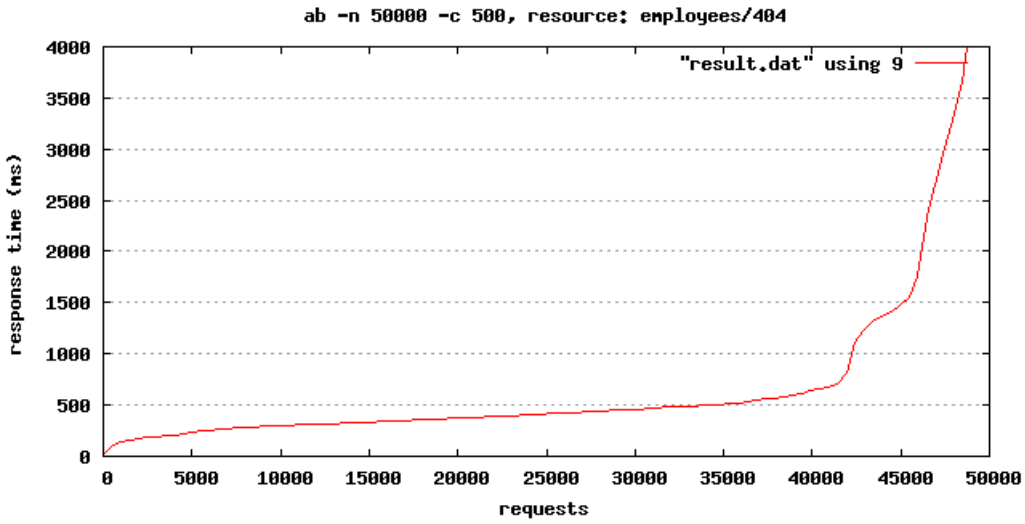

Figure 4.1: Apache benchmarking results

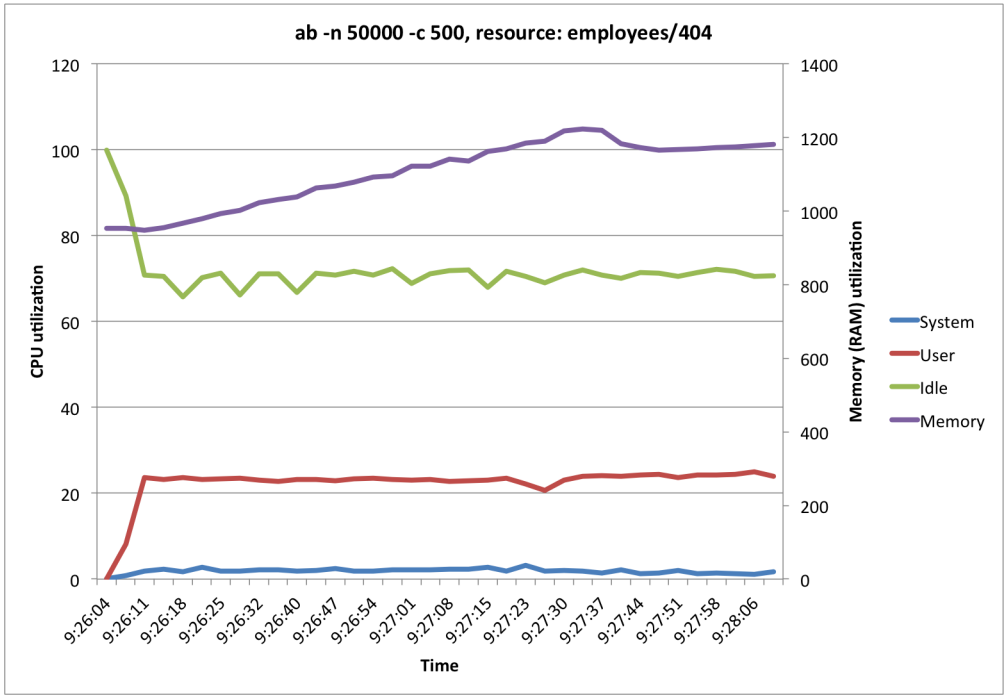

Figure 4.2: CPU and memory consumption during test

4.1.2 No analysis

These are the results from running the benchmark without the analysis module. As we can see in table 4.3, the slowest response was over 31 seconds and the fastest was in 3 milliseconds. Figure 4.3 shows all the response times for the benchmark. Figure 4.4 shows the CPU and memory consumption during the benchmark.

Table 4.3: Results from apache bench, no analysis

	Min (ms)	Mean (ms)	Median (ms)	Max (ms)
Connect	0	440	13	31067
Processing	1	129	42	17588
Waiting	1	123	41	13402
Total	3	569	70	31210

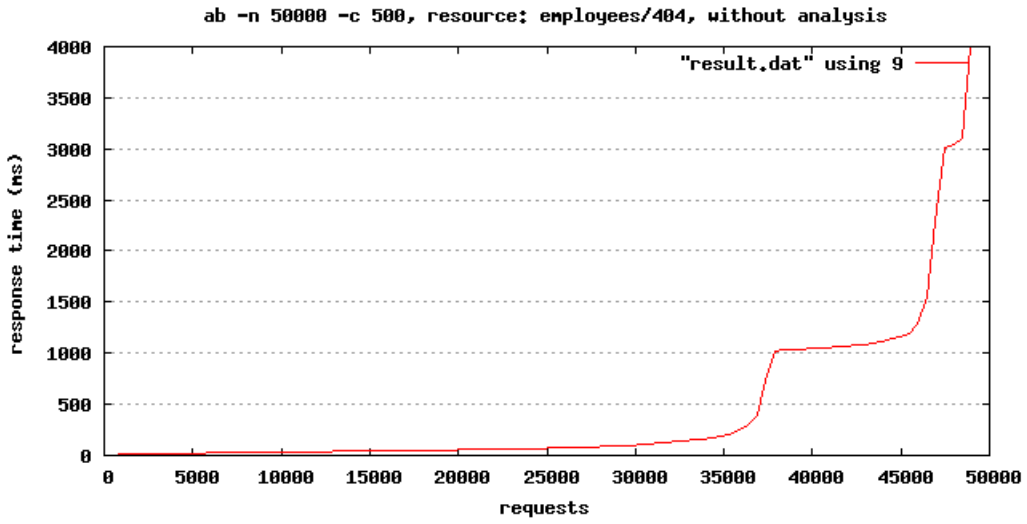

Figure 4.3: Apache benchmarking results

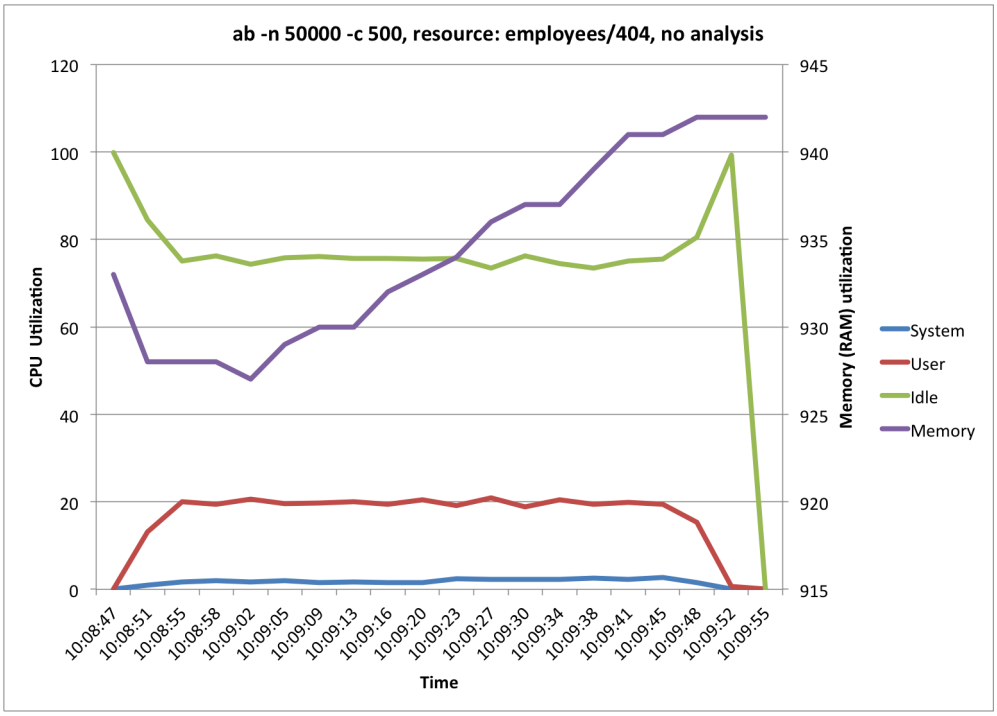

Figure 4.4: CPU and memory consumption during test

4.1.3 Socket fix

These are the results from running the benchmark with only the socket fix applied. Table 4.4 shows the response times when running the benchmarks with the settings mentioned above. As we can see both in table 4.4 and in figure 4.5, the slowest response takes more than 61 seconds. However the fastest response is 13 milliseconds. Figure 4.6 shows the CPU and memory consumption during this benchmark.

Table 4.4: Results from apache bench, with socket fix enabled

	Min (ms)	Mean (ms)	Median (ms)	Max (ms)
Connect	0	412	15	31195
Processing	2	394	241	61171
Waiting	2	380	240	53644
Total	13	806	297	61188

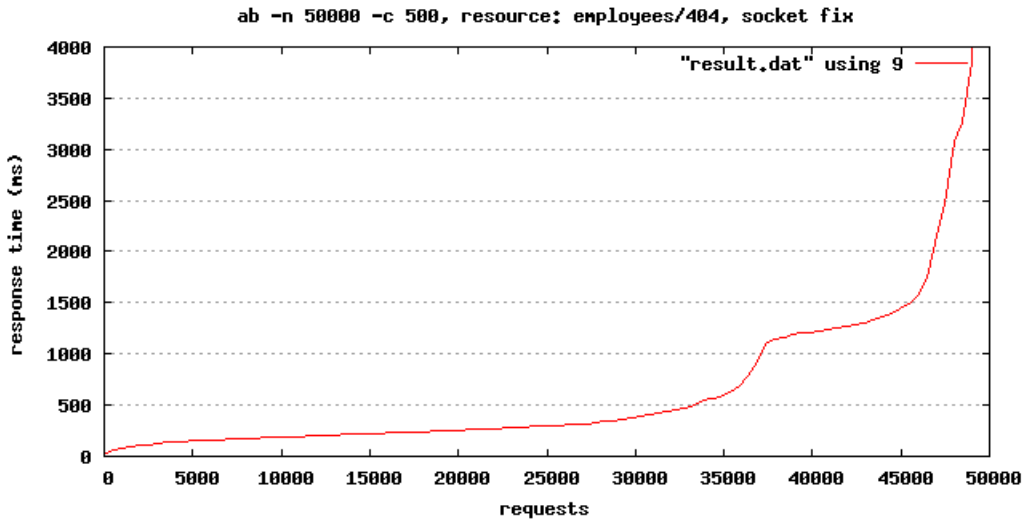

Figure 4.5: Apache benchmarking results

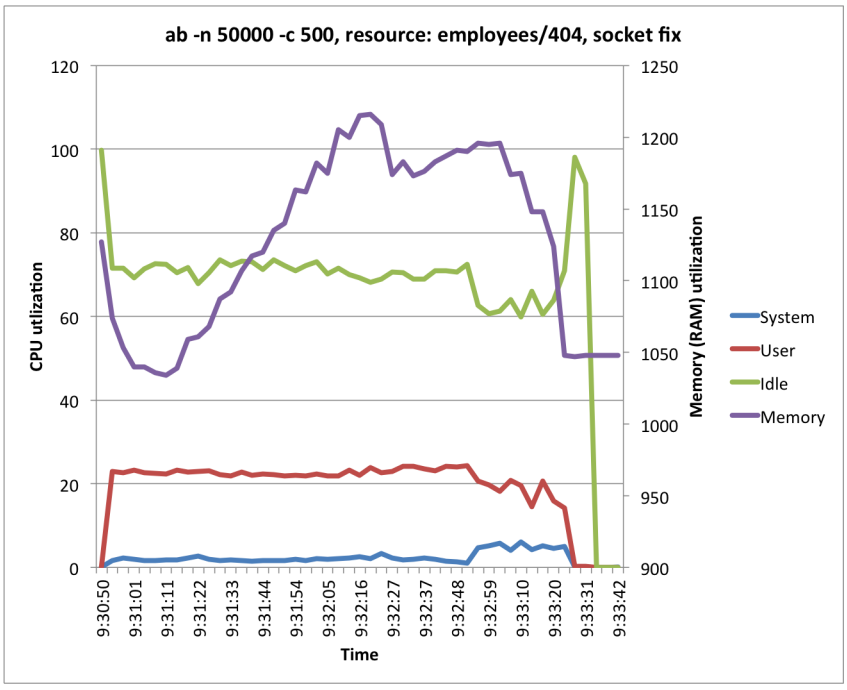

Figure 4.6: CPU and memory consumption during test

4.1.4 Socket fix and no analysis

These are the results from running the benchmark with socket fix enabled and without the analysis and clustering module. Table 4.5 lists some of the statistically interesting response times from this benchmark. Figure 4.7 shows all of the response times in a graph. Figure 4.8 shows a graph over the CPU and memory consumption during this benchmark.

Table 4.5: Results from apache bench, with socket fix enabled and analysis disabled

	Min (ms)	Mean (ms)	Median (ms)	Max (ms)
Connect	0	308	10	32292
Processing	1	106	35	12617
Waiting	1	103	35	12614
Total	2	414	54	32314

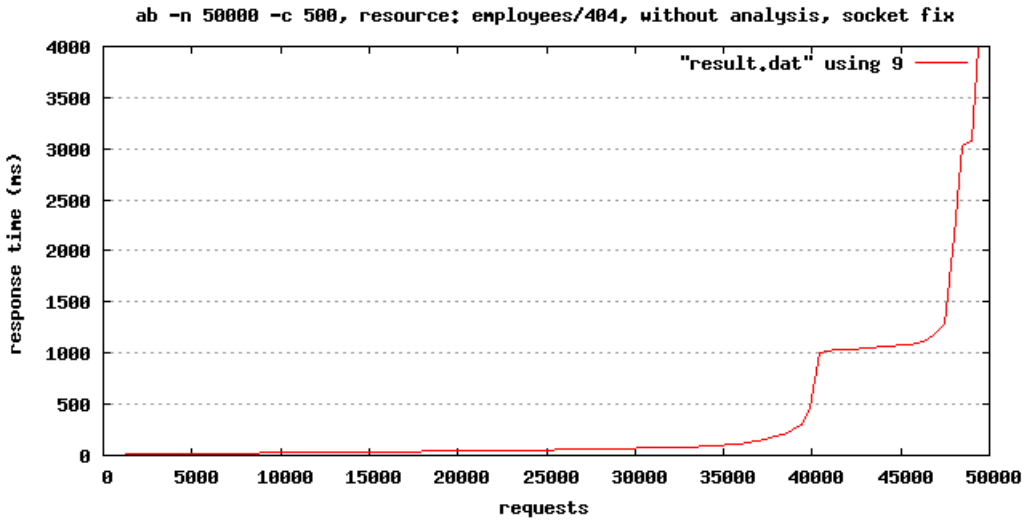

Figure 4.7: Apache benchmarking results

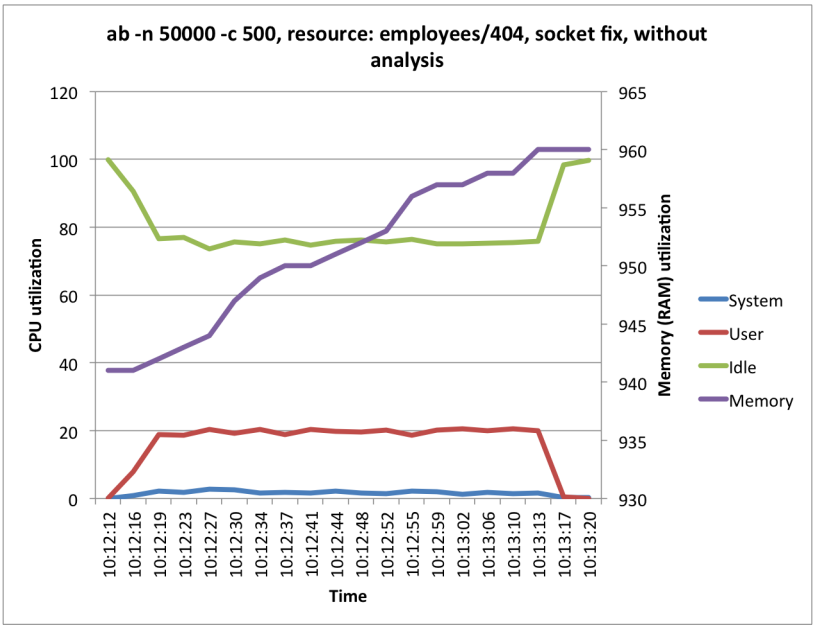

Figure 4.8: CPU and memory consumption during test

4.1.5 Socket fix and clustering

These are the results from running the benchmark with the socket fix and clustering module enabled. Table 4.6 gives us the statistical results from running the benchmark with the above mentioned settings. Figure 4.9 presents all the response times in a graph. Figure 4.10 shows the CPU and memory consumption during the benchmark.

Table 4.6: Results from apache bench, with socket fix and clustering enabled

	Min (ms)	Mean (ms)	Median (ms)	Max (ms)
Connect	0	424	12	31092
Processing	2	395	58	38843
Waiting	1	342	57	30635
Total	2	820	87	43163

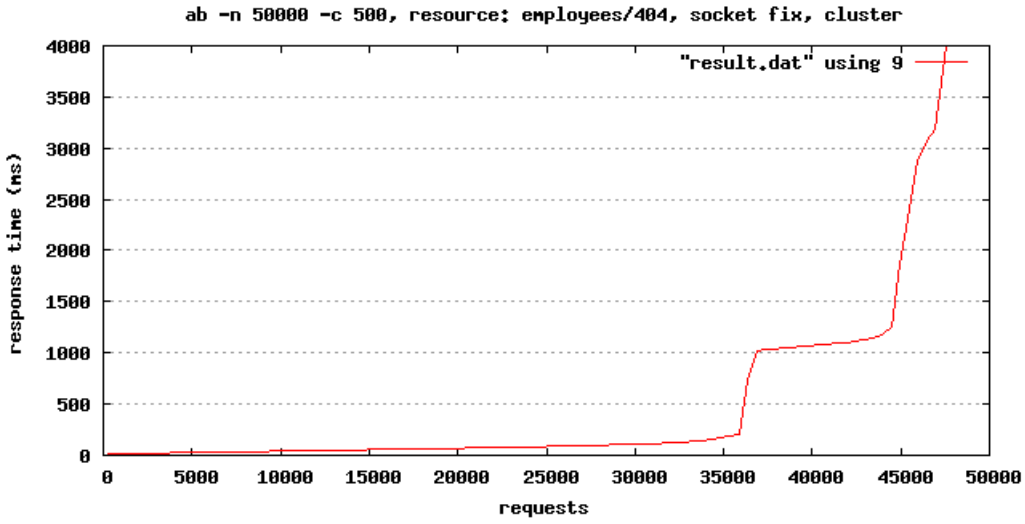

Figure 4.9: Apache benchmarking results

Figure 4.10: CPU and memory consumption during test

4.1.6 Socket fix, no analysis and clustering

These are the results from running the benchmark with the socket fix applied, without any analysis and with clustering module enabled. In table 4.7, we can see the minimum, mean, median and max values of the response times in milliseconds. Figure 4.11 depicts all of the response times in a graph. The CPU and memory consumption for this benchmark are shown as a graph over time in figure 4.12.

Table 4.7: Results from apache bench, with socket fix enabled, analysis disabled and clustering

	Min (ms)	Mean (ms)	Median (ms)	Max (ms)
Connect	0	327	11	31118
Processing	1	71	8	19123
Waiting	0	67	8	19122
Total	2	398	22	31119

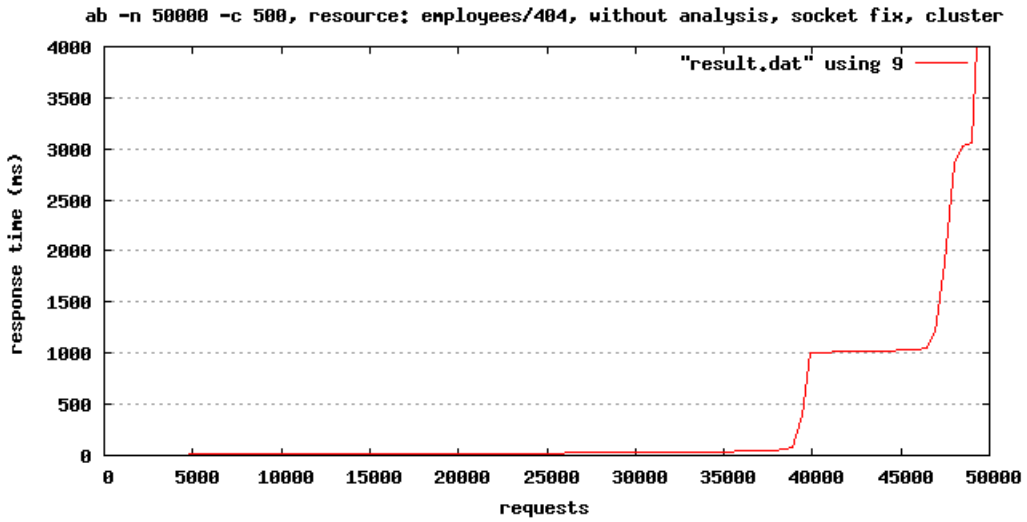

Figure 4.11: Apache benchmarking results

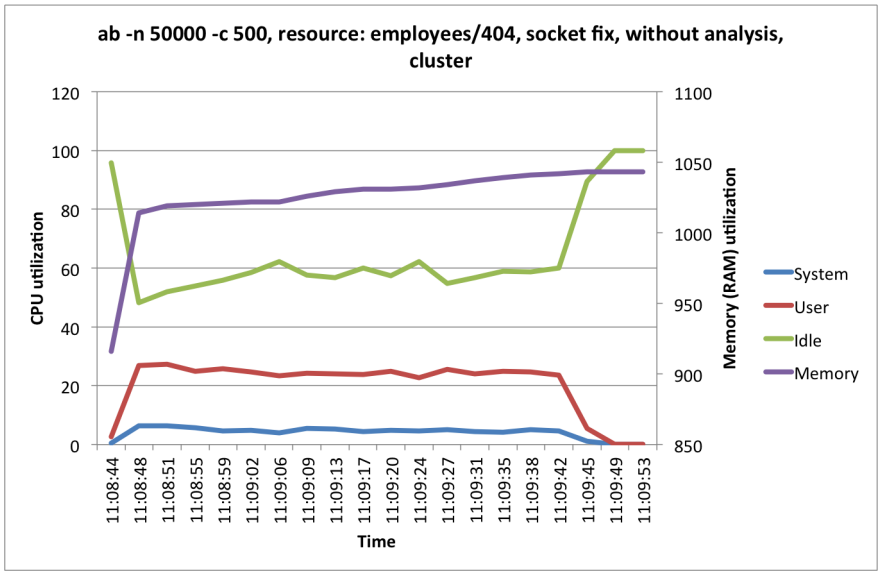

Figure 4.12: CPU and memory consumption during test

4.1.7 Local benchmark, socket fix, no analysis and clustering

These are the results from running the a benchmark locally (not to a machine on the network) with the socket fix applied, without any analysis and with clustering module enabled. Table 4.8 gives us the statistical results from running the benchmark with the above mentioned settings. Figure 4.13 presents all the response times in a graph. Figure 4.14 shows the CPU and memory consumption during the benchmark.

Table 4.8: Results from apache bench, running the benchmark locally with socket fix enabled, analysis disabled and clustering

	Min (ms)	Mean (ms)	Median (ms)	Max (ms)
Connect	0	1	0	47
Processing	1	155	142	519
Waiting	1	155	141	519
Total	1	156	143	519

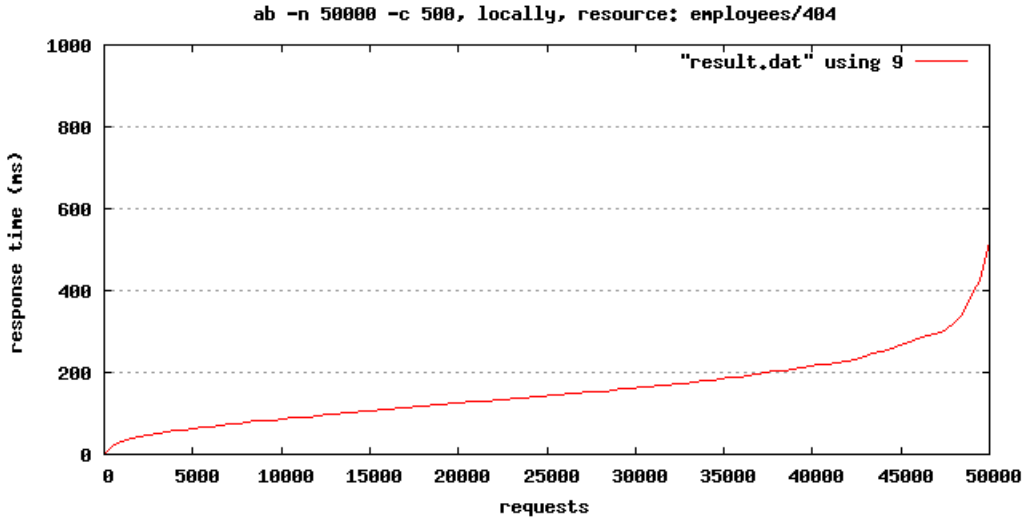

Figure 4.13: Apache benchmarking results

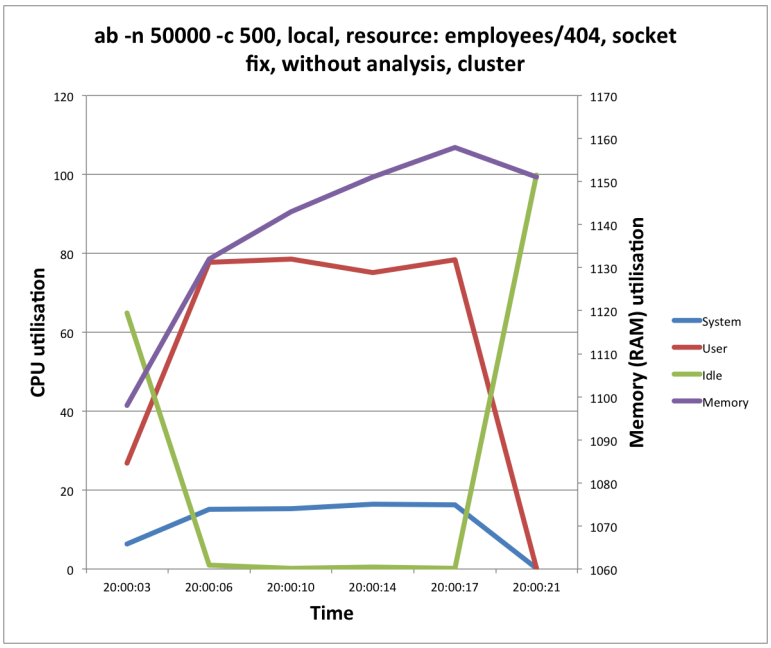

Figure 4.14: CPU and memory consumption during test

4.2 Employees list - page 0

In this section the results for fetching the list of employees for page 0 will be shown. Page 0 implies 10 employee entries in this case. This resource was 3110 bytes.

4.2.1 Default parameters

These are the results from running the benchmark with default settings, implying that analysis module is enabled, socket fix is not applied and no clustering is enabled. As we can see in table 4.9, the slowest response was 10 seconds and the fastest was in 252 milliseconds. Figure 4.15 shows all the response times for the benchmark. Figure 4.16 shows the CPU and memory consumption during the benchmark.

Table 4.9: Results from apache bench, default parameters

	Min (ms)	Mean (ms)	Median (ms)	Max (ms)
Connect	0	14	1	7010
Processing	228	2908	2923	3544
Waiting	228	2908	2922	3544
Total	252	2923	2926	9668

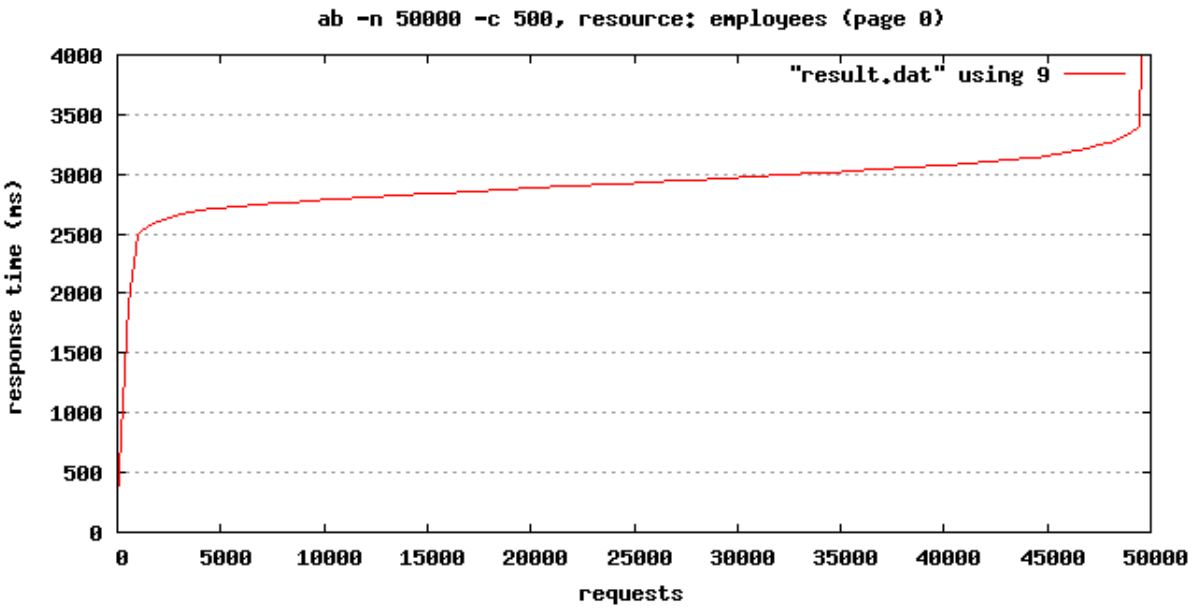

Figure 4.15: Apache benchmarking results

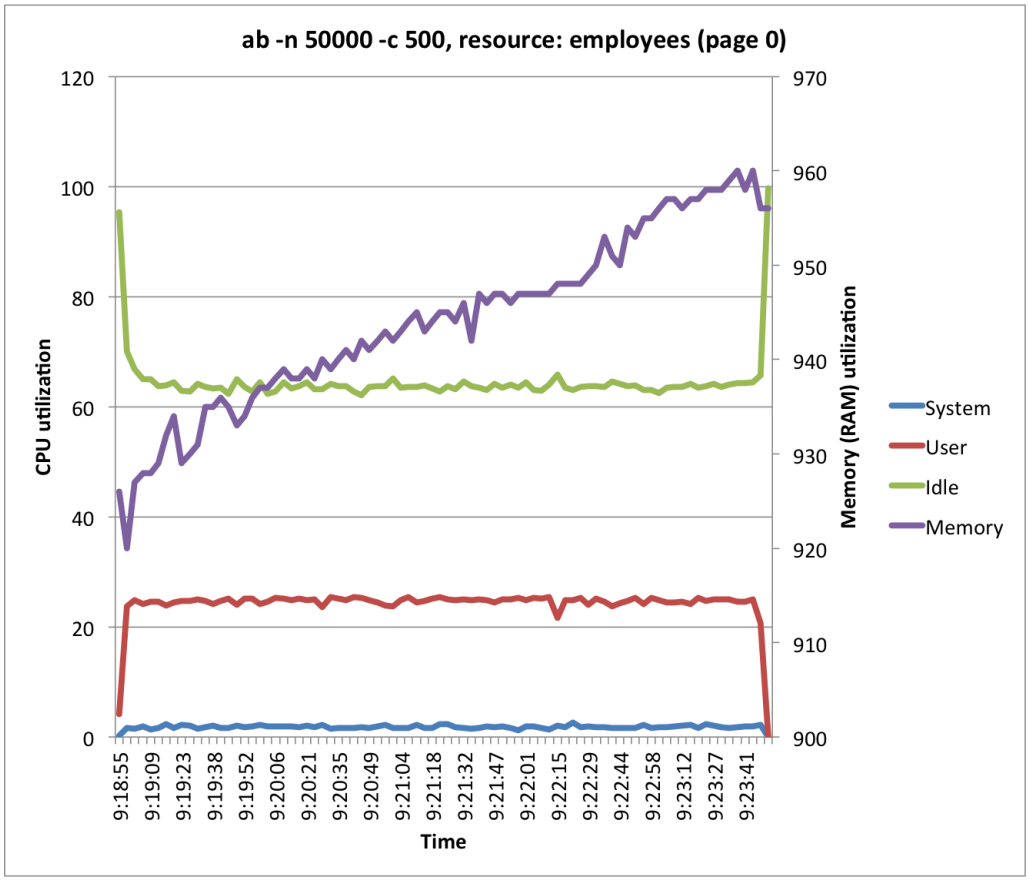

Figure 4.16: CPU and memory consumption during test

4.2.2 No analysis

These are the results from running the benchmark without the analysis module. In table 4.10, we can see the minimum, mean, median and max values of the response times in milliseconds of this benchmark. Figure 4.17 depicts all of the response times in a graph sorted by the fastest response time. The CPU and memory consumption for this benchmark are shown as a graph over time in figure 4.18.

Table 4.10: Results from apache bench, no analysis

	Min (ms)	Mean (ms)	Median (ms)	Max (ms)
Connect	0	10	1	3027
Processing	169	1908	1908	5120
Waiting	169	1908	1907	5120
Total	187	1918	1912	6122

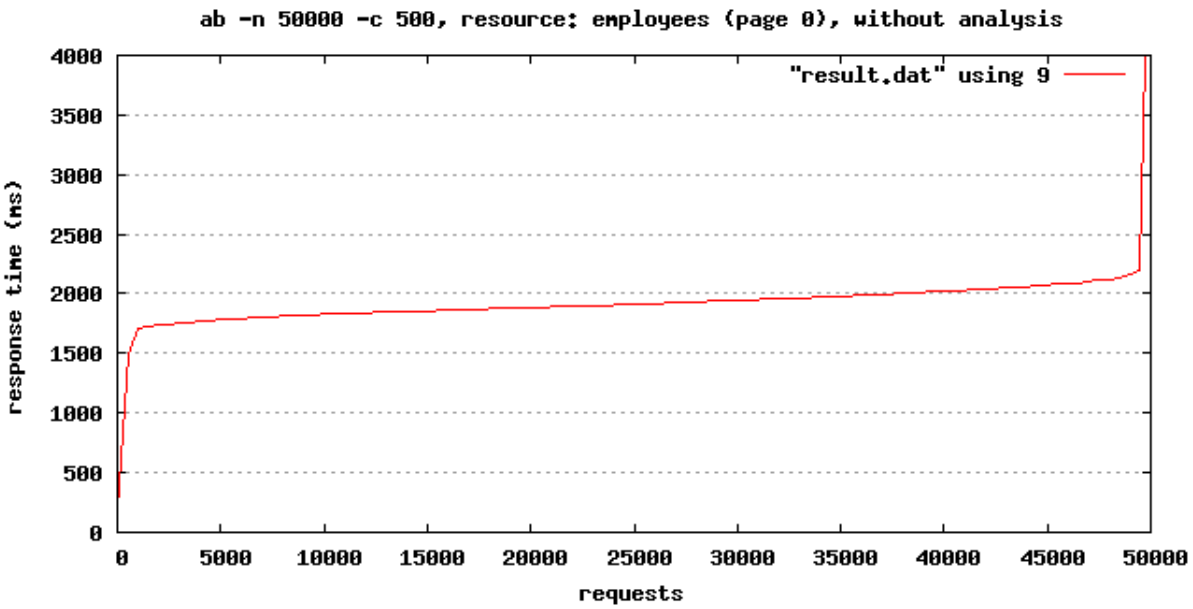

Figure 4.17: Apache benchmarking results

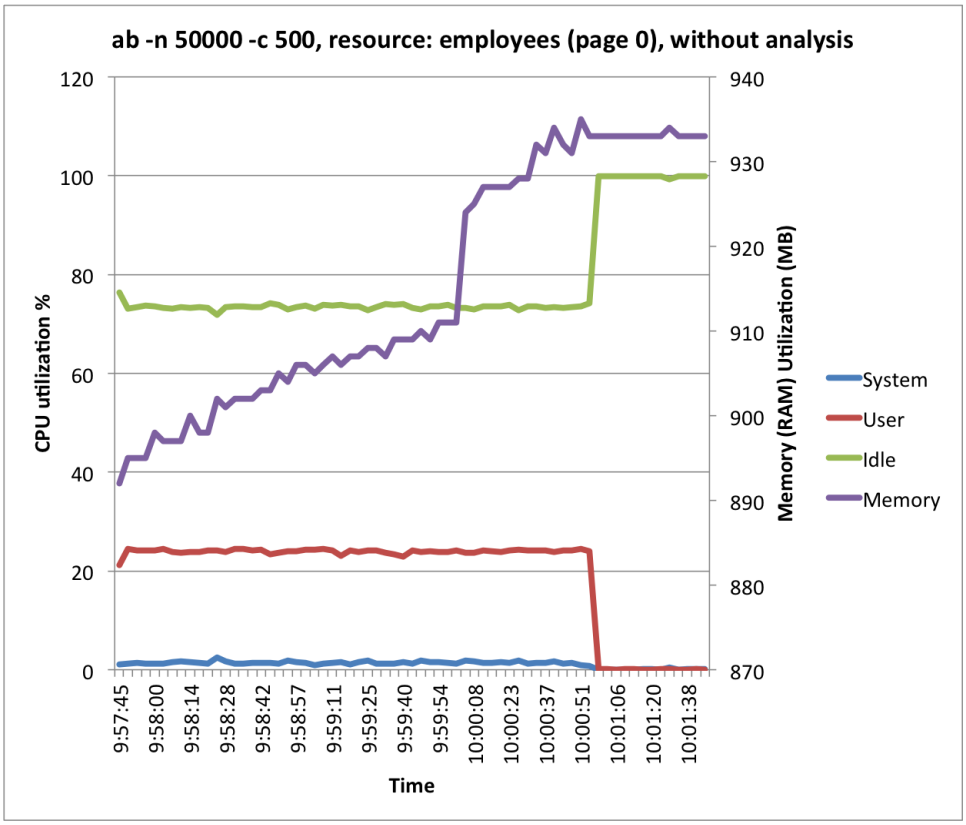

Figure 4.18: CPU and memory consumption during test

4.2.3 Socket fix

These are the results from running the benchmark with only the socket fix applied. Figure 4.19 shows the results in response times from running this benchmark. Table 4.11 shows the minimum, mean, median and maximum response times. Figure 4.20 shows how much CPU and memory was consumed during this benchmark.

Table 4.11: Results from apache bench, socket fix

	Min (ms)	Mean (ms)	Median (ms)	Max (ms)
Connect	0	28	1	8600
Processing	154	2993	2991	4093
Waiting	153	2992	2990	4092
Total	176	3020	2994	11805

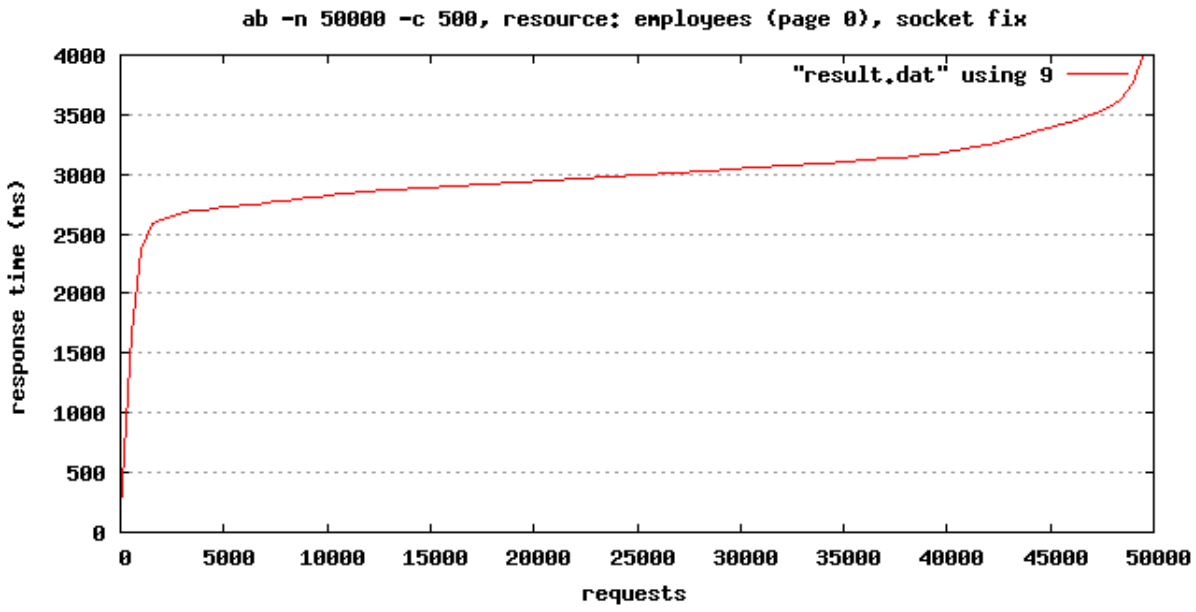

Figure 4.19: Apache benchmarking results

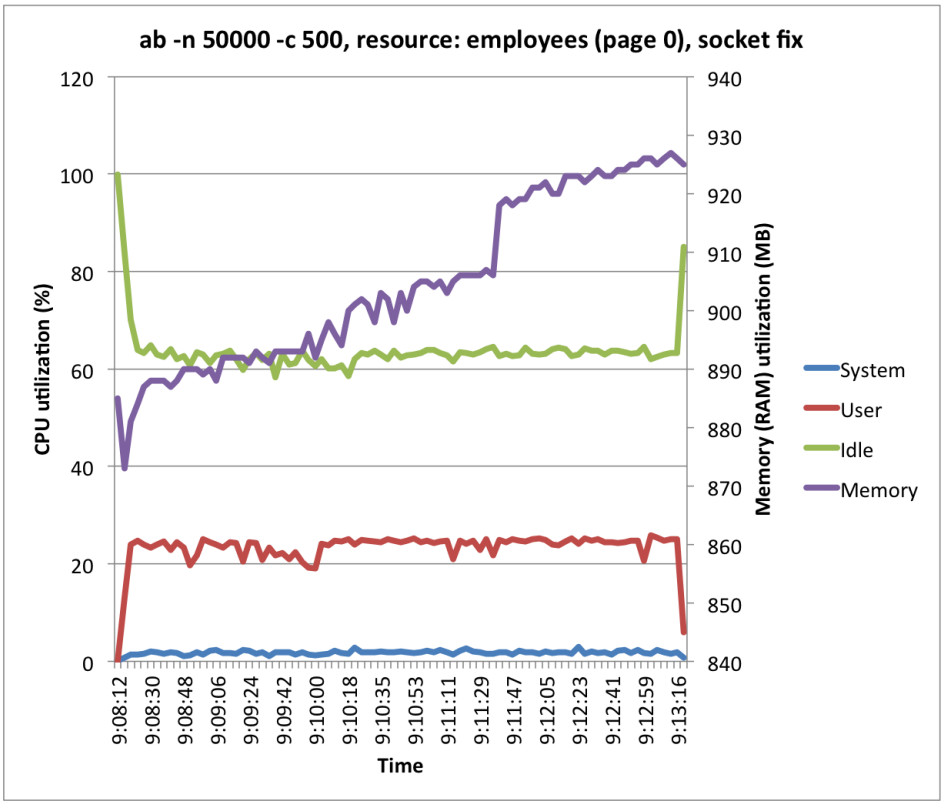

Figure 4.20: CPU and memory consumption during test

4.2.4 Socket fix and no analysis

These are the results from running the benchmark with socket fix enabled and without the analysis and clustering module. In figure 4.21 we can see all of the response times for this benchmark. Table 4.12 tells us that the maximum total time for a response was about 10.6 seconds, while the lowest maximum response time was 218 milliseconds. Figure 4.22 shows the CPU and memory consumption during this benchmark.

Table 4.12: Results from apache bench, socket fix and no analysis

	Min (ms)	Mean (ms)	Median (ms)	Max (ms)
Connect	0	17	1	8613
Processing	191	1903	1912	5497
Waiting	190	1902	1912	4620
Total	218	1920	1915	10606

Figure 4.21: Apache benchmarking results

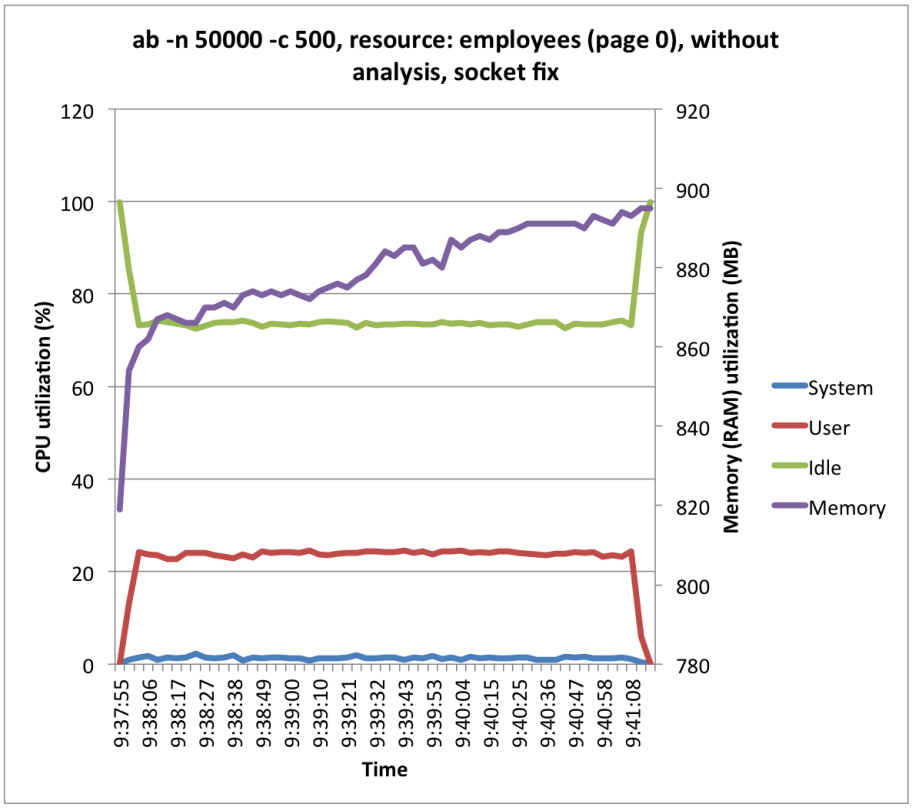

Figure 4.22: CPU and memory consumption during test

4.2.5 Socket fix, no analysis and cluster

These are the results from running the benchmark with the socket fix applied, without any analysis and with clustering module enabled. Figure 4.23 shows the resulting response times from running the benchmark with the above mentioned settings. Table 4.13 shows that the maximum response time was under 37 seconds, while the fastest response time was in 8 milliseconds. Figure 4.24 shows the CPU and memory consumption for this benchmark.

Table 4.13: Results from apache bench, socket fix, no analysis and clustering enabled

	Min (ms)	Mean (ms)	Median (ms)	Max (ms)
Connect	0	178	4	15034
Processing	7	767	510	36734
Waiting	7	719	509	20930
Total	8	945	599	36735

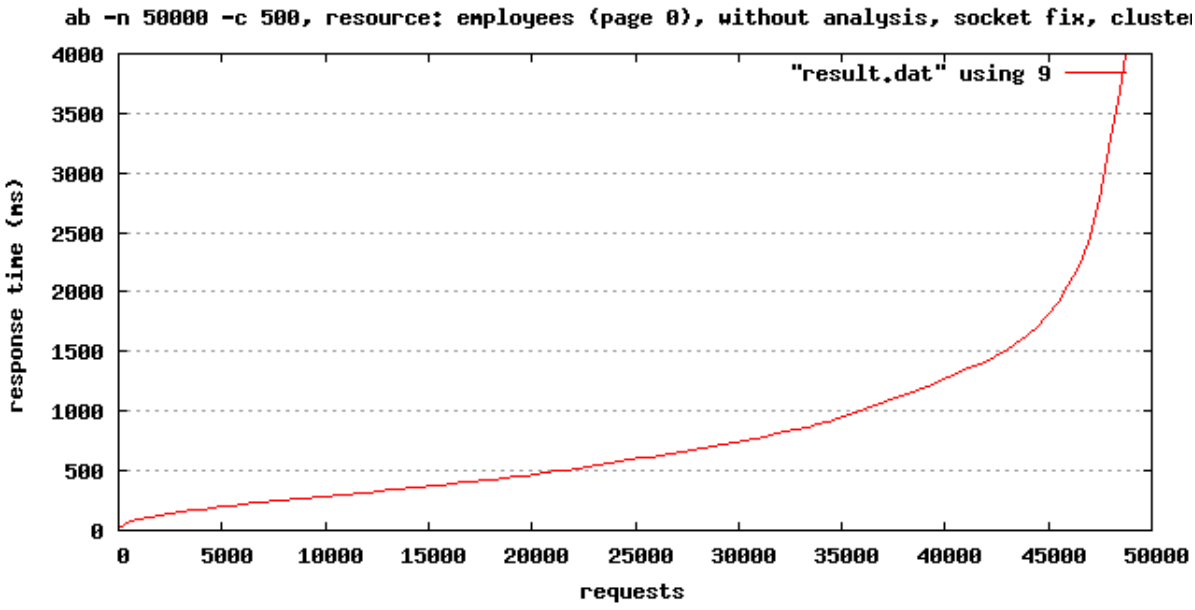

Figure 4.23: Apache benchmarking results

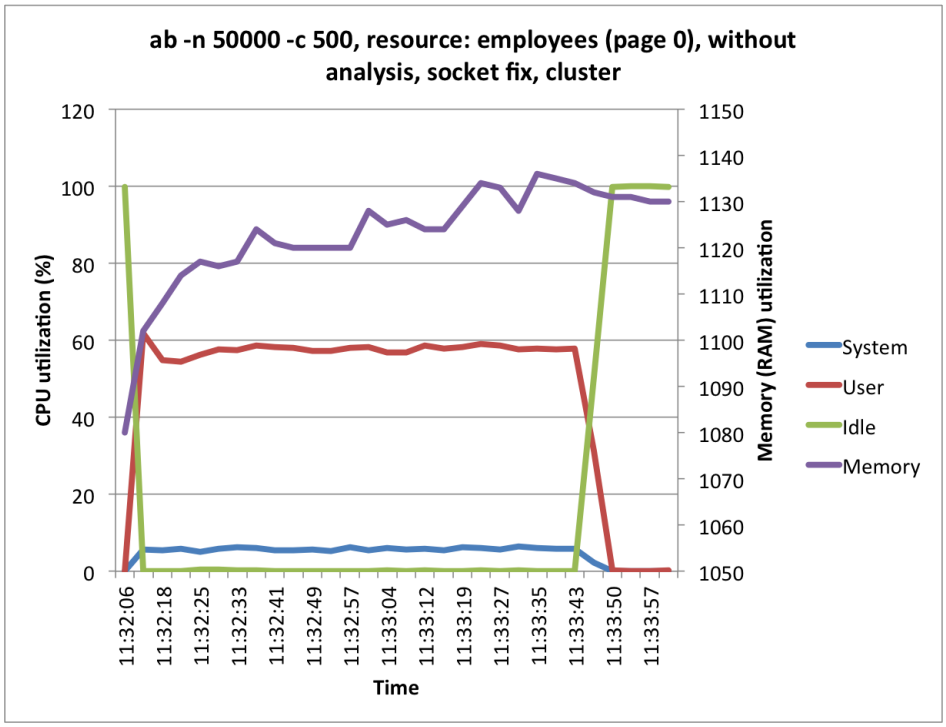

Figure 4.24: CPU and memory consumption during test

Chapter 5

Analysis

This chapter focuses on analysing the results of the tests reported in the previous chapter and assessing the more qualitative goals. Section 5.1 considers each goal and analysing each goal separately. Section 5.2 analyses the system as a whole. The focus in section 5.2 is primarily on how the different components work together.

5.1 Goal and requirements analysis

Figure 5.1 shows a mind map of all of the goals and requirements. The figure will be the basis for analysing whether we have achieved the goals and met the original requirements. The branches are coloured green, yellow, or red. These indicate wether the goal has been fully fulfilled (green), partially fulfilled (yellow), or not fulfilled at all. Each goal will be analysed separately together with the results from the measurement framework. Section 5.1.1 will analyse the goals for the mobile application. Section 5.1.2 will analyse the back end and the goals for it. Section 5.1.3 will go through the goals for the information source in this thesis project.

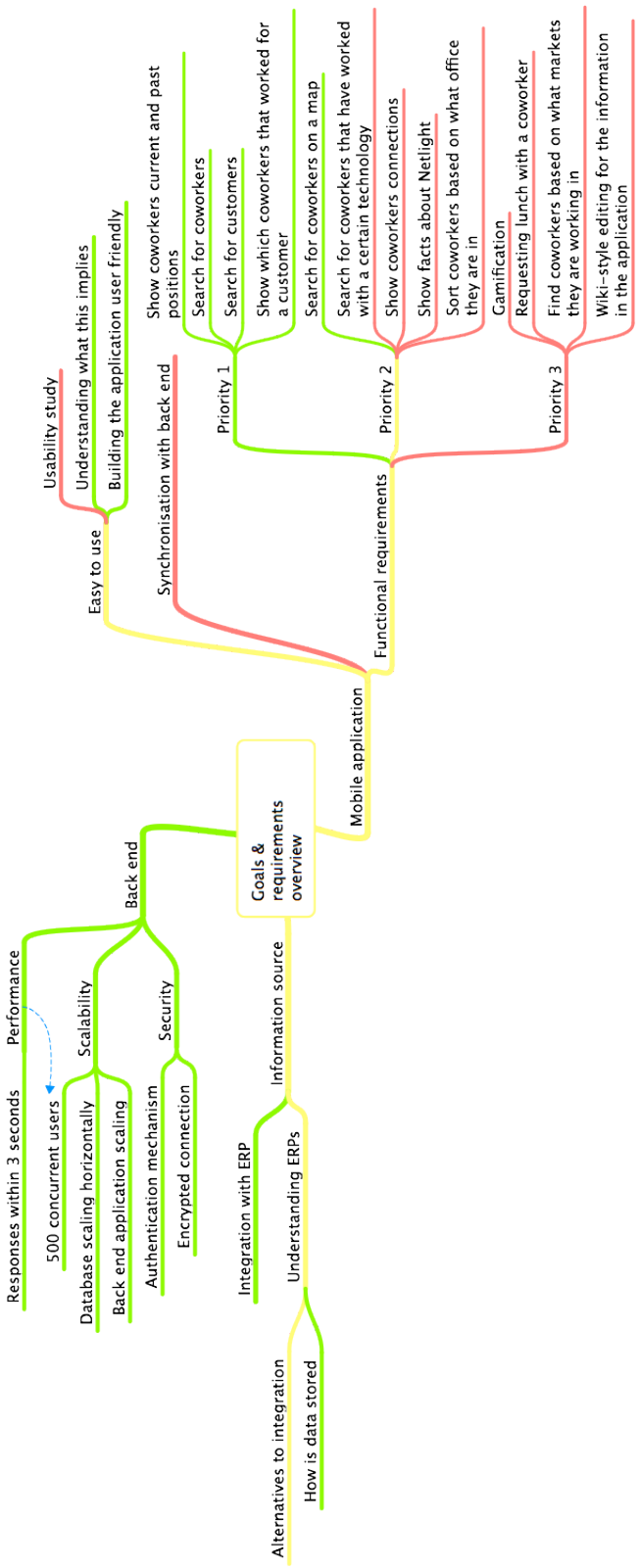

Figure 5.1: Mind map of the goals and requirements set up for this thesis project

5.1.1 Mobile application

The goals for the mobile application consist mostly of the *functional requirements* for the mobile application. These goals had different priorities as assigned, collaboratively with Netlight. All of the four priority 1 goals were achieved, as well as one priority 2 goal (search for coworkers on a map). The user can easily navigate within the mobile application and see where coworkers are and have been in the past, along with a search function.

Another major branch of the mobile application's goals was synchronisation with back end. The design for how this mechanism should be built is outlined in section 3.3.3. However, no practical implementation was done, therefore this goal was not fully fulfilled. Moreover, the solution suggested does not solve the problem when there is a lot of data to be synchronised. The proposed synchronisation mechanism assumes that the initial request will return a successful response, because if it does not then no data is returned to the user. This is of course not a good offline scheme, but rather is a scheme for caching data locally and updating it on the fly. Given this design the mobile application should have much faster loading times (depending on how much it needs to synchronise each time), since it will actively try to fetch data before the user has requested it.

The third and last goal of the mobile application was to build an easy to use mobile application. In order to understand what a user friendly application implies, information had to be gathered about usability and mobile application design. This goal was partially fulfilled since information and knowledge were gathered and the mobile application was built with usability in mind. However, more information could be gathered regarding how to design mobile applications. The design and usability knowledge is based upon a usability book which focuses mostly on web experience rather than mobile applications. Another part of this goal was to complete a usability study with five people, however there was not sufficient time to do so and therefore this analysis is purely based on the writer's own opinions, rather a focus group's.

5.1.1.1 Analysis of the resulting mobile application

Looking at figure 5.2, we can see a simple list of employees. According to usability as defined in section 2.3.1.3, views should be designed to

be scanned not read. This implies that the application should have a clear visual hierarchy, in order quickly convey the desired information. In the figure we can see that the visual hierarchy is very clear with one employee in each row, with a corresponding profile picture, which makes it easy to scan since there is literally no reading required. The employee list directly defines where an employee is currently working, by stating that under their name. However, there is nothing saying that the label under an employees name is their current position. This is according to what was stated in section 2.3.1.4, where needless words should be removed. On the contrary section 2.3.1.1, states that the user should not have to think in order to understand. These are two separate sides that would in practice have to be tested with a focus group in order to get a better understanding how the result should look like.

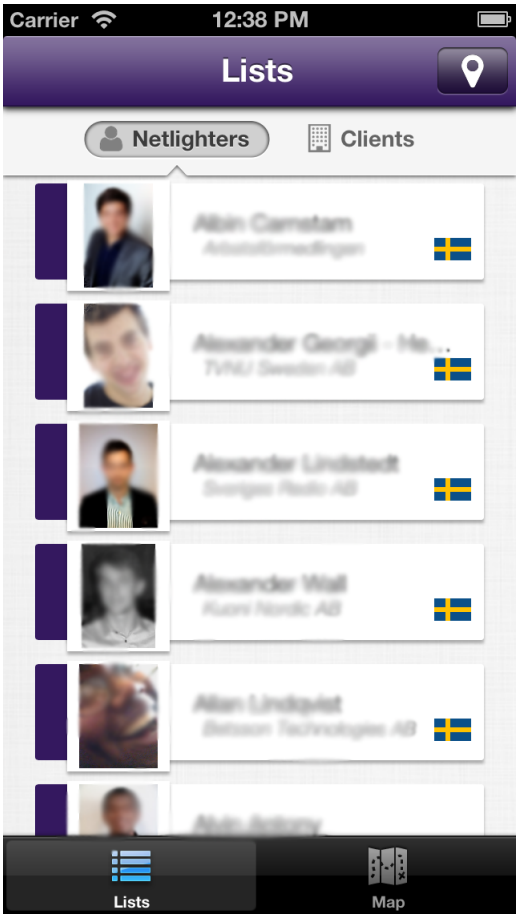

Figure 5.2: Screenshot of the mobile application with a list of employees.

Figure 5.3 shows a screenshot of the list of clients. This list is very similar to the previously discussed list of employes, however this list is even better from a usability perspective. This list strictly follows a design where the material is to be scanned rather than read, which is exactly for what is needed for a big list. The list has icons to its left, which tells the user whether there are multiple people or just one person working at the specific client. This gives a very fast, scannable view of the information. However, this view is not ideal from a usability point of view. The clients names are written text, but could have been something that represents that client, such as a logo. The usability would have increased, if we can assume that these companies' logos are known to the user.

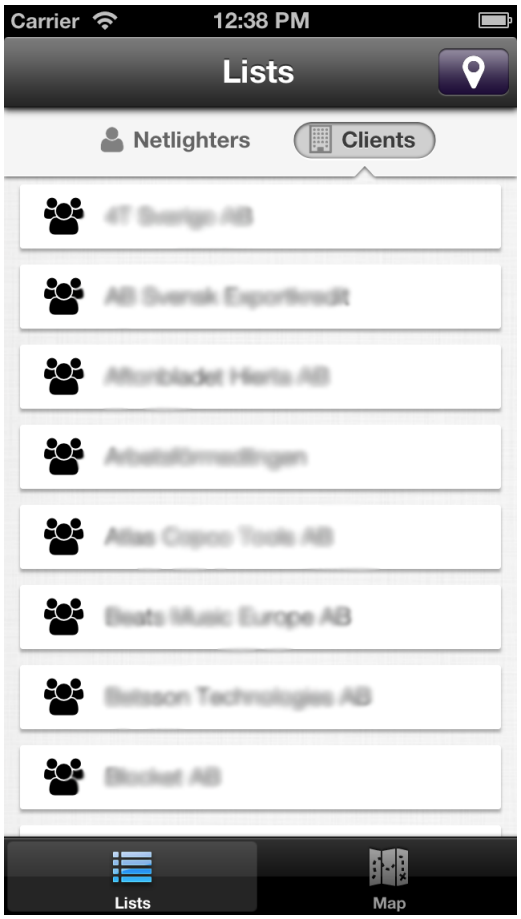

Figure 5.3: Screenshot of the mobile application with a list of clients.

The search bars which exist at the top of the above mentioned lists can be seen in figure 5.4. The search bar is not visible by default, as the user

has to scroll upwards above the list of clients. However, in order to give a hint, when the content is loaded the search bar is visible for a fraction of a second before it slides up and hides itself. A focus group could give more insight in how usable this feature is.

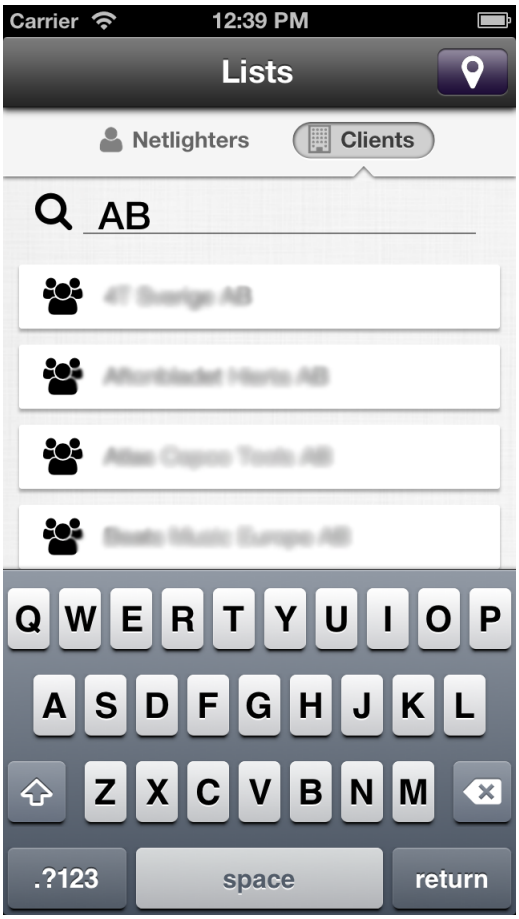

Figure 5.4: Screenshot of the mobile application with the search field for clients.

The last two screens are the map view, where coworkers can find each other on a map as depicted in figure 5.5 (zoomed out) and figure 5.6 (zoomed in). This screen is mostly straightforward, the user can see small icons representing humans at locations on a map. Since many of the employees at Netlight are located near a small number of positions, such as in the city of Stockholm, the map will become very crowded with small icons. Therefore clustering functionality was added, such as depicted in figure 5.5, where the icon represents multiple coworkers at one location.

As the user zooms in on the map, the icon which represents a group of people is split into single icons when the user has zoomed in sufficient to view each of the group's members. The core functionality of this screen is very easily implemented and understood, however something that might not be so straightforward to implement is the ability to click the icons to get further details about one specific employee. However, this is a standard behavior in the iOS operating system, thus the probability of users understanding this feature higher.

Figure 5.5: Screenshot of the mobile application with the map zoomed out.

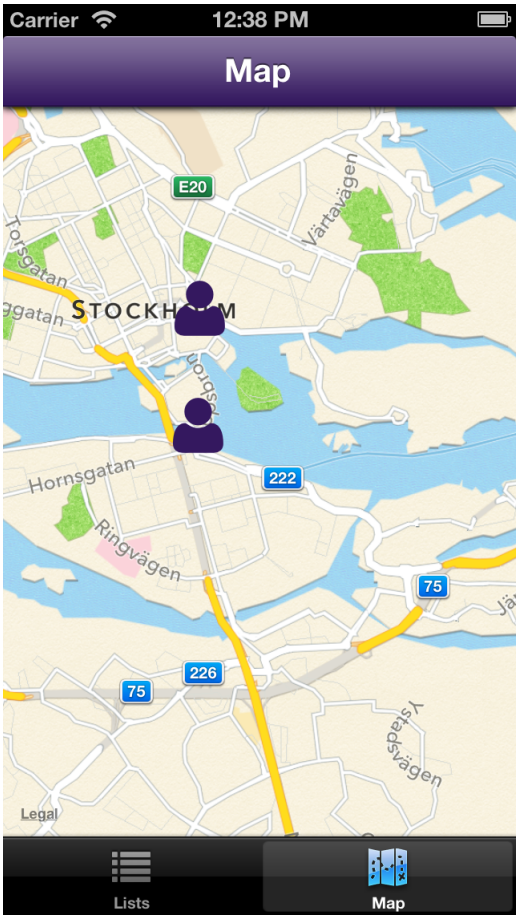

Figure 5.6: Screenshot of the mobile application with the map zoomed in.

5.1.2 Back end

The back end had a lot of goals and requirements, mostly because this is the point that integrates with the rest of the IT infrastructure of the company. If the back end has poor performance or poor security, then the back end might not be allowed to run at all, therefore the back end needs to meet a certain standard of both performance and security. On the other hand if it performs well, other client platforms might connect to it in the future, such as Android or Windows phones. There are three main categories of goals that were set for the back end: performance, scalability, and security. The following sections go through each of these goals and analyses them in depth.

5.1.2.1 Performance

The performance goal has one main requirement that has to be met, which is that the response has to be generated in no more than 3 seconds - even while the system is heavily loaded. The load specified in the requirements is 500 concurrent connections. Since the requirements are not explicit, we assume that if the response time is no more than 3 seconds 95% of the time, then the goal is met.

The plots in chapter 4 shows the back end's performance with different parameters, in order to get an understanding of how the back end can handle different loads. There are three items of information shown for each run that was made: the statistics generated, the response times for this number of requests, and the CPU and memory load on the back end. There were in total 12 different runs, which were both local and remote and they varied with respect to 4 parameters: the resource that is fetched, analysis in the back end, socket fix, and clustering.

The results for each run shows the connect, processing, waiting, and total time in milliseconds. The definitions of these times are as follows:

Connect time. The connect time is the time it takes for the apache benchmark to open a connection to the back end.

Processing time. The processing time is the time it took for the server to generate a response.

Waiting time. The waiting time is the time between the last sent byte of the apache benchmark and the first received byte from the back end.

Total time. The total time from the beginning of the connection until the connection is closed.

The plots for each run were based on the total time for each request and are not serially sorted - but rather sorted on total time, thus the maximum total time is the rightmost value, whereas the minimum is the leftmost value.

As we can see the maximum values are much higher than the mean and median values. This applies for all runs except for the local run. The local run differs in that it is more normally distributed, hence the median and mean are closer than they are when running the bench remotely. We

can also see that the CPU is used more than during the local benchmark compared to the remote benchmarks.

This phenomena can be explain by observing the virtual machine host. The bench mark was run on one physical machine with two virtual machines (except for the run which was local). In between the two virtual machines is a virtual switch. The virtual router handles the traffic that goes from and to the virtual machines running on that host. When observing the virtual machine host during the remote benchmarks compared to the local benchmark, one can see that the virtual switch is utilising the CPU at its maximum during remote benchmarking, as compared to no CPU usage when benchmarking occurs locally. Running the benchmark locally does not require the virtual machine utilise the virtual switch, therefore the distribution of response times is smoothed.

The CPU utilisation has three different measures: idle, system and user percentages. The idle percentage is the fraction of CPU that is utilised when the computer is idling (i.e., not working), the system percentage is the fraction of the CPU used when the operating system is executing and the user percentage is the fraction of the CPU used by user applications. The back end was started by a user and thus it is a user process. A factor that seems to apply to all benchmarks is the fact that not using clustering in the back end does not allow the CPU utilisation to exceed 25% of the total CPU utilisation. This is of course only natural since the processor has four cores and without clustering the back end is only running on one core. Enabling clustering gives a bit better performance as it allows the back end to utilise all four cores. However, due to the fact that the virtual switch was limiting traffic, the CPU resources of all four cores were not fully utilised.

Another interesting factor is the memory utilisation, which is of course important in order to know what the requirements will be for a production server. Running the benchmark seems to increase the system memory utilisation by about 50-200MB. There also seems to be a correlation between disabling the analysis in the back and and lower memory consumption. This could be due to less operations being queued up in the back end, for storing analytics. However, this is not always the case as it seems also to depend on the resource that is targeted. It seems that targeting a smaller resource, for example a specific employee rather than a list, results in higher system memory utilisation. This can be explained by the fact that the smaller resource is faster to load and thus the ratio between time spent on analytics and time spent on loading

the resource is higher with a smaller resource.

Applying the socket fix does not seem to have made a big difference. This can be due to two different reasons (1) the socket fix was not properly configured or there were other limitations in the system which were not changed or (2) because the socket fix is simply not needed as the system recycles sockets fast enough anyway, as 500 concurrent connections are not that many after all (in comparison to the nearly 10 times this number of available sockets - on a machine which is only supporting this application).

The most important question to answer is whether the back end fulfilled its goal to deliver 95% of the responses within 3 seconds. Although looking at the results which were run remotely and did not perform as well as they could due to the virtual switch in the virtual host, we can see that with the right tweaks the back end can deliver responses within the time frame. As we can see in figure 4.11, around 98% of the responses are within 3000ms.

5.1.2.2 Scalability

Analysing the scalability of the system is important for multiple reasons, such as following the current trends and building a system for the future. As Netlight is a growing company, it is important to understand what a scalable system is and how to build one. Section 2.2.2 defined a set of steps in order to check the scalability of the system.

The first step was to characterise the system's performance as a function of resources, which means understanding how much more of a resource needs to be added or removed to see any difference in system performance. In the test runs the resources were the same during all benchmarks and therefore we need additional benchmarking to be conclusive in this step. The second step is to identify bottlenecks in the system and identify scaling design architecture violations. An obvious bottleneck is the virtual switch, however it is part of the virtual host. Another bottleneck in the system is the analysis module, which collects statistics for the back end. Disabling this module increased performance for all benchmarks. Running the back end without the clustering module makes the design unscalable, since clustering allows multiple machines to be clustered (or to utilise CPU cores). However, an alternative solution would be to add external load balancers in front of the back end system. Next a SWOT analysis of the back end was done. Given that the databases

are scalable and the clustering module is in use, one of the strengths of this system is that it would probably scale quite well. However, we can only really know if we try to scale the system and examine its scaling, based on step 1. One of the weaknesses of the system is that profile pictures of the employees are proxied (without being cached) through the back end, which creates a dependency upon another system. This implies that if this back end were to scale and the other dependent system did not scale, then performance issues would probably arise. Due to the popularity of Node.js the opportunities of adapting to new technologies is not a problem, since the community is very active and often writes modules to adapt to new technologies. One threat in the back end is the analytics module, as it seems to fail to handle the work load when under pressure. This is something that needs to be kept in mind when running the back end with the analysis enabled.

Another important part of the back end scalability is the database. However, as already mentioned in section 2.4.3.3, the database that is used is able to scale horizontally hence providing scalability for the database.

5.1.2.3 Security

The information that is in the mobile application is business critical and must not leak to anyone outside the company. This was the foundation for which the two goals of security were based upon. The first goal is utilises a strong authentication mechanism, which is implemented and integrated into the local active directory that stores the authentication information. The connection from the back end to the active directory is explained in section 2.4.4.

The second goal was to have only encrypted communication between the mobile application and the back end. This goal was met by setting up an proxy in front of the back end which only allows encrypted connections and to and from the mobile application. This proxy forwards the requests to the back end.

5.1.3 Information source

The information source goals consisted of two subgoals, which were the actual integration with the ERP system and understanding the ERP system. The goals discussed in this part of the analysis are only for the

practical parts of the thesis, rather than theoretical analysis which will be discussed in section 5.2.

5.1.3.1 Integration with the ERP

The integration with the ERP system utilises an SSIS package, which runs every night. In order to get an understanding of whether this synchronisation is sufficient we must first know the time frame of changes to the information that is being synchronised. In our case the information does not need to be updated more frequently than every night, and thus it is acceptable to have this infrequent synchronisation. However, according to the measurement framework (from section 3.1.2) there is another aspect which concerns how long the back end is unavailable. The integration of the data from the ERP system is done by first truncating all database tables in the back end and then filling them up with information. Due to method of synchronisation the back end will be unavailable during this synchronisation process with the ERP system. Although this aspect is important, no goal was set by Netlight as to how much uptime the back end part of the system had to have. In practice the back end will be useless for a few seconds during night time, which is outside of normal work hours for most of the employees.

5.1.3.2 Understanding ERPs

Another goal for this thesis project was to better understand ERP systems, particularly how they store their data and what alternatives there are for integration. This goal is probably the hardest one to quantify, however we can analyse what was actually done to achieve the integration. As we can see in section 2.5.2, we can have a pretty clear understanding of how the data is stored by looking into the structure of the database. This information tells us how data is connected in the database. However, which it turns out that the data is not highly connected as the database is quite normalised as we can see in the number of tables. Additionally, all of the logic for combining data seems to be outside of the database, probably in the proprietary Agresso application. This makes it very difficult to understand what the data means, as we would need to examine the proprietary application or the documentation of this ERP system. Taking into account that the essential parts needed to implement the integration were understood, we deem this goal is fulfilled.

5.2 System analysis

ERP systems are often big systems, since they can contain all of the data needed to run a business. One might expect that the bigger the business the more information to keep track of. What this implies is that many people or systems need to work with the ERP system, since the ERP model is essentially to maintain all the information, even if you subsequently distribute the database back ends. Due to the size and complexity of ERP systems there are huge complications when it comes to scaling them. Some of the complications that can occur are listed below:

Licensing. Based on the business model of the ERP system, licensing complication might arise. For example, having to pay for extra licenses for each server that is running the ERP or paying based upon the number of users that can access the ERP.

Scalability. Scaling these system might not even be possible or might be so complex that scaling is really hard to achieve. Scaling ERP systems is not only about buying better hardware, but performing maintenance on the system. This is one of the disadvantages of having a central node on which a business depends.

New features. Integrating or implementing new features into the ERP system often requires a lot of work. Since the whole business is dependent on the central ERP system, anything that could break it puts the whole business at risk. What this means in practice is that there is a huge requirement for testing in such a manner as to not break anything, setting up frameworks for maintenance, and keeping track of dependencies. In bigger organisations this can take years to accomplish. However, the tests can be done on a mirrored version of the ERP system's database, allowing the production database to run normally.

Given the above, it might be interesting to think of other strategies for utilising ERP systems. An alternative strategy is to do as is done in this thesis project, where the system is running outside the ERP system itself and this system simply works with a copy of the data from the ERP system. Some of the advantages to building systems around ERP systems in this manner are:

Safer. Building the system around the ERP instead of within it can add an extra layer of security. For example, if the system that is outside

the ERP system breaks, it does not necessarily affect the ERP system. In the case of the system built for this thesis project, the system only reads from the ERP system's database. The worst thing that can happen is that the system cannot read from the ERP system's database anymore, then this external system may fail to function - but it will not cause a failure of the ERP system. Building the system outside follows the abstraction pattern (explained in section 2.6) and has the advantages that comes with this abstraction pattern.

Scalability. Building a system which handles one specific task outside the ERP system makes it more manageable, as it does not necessarily have as many dependencies as an ERP system. Another advantage when it comes to scalability is that the system built outside does not have to have the same underlying database as the ERP system itself, which can allow better scalability. The obvious advantage is the fact that scaling a subset of the ERP system compared to scaling the whole ERP system should be more cost effective and more optimised, however this needs to be tested in order to be proven.

No restrictions. Implementing new features or licensing are not issues when building the system outside the ERP system, since this development is independent of the ERP system.

There are many advantages as for building the system outside of the ERP rather than on the inside, however this external implementation requires that the integration options are suitable for the specific system that is being built. For instance a system which requires real time updates of the data, requires a tighter integration with the ERP system, which can get quite complex. A suggestion for building real time synchronisation is to have a direct feed from the database logs which include all operations and then calculate the difference that need to be made to the system outside. Depending on if data needs to be processed before extracted, this solution might be too slow for a real time system.

Chapter 6

Conclusions

Section 6.1 discusses the conclusions drawn based on the previous results and analysis. Section 6.2 describes some ideas for future work as well as explains further in detail what has been left undone.

6.1 Conclusions

The work done in this thesis project involved a number of different areas, that are quite separate in terms of the problems that needed to be solved. In this thesis project there were two main parts: the work requested by Netlight and the theory necessary to perform this work and to evaluate it. The work itself needed to satisfy a number of requirements and we measured the performance of the resulting implementation to determine if we meet these requirements.

The requirements set for this thesis project by Netlight were important to have, as they frequently identified a direction to follow, especially when some of these requirements had lower priority. However, there are some complications that occurred with regard of these requirements as the definition of these requirements need more care than initially ought. The requirements defines the common platform between the product owner and the people working in the project. A lesson learned during this thesis project is that requirements are hard to completely specify at the start of a project. At some point in time the product owner and the people working on the project may have common thoughts and opinions, however, over time this can change. This implies is that it is important to have good communication and feedback paths to the product owner in order to maintain a common platform. It is also important that the requirements

are prioritised and maybe even to reprioritise these requirements as necessary during the project. Continuous communication is key to keep a common vision of the objectives.

Mobile applications require a certain level of usability in order to actually be used, as the user has to see value in spending time using the application. Building a mobile application that provides a high quality user experience, requires a lot of planning and thought. User interfaces need to be understandable without being descriptive, which requires knowledge and understanding of how the human brain works and reacts to different actions, colours, and behaviours. In combination with this the mobile application also has to bring added value to the user and the work that this user needs to carry out. It is important to balance the time spent on developing the application to bring value and functionality versus the time spent on making the application usable. Depending on the user group for the application, this balance could be towards either one of these sides. This balance can be better understood by looking at the time frame and by carrying out a user study, in order to understand what the user's needs are and how experienced the users are with similar applications or interfaces. With this in mind, one might also want to read and understand what usability implies and how others have defined it in their research, as this could provide additional input when achieving a great user experience.

There are more aspects to building a mobile application than its interface, such as the back end connection and how well the application performs on the user's phone, such as if it drains the battery rapidly or not. As discussed in section 2.3.2, the network connection is one of the most important parts to optimise in order to create a mobile application that is usable for long periods of time each day. Therefore the communication between the mobile application and the server was designed to send only necessary information to minimise the network load. We have assumed that by doing this the system should perform better and achieve the performance goals. however neither of these have actually been shown. Another lesson learned when performing the performance tests is that a lot of time needs to be spent in order to understand what it is that the test is showing. Without understanding the test set up and what is really happening it is hard to draw any useful conclusions. An important thing to remember when performing tests is to not make assumptions. Assumptions can create confusion, and when doing larger system tests with a lot of components it may be hard to not

make assumptions, especially with regard to making the tests realistic . However, an understanding is needed to what is important to know and what is less important as this helps produce usable results from the test. The way the back end was built in this thesis project had more advantages than disadvantages (as specified in section 5.2), which makes it interesting to continue the research in building a system around an ERP system. Integrating with ERP systems can be a really hard problem to solve, and requires planning ahead, as well as knowledge about how the ERP system is designed. However, once the integration is in place, there are many advantages for scaling and maintenance of these external systems, which can be attractive in the long term.

6.2 Future work

Section 6.2.1, lists what has been left undone. All of these tasks need to be completed in order to complete this thesis project. Except for what needs to be done finish this thesis project, section 6.2.2 gives insight and suggestions for what could be done in the future.

6.2.1 What has been left undone?

Looking at the system and how it behaves most of the requirements are fulfilled, except for one practical point which is the synchronisation between the mobile application and the back end. This synchronisation is important to better understand in order to estimate how well this type of system could scale and to understand the implications this synchronisation can have on the system. The synchronisation could also be redesigned in order to allow changes in the mobile phone to be propagated back to the back end, however no such changes are made in the mobile application developed in this thesis and is therefore not needed. In order to better understand the synchronisation we should compare the way it is done today (loading data on request) with having a local synchronisation from the back end to the mobile application. This could be implemented by using an existing framework such as OpenMobster (described in section 2.1.1). Perhaps the comparison could utilise the measurement framework developed in this thesis project. Another important aspect of this synchronisation is to get a better understanding of the data flows that work best with the synchronisation

pattern suggested in this thesis project and to suggest improvements. Since the synchronisation pattern has not been tested in practice, it is important to test the design to see what limitations it might have.

The next obvious thing is to continue working on the functional requirements of the mobile application set by Netlight. There are a couple of requirements with different priorities and these should be implemented in prioritised order. However, some of the requirements are quite vague and might need more iterations o the requirements specification.

This thesis project lacks a comparison between the time to integrate with an ERP system as compared to the time it takes to implement functionality directly in the ERP system. Further analysis of this is necessary to address the problem which this thesis project initially set out to solve.

6.2.2 Insights and suggestions for further work

Building a system that uses a small subset of data copied from an ERP system requires knowledge of how that data gets in to the ERP and how it is stored, as well as how it is maintained. If one of these aspects are unknown, then a problem may occur in a system that is built based on the data from the ERP system. When building systems that will utilise the data from the ERP system one should develop a dependency graph. The dependency graph should include dependencies from the ERP system to the system built around it. Using this dependency graph decisions can be made for when to upgrade the systems around the ERP system or to upgrade the ERP system itself.

If the integration is based on the database transactions logs the knowledge of how data comes in to the ERP could be handled directly in the integration, which would decide how to handle such data. However, it is still important to understand how the data is entered to the ERP, to understand what kind of data, and what format that will enter the database. By integrating this way the system does not go down when the synchronisation runs, but rather synchronises in real time. Two things that should be considered concerning this for the future are, how database transaction logs can be filtered and how the logic can be built in order to arrange data in the system outside the ERP. If such a integration can be built for an arbitrary back end database then systems could be integrated in a standardised way. This is where it might get interesting

to use NoSQL databases, as some of them scale better than relational databases as the relaxation of the ACID properties may be acceptable. When the flexible bridge has been made a comparison to the use of a direct connection to the ERP system's database should be made. Based on this comparison conclusions can be drawn as to whether building systems in the way suggested in this thesis project is truly worth while.

6.2.3 Required reflections

In this thesis project we proposed a way of how to utilise mobile phones to spread information from within an internal ERP system. The work done in this thesis is expected to have an impact in further development of utilising mobile phones to distribute information, especially through internal systems. Allowing more and more information in mobile phones, the mobile phone trend is boosted upon, by providing more functionality and thus creating more value. However, this also encourages the users of the mobile phones to take in more information than what is already done. The impact on human lives considering the overflow of information available is something to keep in mind when further developing systems like the one done in this thesis project.

The information that is stored in the internal ERP system is spread in the mobile phone application, which enables any user to extract sensitive business information. This is an important factor because of the risks that this implies and might even need to be considered in the employment contract, making sure that sensitive information does not leave the company through the mobile phone. It is therefore important to understand the impact of spreading such information.

In this thesis project, the impact is potentially going to connect Netlight together by allowing to know more about their coworkers and what history they have. This may enable more internal communication which could make the employees feel more connected to the company and their coworkers.

Bibliography

[1] Cisco, "Cisco visual networking index: Global mobile data traffic forecast update, 2012-2017," January 2013. [Online]. Available: http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-520862.pdf

[2] ——, "Cisco global cloud index: Forecast and methodology, 2011 2016," January 2013. [Online]. Available: http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns1175/Cloud_Index_White_Paper.pdf

[3] H. Mendelson, "Erp overview," January 2013. [Online]. Available: http://faculty.ist.psu.edu/yen/421/erp.pdf

[4] "Openmobster platform services."

[5] S. Long, "Database synchronization between devices: A new synchronization protocol for sqlite databases," Master's thesis, KTH, School of Information and Communication (ICT), Stockholm, May 2011, TRITA-ICT-EX-2011:88. [Online]. Available: http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-46712

[6] M. Meeker, "Internet trends," February 2013. [Online]. Available: http://s3.amazonaws.com/kpcbweb/files/58/KPCB_Internet_Trends_2012_FINAL.pdf?1340750868

[7] MobiThinking, "Global mobile statistics 2012 - mobile broadband," February 2013. [Online]. Available: http://mobithinking.com/mobile-marketing-tools/latest-mobile-stats/b#mobilebroadband

[8] C. B. Weinstock and J. B. Goodenough, "On system scalability, performance-critical systems," School of Computer Science Carnegie

Mellon University Pittsburgh, PA 15213, Tech. Rep. CMU/SEI-2006-TN-012, March 2006. [Online]. Available: http://www.sei.cmu.edu/reports/06tn012.pdf

[9] C. Garrod, "Putting the scalability into database scalability services," School of Computer Science Carnegie Mellon University Pittsburgh, PA 15213, Tech. Rep. CMU-CS-08-150, August 2008. [Online]. Available: http://reports-archive.adm.cs.cmu.edu/anon/2008/CMU-CS-08-150.pdf

[10] MySQL, "Guide to scaling web databases with mysql cluster," February 2013. [Online]. Available: ftp://ftp.heanet.ie/mirrors/sourceforge/b/bi/bibbleitm/doc/MySQL_Cluster_Scaling_Web_Databases.pdf

[11] "ios," February 2013. [Online]. Available: http://www.apple.com/ios/what-is/

[12] MobiThinking, "Global mobile statistics 2012 - mobile device shipment," February 2013. [Online]. Available: http://mobithinking.com/mobile-marketing-tools/latest-mobile-stats/a#phone-shipments

[13] S. Krug, *Don't Make Me Think! Second edition*. New Ruders publishing, 2006. ISBN 978-0321344755

[14] Amsterdamned, "Amazon book review: Usability - for americans," May 2013. [Online]. Available: http://www.amazon.co.uk/Dont-Make-Me-Think-Usability/dp/0321344758/ref=sr_1_1?s=books&ie=UTF8&qid=1360753169&sr=1-1

[15] J. Nielsen, "How users read on the web," January 2013. [Online]. Available: http://www.nngroup.com/articles/how-users-read-on-the-web/

[16] H. A. Simon, *Models of Man: Social and Rational- Mathematical Essays on Rational Human Behavior in a Social Setting*. Wiley, 1957.

[17] W. S. Jr. and E. B. White, *The Elements of Style, Third Edition*. Macmillan, 1979.

[18] A. Carroll and G. Heiser, in *2010 USENIX Annual Technical Conference (USENIX ATC '10)*.

[19] E. Katz, "How to write online mobile apps consuming sap back end systems," January 2103. [Online]. Available: http://scn.sap.com/docs/DOC-30760

[20] A. Inc., "Introduction to core data programming guide," February 2013. [Online]. Available: http://developer.apple.com/library/mac/#documentation/cocoa/Conceptual/CoreData/cdProgrammingGuide.html

[21] Unknown, "About sqlite," February 2013. [Online]. Available: http://www.sqlite.org/about.html

[22] J. In.c, "Node.js," April 2013. [Online]. Available: http://nodejs.org/

[23] Tiobe, "Tiobe programming community index for february 2013," February 2013. [Online]. Available: http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

[24] T. Hughes-Croucher and M. Wilson, *Node Up and Running*. O'Reilly Media, Inc, USA, 2012. ISBN 978-1-4493-9858-3

[25] K. Duuna, "Analysis of node.js platform web application security," Master's thesis, TALLINN UNIVERSITY OF TECHONOLGY Faculty of Information Technology Department of Computer Science.

[26] J. Celko, *Joe Celko's SQL for Smarties Advanced SQL Programming Fourth edition*. Morgan Kaufmann, 2011. ISBN 978-0123820228

[27] R. Cattell, "Scalable SQL and NoSQL data stores," February 2013. [Online]. Available: http://www.cattell.net/datastores/Datastores.pdf

[28] S. Batra and C. Tyagi, "Comparative analysis of relational and graph databases," *International Journal of Soft Computing and Engineering (IJSCE)*, vol. 2, no. Issue-2, May 2012. [Online]. Available: http://www.ijsce.org/attachments/File/v2i2/B0625042212.pdf

[29] R. Shoup, "Scalability best practices: Lessons from ebay," May 2013. [Online]. Available: http://www.infoq.com/articles/ebay-scalability-best-practices

[30] H. V. Ltd., "Sso and ldap authentication," February 2013. [Online]. Available: http://www.authenticationworld.com/Single-Sign-On-Authentication/SSOandLDAP.html

[31] S. Chen, "The concept of representational state transfer (rest)," February 2013. [Online]. Available: http://ceit.uq.edu.au/system/files/fileupload/the_concept_of_representational_state_transfer.pdf

[32] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext transfer protocol – HTTP/1.1," vol. RFC 2616 (Draft Standard), June 1999. [Online]. Available: http://tools.ietf.org/html/rfc2616

[33] Unknown, "Introducing json," February 2013. [Online]. Available: http://www.json.org

[34] E. F. Monk and B. J.Wagner, "Concepts in enterprise resource planning."

[35] Agresso, "Unit4 agresso." [Online]. Available: http://www.unit4agresso.se/

[36] Microsoft, "Microsoft SQL server," May 2013. [Online]. Available: http://www.microsoft.com/en-us/sqlserver/default.aspx

[37] P. V. Roy, "Programming paradigms for dummies: What every programmer should know," February 2013. [Online]. Available: http://www.info.ucl.ac.be/~pvr/VanRoyChapter

[38] J. Hamilton, *Object-Oriented Programming*. O'Reilly Media, Inc, USA, 2002.

[39] Unknown, "Parametric polymorphism," February 2013. [Online]. Available: http://www.pmg.lcs.mit.edu/papers/thetaref/node8.html

[40] J. Deacon, "Model-view-controller (MVC) architecture," February 2013. [Online]. Available: http://www.jdl.co.uk/briefings/MVC.pdf

[41] W. Liu, "Mvc model-view-controller," February 2013. [Online]. Available: http://www.cs.toronto.edu/~wl/csc309/handouts/mvc-rest-gdata.pdf

[42] C. Bauer and G. King, *Hibernate in Action*. Manning Publications, 2005.

[43] J. Nielsen, "How many test users in a usability study?" March 2013. [Online]. Available: http://www.nngroup.com/articles/how-many-test-users/

[44] ——, "Response times: The 3 important limits," March 2013. [Online]. Available: http://www.nngroup.com/articles/response-times-3-important-limits/

[45] S. Depold, "Sequelize," May 2013. [Online]. Available: http://www.sequelizejs.com/documentation

[46] A. S. Foundation, "ab - apache http server benchmarking tool," May 2013. [Online]. Available: http://httpd.apache.org/docs/2.2/programs/ab.html

[47] "Tcp connection states and netstat output," May 2013, article ID: 137984. [Online]. Available: http://support.microsoft.com/kb/137984

Appendix A

Requirements specification

Koala – Requirements specification

Stockholm 2012-05-31

This document contains specifications of all the requirements for the project “Koala”.

TEL +46 8 616 99 40
ORG NR 556575-6227
WWW.NETLIGHT.SE

NETLIGHT CONSULTING AB
BIRGER JARLSGATAN 7
111 45 STOCKHOLM

1 Introduction

This section describes the project, its purpose and also the scope of this document.

1.1 Project overview

The project, which goes under the name “Koala”, is part of a thesis work done by Konstantinos Vaggelakos. Koala will be an application where employees can see where Netlight has worked before and where Netlight employees are seated today. The idea behind the project was born when realizing that this was valuable information to know.

1.2 Purpose

The purpose of this document is to explain the thoughts and values for each requirement set out for the project and also to serve as an explanatory tool for understanding what the Koala project is about.

1.3 Scope

This document will define the requirements specification for this product. This will especially be done more extensively for the core components of the system. The scope of this document reaches as far as briefly explaining the product and its core concept, however this document is to strictly define the product itself.

2 Product description

A product description in context will be given in the following section.

2.1 Core concept

The core concept of this project is to deliver a mobile application that will help Netlight employees to better find and keep track of each other. This includes having an idea of what companies they are seated at as well as were Netlight employees have been seated in the past. This will provide the employees with a better consciousness of Netlight as well as providing a unified feeling across the company.

2.2 Product context

The product does not have a clear connection to any other project maintained by Netlight today. Netlight’s internal systems keep all information needed to get the information that the application will provide. However, this is not easily accessible especially for employees working out of the Netlight office. As mentioned Koala will work together with internal systems in order to provide the information needed, more specifically the internal system Agresso.

2.3 Use cases

The use cases for the product will depend in what department they work, however this can be a bit generalized to get an idea for how the system can be used by different processes.

1. **BC**
    a. Contact with companies can be handled through people that are working or have worked at a certain company.
    b. The system can be used to get a better understanding as for what have been done or not, which in turn can be used for producing showcases.
2. **DC**
    a. Talk to co-workers that are working or have worked at a certain company, to get a kick-start of a project.
    b. See where your co-workers are located to have lunch with them, or have a discussion.
3. **CC**
    a. Get an understanding for where co-workers are located, to easily plan a trip to a company.

2.4 Assumptions

Below is a list with all assumption made to make this project work:

1. Agresso will keep the history about its employees as well as continuously be updated with new information in the future.
2. Future versions of Agresso are integrated with Koala in case the newer versions change in a way to affect Koala.
3. Server space is available for the back end to be running.
4. The mobile application is updated, if dependencies are broken in future versions of the mobile operating system.

3 Requirements

Requirements section will define the priorities as well as explain all requirements within each category.

3.1 Priorities definition

The following priority definitions are intended to give a better understanding of the importance of each requirement, throughout the requirements section.

LEVEL	DEFINITION
1	This priority level is a “must have”, which implies that it is a core component in the product
2	Priority level 2 means that it will bring real value to the product, however it cannot be counted as a core component.
3	Level 3 are requirements will bring value, however not as much as compared to a level 2.

3.2 Functional

REQ. NR.	PRIORITY	REQUIREMENT
1	1	The user should be able to see where their coworkers are working at the moment.
2	1	The user should be able to search for coworkers by looking in a list.
3	1	The user should be able to search for coworkers by searching for a coworkers name or searching for a company
4	1	The user should be able to view their coworkers past to see where they have been working, as a Netlight consultant.
5	1	The user should be able to search for a company and see if Netlight has been working there before and whom that has worked there.
6	2	The user should be able to search for coworkers by looking on a map.
7	2	The user should be able to look for people that have worked with a certain kind of technology. This will be based on Linkedin data.
8	2	The user should be able to look for their coworkers' connections.
9	2	The user should be aware of Netlight and be able to see "fun facts". These fun facts are: Companies where Netlight have the most consultants, the people that have worked in Netlight for the longest time, people that are newly hired and statistics.
10	2	Find coworkers based on which Netlight office they are seated (such as Stockholm office, Oslo office, etc).
11	3	Find coworkers based on which market they are currently working in.
12	3	Add gamification to the app, where connections and use of connections are rewarded, especially female connections.
13	3	The user should be able to request lunch to another coworker.
14	3	The user should be able to insert information in the application, like a wiki to build a more complete information base.

3.3 UI and usability

The application should follow the design of Netlight's branding. This means that the application needs to follow a specific colour scheme, as well as having Netlight logotypes and graphics. Overall the application should have a "Netlight feeling" to it.

The interface should also be easy to use and not require any previous knowledge to use the functionality in a beneficial way. Documentation of the application itself will not be written as it should be straightforward to use.

3.4 Performance

The performance requirements of Koala are the following:

1. When requesting information from the system, it should not take more than 3 seconds
2. 500 simultaneous users should be able to use the system at the same time.

3.5 Maintainability

3.5.1 MONITORING

Monitoring should be enabled, all around the clock. All components should be monitored in the back end, including the web server and database.

3.5.2 MAINTENANCE

Except for upgrading or continuing development the system should be self-sufficient.

3.6 System integration

Koala is going to extend and build upon the ERP Agresso.

Note: TBD how this is going to be done exactly.

3.7 Security

Note: This is not decided, what follows is a suggested plan for securing the system.

3.7.1 PROTECTION

The communication between Koala application and the backend is going to be encrypted using SSL over HTTP (also called HTTPS). This provides server-side authenticity and confidentiality. Together with this the back end is going to log all operation requests coming in.

3.7.2 AUTHORIZATION AND AUTHENTICATION

Since the data that will be available in the Koala application and the system attached to the backend called Agresso contain confidential information there is a need to authenticate and authorize the user. A module in the backend will do the handling of authentication and

authorization together with Agresso. A user will only get a response when authenticated and authorized.

Appendix B

Measurement script

```
#!/bin/bash
dateTime=$(date +"%m_%d_%Y_%H_%M")
OUTFILE=log_$dateTime.csv

function getTop
{
    top -b -n 2 | grep Cpu | tail -n 1
}

function getSysLoad
{
    echo $1 | awk '{print $3}' | sed 's/%sy,//'
}

function getUsrLoad
{
    echo $1 | awk '{print $2}' | sed 's/%us,//'
}

function getIdle
{
    echo $1 | awk '{print $5}' | sed 's/%id,//'
}

function getMemoryUsage
{
    free -m | awk '/^Mem:/{print $3}'
}


for i in {0..10000}
do
     top=$(getTop)
    sys=$(getSysLoad "$top")
    usr=$(getUsrLoad "$top")
    idle=$(getIdle "$top")
    memory=$(getMemoryUsage)
    timeSecond=$(date +"%T")
    echo "$timeSecond,$sys,$usr,$idle,$memory" >> $OUTFILE
done
```

Listing 1: Measurement script used to measure the server cpu and memory utilisation during tests

TRITA-ICT-EX-2013:89

www.kth.se