NTNU

Norwegian University of
Science and Technology

Large Scale Privacy Architecture

Implemenation and Evaluateion

Erik Reimer

Master of Science in Computer Science
Submission date: June 2016
Supervisor: Guttorm Sindre, IDI
Co-supervisor: Carl-Fredrik Sørensen, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Abstract

Various enterprises store information and transaction data about people who have, or have had, customer or client relationships with them. In addition, many people today use numerous online services where they disclose information like: name, email, address etc. This information may be misused and thus hurt the person's privacy. The data on its own is no threat to people but through profiling, this information may become sensitive.

This project introduces an architecture and a system called IDMegler that tries to mitigate these threats. There is an increasing need to protect data in a fast growing digitalised world. IDMegler is a proposal for a nationwide middleware system which decouples identity data from transaction data, and enforces integrity of certain information about people. IDMegler also works as an insight service for users that enables people to control who retrieves information about them. The project takes the idea of IDMegler, and through design science, interviews, and a questionnaire, evaluates and calculates threshold values IDMegler will need to satisfy.

The interviews and questionnaire have gathered information about transactions concerning personal information and people's lack of control over their personal data distributed through online services and enterprises. The data collected have been the basis for the estimation of the likely workload for the system, and of required performance in terms of throughput and response times. The design science generates a prototype of IDMegler through a proposed architectural design and requirement specification. Performance tests have been performed to test the design against requirements, and to identify limitations and bottlenecks in the architecture. The architectural design, the requirement specification, and the performance tests have all been developed and accomplished during this project.

Through performance tests the project identifies one bottleneck, and suggests how to remove this bottleneck. The remaining part of the design suggests that the architecture and system is good enough to meet the estimated performance requirements.

Sammendrag

Mange bedrifter lagrer informasjon og transaksjonsdata om personer som har, eller har hatt, kundeforhold til dem. I tillegg så bruker mange personer tjenester på nettet hvor de frivillig gir ifra seg informasjon som: navn, epost, adresse etc. Denne informasjonen kan bli misbrukt og dermed skade en persons privatliv. Dataen i seg selv er ingen trussel for privatlivet, men igjennom profilering kan denne dataen regnes som sensitiv.

Dette prosjektet introduserer et system kaldt IDMegler, som prøver å forhindre disse truslene. I våre dager er det et økene behov etter å sikre data i en kjapt voksende digitaliserende verden. IDMegler er et forslag til et mellomvare system på nasjonalt nivå, som fjerner koblingen mellom identifikasjonsdata og transaksjonsdata, og håndhever integriteten av visse typer person informasjon. IDMegler fungere også som en innsikts tjeneste for brukere av systemet som muliggjør kontroll over hvem som henter ut informasjon om dem. Prosjektet tar utgangspunkt i ideen til IDMegler, og igjennom design science, intervjuer, og en spørreundersøkelse, evaluerer og kalkulerer prosjektet krav som IDMegler må oppfylle.

Intervjuene og spørreundersøkelsen har samlet informasjon om transaksjoner som bruker person informasjon og om personers manglende kontroll over sin egen distribuerte personlige informasjon i nett-tjenester og bedrifter. Dataen som er samlet inn har vært basisen for å estimere den totale arbeidsmengden systemet må takle, og kravet til ytelse for mengden transaksjoner av personinformasjon og responstiden. Ved bruk av design science genereres en prototype av IDMegler igjennom en foreslått design av arkitektur, krav spesifikasjon, og ytelsestester for å identifisere mulig begrensninger og flaskehalser i arkitekturen. Arkitektur designet, kravspesifikasjonen, og ytelsestestene har alle blitt laget og gjennomført i dette prosjektet.

Igjennom ytelsestestene identifiserer prosjektet en flaskehals, og kommer med forslag til hvordan flaskehalsen kan løses. Den resterende delen av designet gir utrykk for at arkitekturen og systemet er bra nok til å holde de estimerte kravene til ytelse.

Preface

This master thesis is written as the last part of my degree of Master of Science at the Norwegian University of Science and Technology (NTNU), carried out in the Department of Computer and Information Science. The project was supervised by my main supervisor Guttorm Sindre, and co-supervisor Carl-Fredrik Sørensen. The master thesis is a continuation of a pre-project executed by me in the fall of 2015 *Privacy Architecture IDMeglers Impact* [36].

Acknowledgements

The thesis work have been conducted with guidance from Professor Guttorm Sindre and Carl-Fredrik Sørensen at the Department of Computer and Information Science (IDI). I would like to thank them for giving me a lot of helpful feedback and guidance during my master thesis. I would also like to thank my family and friends for continues support throughout the project.

Table of Contents

List of Tables

List of Figures

Abbreviations

ACID	=	Atomicity, Consistency, Isolation, Durability
BPMN	=	Business Process Model and Notation
DRM	=	Digital Rights Management System
DSR	=	Design Science Research
DSRM	=	Design Science Research Method
IDM	=	Identity Management System
IKDM	=	Insight Key Data Management
LBS	=	Location Based Service
LRIM	=	Log Request Information Management
PDV	=	Personal Data Vault
PRIME	=	Privacy and Identity Management for Europe
PRM	=	Privacy Rights Management System
P3P	=	Platform for Privacy Preferences
RDBMS	=	Relational Database Management System
SSL	=	Secure Socket Layer
TLS	=	Transport Layer Secure

Chapter 1

Introduction

Everyone is concerned about privacy. That is what people usually say if someone asks about it. However, many people share and expose themselves on the Internet with varying degree of knowledge about the implications. Various enterprises store information and transaction data about people who have, or have had, customer relationships with them. This information may be misused and thus hurt the person's privacy. This project will introduce an architecture and a system that will decouple identity data from transaction data. In addition, the system gives each citizen an insight service about which enterprise use their personal information. The suggested system is supposed to aid people to remain in control of their own personal data.

Enterprise
This project defines the term enterprise as any type of business or organisation and anything in-between.

1.1 Privacy in the Digital World

Threats
People use many services online. By using these services, people often have to give away information about themselves. The information often disclosed include; email, address, name, credit card etc. The problem with disclosing information is that the individual loses control of the information, and the threat of misuse of personal data increases. The increasing distribution of personal information increases the possibility of malicious use and identity theft, or use of profiling to discover personal secrets or weaknesses.

Big data technology is becoming a more developed field. This enables systems to traverse extreme amounts of data and spot patterns about people's behaviour. The more developed methods present a threat to privacy because it is possible to combine information and use

deduction to infer information not given. Barocas [8] argues for the risks big data pose to privacy. He presents the problem that big data is able to classify a person without their knowledge. Social media makes this possible. Cuzzocrea [20] describes different risks to privacy because of big data technology. The risks include missing control of ownership to personal data, the risk of combining information through interpolation and extrapolation, and the problem of exposure in form of specifying whom a person is, based on transactions shared on the Internet.

Security and privacy
Clarke and Roger defines privacy as [17]:

> "*Privacy is the interest that individuals have in sustaining a 'personal space', free from interference by other people and organisations*"

As the quote implies, privacy is something personal and people do not want their information to come astray.

Two terms, security and privacy, are often used in relation to keeping information secure. Three other words describe information security in IT: Confidentiality, integrity, and availability. The ability to keep things safe through confidentiality, make sure that the information is correct through integrity, and has control over peoples knowledge, and the access people have to the information through availability.

1.2 Motivation

There exists many privacy threats. This project tries to mitigate some of the problems or issues. In general, there is no way a person can totally control the distribution of his (personal) information. The information belonging to the individual is shared through Internet services and social media. And, people share information willingly. The reason for sharing is the benefits it brings. By using online services, many everyday tasks become easier and more efficient. All people have to do is agree upon the terms of agreement when using a service. However, if people want to withdraw from the use of the service and get their personal data removed/deleted, they have no simple way to do that. Storage space is not a problem for big enterprises, and the enterprises will not delete potential valuable data.

The current mainstream practice for enterprises is to store both identity information and transaction data. The stored information enables others to identify transaction data related to a person. This poses a threat to privacy. The architecture and system in this project try to mitigate this problem.

In general, people do not want others to know about their sensitive data. According to Norwegian law, there are two different types of personal data (information about a person): personal data, and sensitive personal data. The definitions in the Personal Data Act[1] are as follows [1]:

[1] This document is "Datatilsynet"s translation of the Norwegian law (Personvernsloven §2).

> "*personal data: any information and assessments that may be linked to a natural person ...*
> *sensitive personal data: information relating to*
> *a) racial or ethnic origin, or political opinions, philosophical or religious beliefs,*
> *b) the fact that a person has been suspected of, charged with, indicted for or convicted of a criminal act,*
> *c) health,*
> *d) sex life,*
> *e) trade-union membership.*"

In a big data context, all types of personal information may become sensitive. Through profiling, it is possible to use small amounts of data, not necessarily classified as sensitive, to identify a specific person. Again, the architecture and system in this project try to remove the traceability between the different sources of information. The work accomplished in this project will not solve all problems about profiling, but it will try to tackle the aspect of directly stored personal data. Profiling by saving actions people do online, through tracking their IP and MAC-addresses will still exist.

1.3 Introduction of IDMegler

This project is based on the suggested idea and architecture IDMegler, Sørensen [38], as well as a continuation of a pre-project [36] done in the Fall 2015 (carried out by me). The project did a threat analysis of the system as well as identified stakeholders, and their concerns. Additionally, this project is based on earlier master and pre-project respectively, Drange [21] and Fossestøl [22].

IDMegler is a middleware architecture intending to improve privacy. The system strives to be a broker (tunnel) between Enterprises and services using personal information, and a common database, which stores personal data (An improved version of the Norwegian "Folkeregisteret"). The outcome of the system should bring:

- Increased integrity of information about persons
- Enable an insight service for citizens where they can check which enterprises are accessing their personal information
- Give citizens the opportunity to refuse the enterprise access to their personal data (if the enterprise continues to use information still after the citizen has withdrawn from the services of the enterprise)

Areas of use
The service will work as a key chain that enterprises can use to access customer's information. Each enterprise gets their own ID (to authenticate who they are to IDMegler) and an ID for each of their customers. They will need validation based on how they want to use the personal information to get access to IDMegler. A method/process will handle the task of accessing a customer for the first time and generate a customer-ID. This prevents

an enterprise of accessing information about citizens who are not their customers. The enterprise asks for personal information (name, email, address, phone-number, social security number) through IDMegler by using their own enterprise Id and their customer ID as keys. This enables IDMegler to use the ID's as keys to discover who the keys link to, and to retrieve the information the enterprise needs from an improved version of "Folkeregisteret". This also enables logging of transactions. See Figure 1.1 for a BPMN model of the process.

A possible solution for incorporating IDMegler, might be to make people able to generate a temporary ID through IDMegler, and use this as the only sign-up information when signing up with an Internet service. Another approach would be to get the ID automatically using MinID, Facebook or other identification services.

The overall motivations for this system are to:

- Reduce the usefulness of knowing a person's social security number or any key identifying a person thus making it more difficult to perform identity theft, tracking, etc
- Separate the physical location of personal information and the transaction data (avoid the links between transaction data and the person it concerns, to mitigate data leakage)
- Give each citizen an insight service about usage of personal information
- Increase the integrity of personal information (the integrity is increased when all enterprises retrieve the same correct data from the same place)
- Log any service or enterprise using personal information and the amount of data the enterprise used

For enterprises, IDMegler will operate as a lookup service and will only affect functions and transactions that use or need customer information: Name, address, phone number, email, and/ or social security number. IDMegler transports this information from a governmental controlled improved version of "Folkeregisteret" to the consumer. IDMegler only stores the logs of the transaction performed and transports the personal information. IDMegler never stores the personal data requested.

To summarize:

- The system authorises insight
- IDMegler exposes services that allows enterprises to access only information about their customers
- The system permits only access to customers the enterprise deals with.
- IDMegler ensures separation of concerns between identifying information and transactions
- IDMegler is a broker (tunnel) between enterprises and an improved version of "Folkeregisteret"

IDMegler assumes that the enterprises using their services do not save the personal information in any local register. Each time an enterprise needs data about one of their customers the enterprise will have to make a request for the information through IDMegler.

Scenarios of use
Many different scenarios will include the use of personal information. In fact, they roughly categorize into the following temporal categories: Daily, weekly, monthly, yearly, and occasionally.

Frequency	Description of Scenarios
Daily	Use of online banks Use of social media
Weekly	Use of online banks Sending newsletters
Monthly	Sending pay checks Sending Bills Sending newsletters
Yearly	Tax returns
Occasionally	EU car control Background check Signing up for insurance

1.4 Problem

The project will try to answer the following research questions.

RQ1: What are possible bottlenecks and limitations of IDMegler?
RQ2: What is the minimum amount of traffic a system like IDMegler should be able to handle?
RQ3: How can the architecture be designed to cope with the workload?
RQ4: How can the architecture fulfil the functional requirements of the solution?

To answer these questions, the project designs an architecture of IDMegler, implements a prototype of the architecture, and tests the system with regards to performance based on the estimated threshold values the system has to operate under. These threshold values have been estimated based on transaction data from ”SpareBank 1” (the finance sector) and assumptions about other relevant enterprises. The data relevant to this project were information about transaction usage and number of enterprises that handle personal information. This project has also made a survey about the average person’s number of customer relationships.

1.5 Contributions

Through data gathered throughout this project, the estimated number of transactions per second Norway will need is 4 500. The questionnaire used to classify the average person's control over different distributed personal information suggests that people in general have no knowledge of all the places the information is distributed. This project specifies the requirements, designs an architecture, and implements a prototype of the architecture. The discoveries suggested that the architecture with a small set-up is able to handle around 1 000 transactions per second, with good response times. Due to the scalability of the architecture, the conclusion is that the design is able to handle far more than the estimated value. The project also classifies a bottleneck with the generation of logs and suggest solutions for the architecture.

1.6 Structure

This report uses the following structure to explain and display how the work is done in this project.

- Chapter 1
  - Introduces the motivation for the project and describes IDMegler, and ends with listing the research questions for this project.
- Chapter 2
  - Introduces the literature and related works.
- Chapter 3
  - Presents the different research methods and why this project uses the different methods.
- Chapter 4
  - Defines the requirements, identifies stakeholders and roles, and goes into detail about the different possible designs in this project. The chapter continues by showing the final design of the architecture implemented in this project.
- Chapter 5
  - Discusses and explains the implementation of the design and architecture given in chapter 4.
- Chapter 6
  - Describes the results of the work done in this project. This include the tests the system implemented, calculations of threshold values, data from the questionnaire, and data from the interviews.
- Chapter 7

  - Discusses the whole project and lists the overall findings of this project. At the end of the chapter validity is discussed.

- Chapter 8
  - Concludes the project and recaps what has been discovered in regards to the research questions. It also lists possible future works.

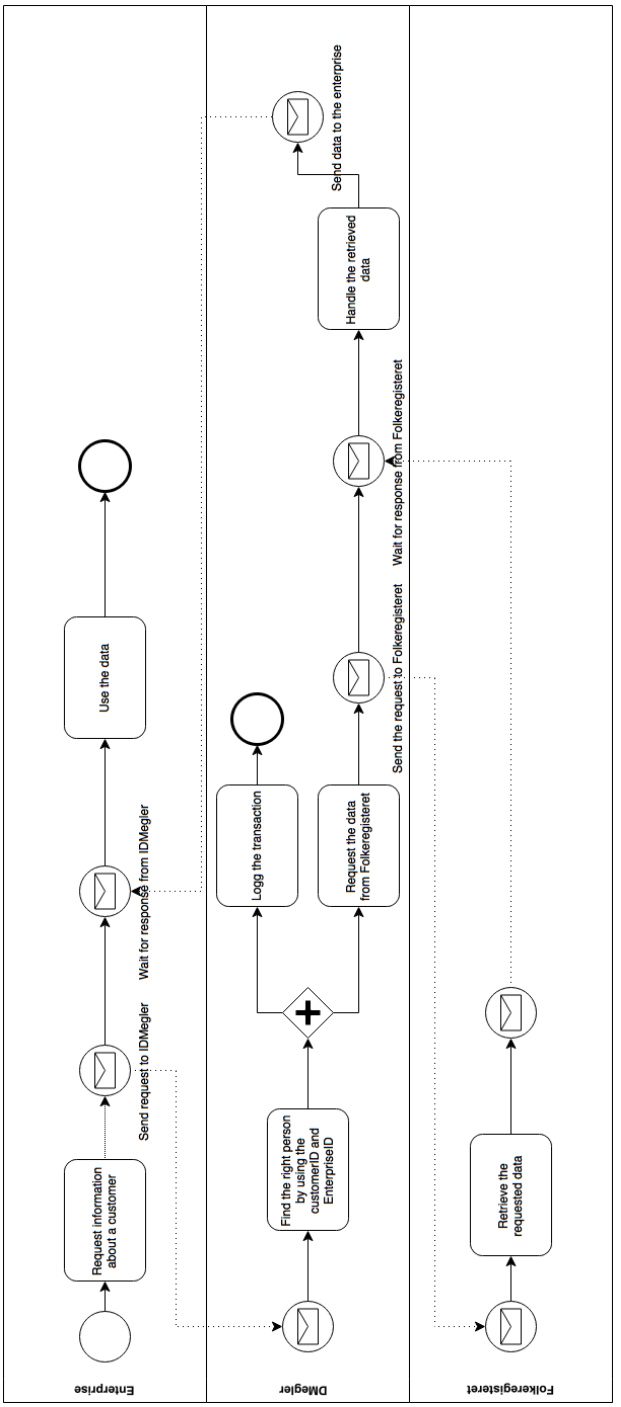

Figure 1.1: BPMN Model of the IDMegler Process

Chapter 2

Literature Review

A variety of articles describing the problem domain has been investigated to better understand the context of privacy architecture and privacy aspects. The primary source of articles has been the search engine ACM digital library. The search strings used have been ”privacy architecture”, ”privacy management”,”privacy management systems”, and ”privacy information systems”. The following sections present different aspects that concern privacy and hold interest for this project.

2.1 Privacy Aspects

Many people find the sharing of information across different services a threat to privacy. Google is a big commercial company with many different services. Corwe et al. [19], went through the privacy policies Google have had throughout the years. The discovery shows that the policies have not changed drastically. In 2012, many discovered that Google’s privacy policies included sharing information among their services. Many thought that this was something new. However, according to the article this has been part of Google’s privacy policy since 2004. The conclusion is that there are little people can do to control their information except find alternatives or to stop using them at all. The reality is that Google holds an extreme amount of information about people who use their services.

Guha et al. [23], suggest a possible way to mitigate the problem with privacy and use of personal information. The approach consists of making insurances for people’s online behaviour. The insurance should compensate economical in the case a person’s data is misused. The main advantage according to the article would be to better mitigate data breaches (leakage of personal information, identity thefts etc.). The article did a survey where they asked people if they would be interested in being insured for their Internet activities. A good number found it interesting. In addition, many where interested in having a software run on their computer monitoring their actions on the Internet, that

would suggest different behaviour when they did foolish choices, to reduce the premium pay.

Recently, people often answer surveys on the Internet. Kandappu et al. [7], present the problem of profiling people based on answers from different online surveys. The article proves how simple it is to generate enough information to infer sensitive information about a person. The solution they present is to use an application that gives the user the option to set their own privacy setting. The survey is then filled out normally. However, when the survey is completed and the answers sent, the info is obscured according to the persons privacy preferences set in the application to mitigate profiling. The article's argument is that the best action to mitigate profiling of combining surveys, is to give the control back to the user.

The government is interested in information about people and is continually working on making services digital. Production of digital solutions emphasise the importance of securing the management of data collection, storage, and analysis without revealing privacy. Vaidya and Jaideep [40] discuss different tactics the government should consider when going digital. The main advice they give is to use Platform for Privacy Preferences (P3P) Project [3] when designing services. In addition, they argue for using good access control to enhance trust of the system.

People use many web services. Consequently, the privacy aspect of e-commerce has received a lot of attention. Ackerman et al. [7] present a survey where they are categorising what a person is willing to disclose of private information. Their findings suggest that people are reluctant to disclose credit-card information and government issued identification. Most people have no problem sharing email address, age, or street address, but they are a little more reluctant to disclose phone numbers. When it comes to sharing information about children, all participant's willingness of share were reduced drastically. Another finding is that people seem to have strong feelings concerning what the gathered information is used for, including the trustfulness of the service.

In general, there are few things people as consumers can do regarding a service. The choices presented are usually to either accept the privacy policies of the service or choose to omit the service. Pettersson et al. [35], suggest a system, which hopefully will reestablish the power of the user. By making a more extensive sign-up form when using a system where the person specifies their privacy preferences, it will be possible to be part of the policy-making. The article is aware of the disadvantage of the increased signing-up time, but argues for the increased benefits of trust it will create. Another strong point is the standardization of specifying privacy. This would make it easier for the public to understand the policies and their implications.

To benefit from many services on the web the individual has to create multiple internet accounts. These spread the information about people to other services. Consequently, each service needs to prove itself trustworthy. Bonatti et al. [13] discuss the problem, and suggests the need for a better certificate driven framework. The idea is to have certified certificate authorities who distribute the certificates. In addition, clients need to request certificates from the servers to validate the trustworthiness of the service. This should happen before sending any information from the client to the server. Given the opportunity

to get information about the actual service, facilitates the protection and evaluation of one's own privacy.

Would it not be better if a service used a minimum of information to ensure privacy? Hansen et al. [24] imply something different. When many small services contain private information there is no guarantee of the security. In addition, little information about a person complicates the detection of patterns. Patterns are beneficial in order of securing misuse of personal information. Knowing a person makes it possible to detect posers. In other words, the important parts of developing privacy systems are to evaluate privacy out of context, and not just based on initial thoughts to minimize information to protect the target.

Massive amounts of people use the web and its services because of its usefulness. Malicious people are among those, and the web provides no safety. Even if people do not use services on the Internet ones privacy is not assured. As long as a person uses the Internet he becomes subject to tracking. Acar et al. [6] present two types of tracking, which are very hard to mitigate. The types are; canvas fingerprints and ever-cookies. Canvas fingerprints are using built-in functionality in most browsers. These make images of a person's websites through the canvas method. By analysing the images, the mechanism generates a profile about a person. Ever-cookies are cookies, which are very hard to clear. Ever-cookies use multiple storage vectors. They are less transparent and concealed for the user. Many big websites use ever-cookies. The findings of the article suggest that there are no ways for regular browsers to fight against these kinds of tracking, and the only working solution today is to use the Tor browser[1].

There exist many threats to a person's privacy. Many identity management systems (IDM) exist. Clau et al. [18] suggest a way to evaluate IDM's, and takes inspiration from the PRIME approach (Privacy and identity management for Europe). To evaluate IDM's the article propose a simple attack classification scheme. The article suggests that there is a need to focus on the weakest part of the IDM's, which they classify as the release data (the data a person sends over the Internet or otherwise instantiates) and the meta-data during interactions. These parts make up the main attack surface of the IDM's. In addition, there are three other primary targets for attacking IDM's; their statistical databases, network anonymity, and interactivity.

2.2 Related Work

IDMegler's focus is to increase privacy and transparency to the public. This section will present other proposals in the same direction of IDMegler. The main difference between IDMegler and other works is the overall scope. IDMegler aims for every aspect of a whole nation and all of its services and enterprises, compared to the e-commerce side of the spectre. The e-commerce has received a lot of attention without proper consideration of the nationwide aspect.

[1] https://www.torproject.org/

A common problem when using e-commerce is, there is no guarantee of how information about a customer is spread or dealt with. Chong et al. [15] imply the need of a protocol that hides the customer's purchases. The idea behind the protocol is the use of a server handling the keys in purchasing an item. Here the customer contacts the server with identification, and then the server manage the exchange. The transaction works without notifying the enterprise of the item purchased by the given customer.

E-commerce also have the problem of profiling. Such et al. [39], suggest a method for dealing with profiling. In addition, they present some of the challenges with profiling. According to the article, profiling can result in price discrimination. This means that two different people with different profiles may see different prices on the same product or service based on their profile. To avoid this scenario, the article suggest using pseudonyms. Even though they are used today, the article argues that pseudonyms are not changed fast enough. Ideally, the pseudonyms should change during each transaction, to mitigate the profiling aspect.

Mobile smart phones are becoming an increasingly important aspect of the everyday life. There are few ways today to control the data that people willingly gives away using mobile applications. Mun et al. [30], present personal data vaults (PDV) that are designed to give the ownership of personal data back to the user in smart phone scenarios. The PDV-architecture uses three different techniques: Granular ACL, Trace Audit, and a Rule Recommender. Granular ACL restricts the value of the given information. Granular ACL manages this by reducing the frequency, and amount of information uploaded to a service. Trace Audit is a logging mechanism that wants to achieve the same result as IDMegler (that is to log any service using information and the amount of data used). Rule Recommender is a high-level interface for setting sharing policies.

People use social media every day. Dhia et al. [11], introduce a privacy management system called Primates. The system is designed to improve the access control deployed in social media. The article argues for the need of more sophisticated access controls in social media. The Primates system makes nodes between the connections in social media, and weighs the nodes according to trust. According to the article, this enhances people's ability in controlling shared information in social media.

People use many mobile devices, and this brings location-based services (LBS) into the equation. Chow et al. [16], look at different architectures for mitigating discovery of people's location while still using LBS's. In a client-server architecture, they argue for using false requests to hide the user's actual location. In a trusted third party system, the solution might be to use middleware to conceal the different requests. If the number of users requesting locations is high enough, it will continuously hide the receiver. The disadvantage is that the third party continuously is recognizing each individual. In a distributed scenario, each user collects the request and sends them in as one individual. In a peer-to-peer setting, the users can use close neighbours to send their request to hide who they are. The article concludes that there are challenges to be dealt with. One of the challenges is to find out how privacy in LBS looks from the perspective of the user. In addition there exists no way to measure privacy in LBS. Adversary attacks are some of the areas the article finds crucial to clarify.

Basso et al. [10], suggest a privacy reference architecture to enhance privacy sensitive services. The reference architecture is made up of four layers. The presentation layer consists of the visual display for the service. The application layer performs the logic in the service. The privacy layer is responsible for defining and enforcing the privacy policies. The persistence layer saves all the data in the service. The architecture framework was evaluated in: completeness, usability, applicability, and feasibility. The article concludes with the benefit of defining a reference architecture to help to set up privacy in crucial scenarios.

Bodrik et al. [12], argue that people do not want to use services online out of privacy reasons. They suggest an architecture based on P3P, focusing on control, data purpose, and separation of persona profiles. The architecture is made up of a collection of agents with different tasks. Each agent accesses a set of different supporting repositories. The different repositories store different data (privacy preferences, private data, and persona's). The architecture uses Audit trail and history repository for accountability. Each agent and user can manage and control the data in the different repositories. The separation of functionality enables easy replacement of agents. Just like P3P the suggested architecture does not enforce privacy contracts but suggest and communicates with the server to ensure the best option for the user. The downside is that there are no guarantees for using false or incorrect privacy policies. To mitigate the problem, there should be a mechanism to trace the usage of the transaction information, to ensure that the data is used as the policies suggest.

The most related work to this report was found in the work done by Camenisch et al. [14]. Their work presented an architecture with the same functionality in mind as IDMegler. The main difference between IDMegler and the work they present is the scope. This article only looks at e-commerce systems. The article suggests making a middleware service which acts as a mediator. The mediator uses access control, identity control, and an obligation manager. The obligation manager ensures that event condition actions are performed. In the case some data should no longer belong in the system, the obligation manager ensures that it is removed. By operating as a middleware where the users identity is hidden, makes it possible to create a system where anonymous transactions can exist. This system is an embodiment of PRIME.

Another system for managing privacy is presented by Kobsa and Alfred [27]. The problem domain discussed is the different rules in different jurisdiction where people do online business. The work presented is an architecture they call RAIC. RAIC is a service-oriented unit that contains an array or group of similar or identical components. The different components contain varying degrees of privacy. When a customer buys an item with the RAIC architecture, the system chooses the privacy components, which ensure that the enterprise performs the transaction according to the customers geographically laws and regulations. The architecture is able to operate in countries with both weak and strong privacy laws.

Access control is an important part of any privacy system. Kenny and Korba [26] argue for using standard digital rights management's systems (DRM) concept inside of privacy rights management systems (PRM). PRM takes several actions to enhance privacy.

These include: only logging that information has been accessed not by whom, use content providers that enforces role based access control, and logs the use inside of the PRM. The main advantage according to the article is that the system would satisfy the EU's rules and regulations for data privacy.

Different DRM's can preserve privacy, however Mishra and Dheerendra [29], suggest a privacy architecture for DRM's that will be able to preserve privacy and detect malicious users. To ensure the privacy, the DRM utilize protected content download. The DRM consists of two servers; the content server, and the licence server. The content server provides the content. The licence server makes sure that the person requesting content and the server providing content are legitimate people and servers. When downloading content, the user can chose to use an anonymous IP-address. In addition, the license server knows who is requesting, but not what is requested. This makes the system private. To mitigate thieves, the system uses traitor tracing. This involves the use of a revoking list of untrusted users. If a person or user is on the revoking list, they will not be able to retrieve content through the DRM.

Chapter 3

Research Approach

The project tries to answer the following research questions.

RQ1: What are possible bottlenecks and limitations of IDMegler?
RQ2: What is the minimum amount of traffic a system like IDMegler should be able to handle?
RQ3: How can the architecture be designed to cope with the workload?
RQ4: How can the architecture fulfil the functional requirements of the solution?

This chapter discusses and reasons about different research approaches, to answer the research questions. This project uses the following three methods:

- Design science research (DSR) to design, analyse, and implement the system.
- Surveys find the scope of the problems. The project uses both qualitative and semi qualitative approaches.
- Different metrics validates the implemented system.

The methods chosen for the various research questions is based on the methods ability to prove or disprove the research questions.

The first research question (RQ1) is about the bottlenecks and limitations of IDMegler. To identify the bottlenecks and limitations, there is a need for a system prototype. To create a system it is natural to create an architecture and system design. The method of choice becomes design science. In design science, through iterations, it is possible to improve the architecture and system to categorise possible architecture and system flaws.

What is the least amount of traffic a system like IDMegler should be able to handle? (RQ2). It is important to gather information about enterprises that uses information about people to answer this question. Interviews and surveys are methods that could provide data to calculate an estimate of the traffic load.

How can the architecture be designed to cope with the workload? (RQ3). To answer the research question, design science is a good method. By iterating through different designs and possible implementations, it will be easier to validate the system. Through performance testing, it will be possible to see if the implementation is capable of handling the workload as defined through RQ2.

How can the architecture fulfil the functional requirements of the solution? (RQ4). To answer RQ4, the system requirements needs to be specified, and define different metrics based on the system requirements. This project uses design science and surveys to get an indication to what criteria the suggested architecture should manage.

3.1 Design Science Research

Design science research (DSR) aims to create and design an artefact to later evaluate the implementation and determine if the design holds its criteria (Baskerville [9]). The DSR approach first researches the domain to analyse the problem, then designs the artefact and implements it, lastly DSR analyses the implementation. This project uses the same method.

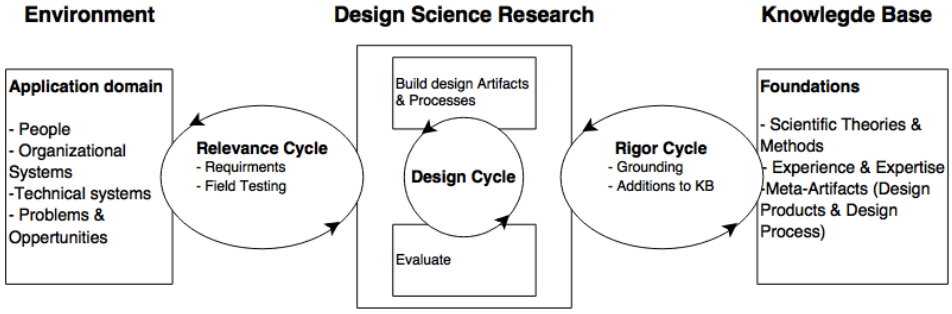

Figure 3.1: Design Science Research Cycles (adapted from [25])

There are three different cycles in design science according to Hevner and Alan [25]. These are the relevance cycle, design cycle, and rigor cycle. The design science uses these cycles to perform and link the different aspects of the practice together. Hevner and Alan argues for the importance of defining the cycles when conducting design science.

The relevance cycle connects the environment with the design science research. As Hevner and Alan state, it is important to define why the research is done. The reason behind the research in this project is to improve the privacy and control over information about people. The focus is on decoupling identity data from transaction data.

The rigor cycle ties the design science research to the knowledge base. This cycle is important to confirm that the design is innovative, and to determine that it is not an im-

plementation of an existing system. This project uses the literature review to search for similar research, and to make sure that the system is innovative.

The design cycle is the foundation of DSR. This is the cycle that iterates between design and improvement. Figure 3.1 is adapted from [25], and it shows the different cycles and how they are related. The project applies this technique when designing the architecture and implementing the prototype.

Peffers et al. [34] introduces the DSRM (Design science research method), and describes six activities.

Activity 1: Problem identification and motivation
Activity 2: Define the objectives for a solution
Activity 3: Design and development
Activity 4: Demonstration
Activity 5: Evaluation
Activity 6. Communication

The activities and the implications they have on this project are discussed in Chapter 7.

Chapter 4 introduces and evaluates the design. First, the chapter shows the requirement specification, then introduces three different architectural designs, and selects the best one for the purpose. Chapter 5 describes the systems implementation as well as reflections on design decisions and problems under deployment and design. Chapter 6 shows the tests, results, and calculations for the threshold values, and Chapter 7 discusses the results.

3.2 Surveys

There are different kinds of surveys (Shull et al. [37]). The main purpose of surveys is to collect information from a targeted set of individuals. This report uses two different kinds of surveys. First, a qualitative survey or interview survey with ”SpareBank 1”. The purposes of the interviews were to estimate the number of transactions with personal information a big financial company in Norway has. ”SpareBank 1” also provided information about response time requirements. In addition, general information was asked about what kind of technology and what ”SpareBank 1” see as the future technology for handling personal information. The second survey was a quantitative questionnaire (with some qualitative questions) with focus on obtaining data about people’s distribution of personal information. The survey asked about people’s number of memberships or customer relationships with different enterprises to identify the distribution of information about people.

The two forms of surveys used in this project according to Shull et al. [37], interviews, and questionnaire, are both part of the inquisitive techniques. This means that they are most efficient for gathering a general understanding of a process or concept.

The interview in this project was semi structured. The participant "SpareBank 1" received questions in advance. The interviews were done over a series of weekly phone meetings where the newest information regarding the provided questions were discussed.

Google schema was used to create the questionnaire. The questionnaire had 13 explicit questions about enterprises, and one open question encouraging the participants to mention all current and previous customer relationships. In addition, the questionnaire had a section asking about general information (sex, age, grade of education, and type of smartphone).

Chapter 6 shows the results of the data collected.

3.3 Metrics

Metrics identify what the system should test. The main metric tested in this project was performance. With performance tests, this project tries to find response times for the transactions. The measured metrics in this project include longest, shorts, and average response times. In addition to the response times, there are also metrics for the number of transactions the system can handle.

This projects main measurement is response time based on transactions per second. The response time is measured from sending a request until the response message is retrieved. (3.1) is the input parameter when testing the system. The program used to test the system then calculates the longest, shortest, and average response time over a given time. t and s in (3.1) stand for transactions and second respectively.

$$t/s \tag{3.1}$$

All enterprises that need to request data from IDMegler are divided into a set of categories. A category represents enterprises with similar transaction patterns towards IDMegler. To estimate the average number of transaction of an enterprise, the load of various types of services are estimated separately and then added to get the total. This project uses the following formulas to find the threshold values for the total transaction load.

$$\frac{\sum_{i=1}^{n} x_i}{daysInMonth \cdot 24 \cdot 60 \cdot 60} \tag{3.2}$$

$$\sum_{i=1}^{m} y_i \cdot z_i \tag{3.3}$$

(3.2) calculates the average transactions per second from the amount of transactions happening over a month for a enterprise belonging to one category. x_i represents the total transactions throughout a month for service type i. n is the number of services.

(3.3) calculates the total transaction load from all categories of enterprises who use information about persons. y_i represents the number of enterprises within i categories, and

z_i represents the transactions per second in the given category. m is the number of categories.

3.4 Types of data

This project uses three data types gathered form three different sources.

- Qualitative data from ”SpareBank 1”
- Partly qualitative data from the survey distributed on Facebook
- Metrics generated through test performed on the proxy system implemented

3.4.1 Qualitative data

”SpareBank 1”
A set of questions where used during the cooperation with ”SpareBank 1”, as well as weekly phone meetings over 4 weeks to discuss what data they could provide.

”SpareBank 1” uses both internal and external channels when handling personal information. During the communication, it became clear that they were not able to provide information about the total number of internal transactions. This means that the data for calculating the total amount of transactions for a big enterprise in Norway will be partly guessing.

Some general questions were asked, during the communication with ”SpareBank 1”. The reason for asking these questions were to get an understanding about certain aspects of their enterprise. The questions included information about the number of insight requests they receive per year and their use of technology, especially about NOSQL databases.

3.4.2 Quantitative Data

The main intention of the survey was to get more information about the use of different services. The questions asked in the questioner were general questions about the number of enterprises a person have or have had customer relationship with. The semi-qualitative data gathered gives an indication about both the total and average number of different enterprises a citizen have relationship with. It also gives some information about the overall control people have over their membership.

It is important to note that the sample of people were relative to the people who distributed the questionnaire. The survey was shared on Facebook by the author of this master thesis and one of the two supervisors. This means that the sample of people answering the survey was people who either are studying themselves or highly educated.

3.4.3 Test Data

Test types
There are many types of tests, often categorised into types, levels, and methods. The methods are black box, and white box testing (Nidhra et al. [33]). The levels range from unit testing to acceptance testing. Lastly, the levels comes in many variations, ranging from installation, regression, to performance tests. White box tests are tests where the tester have complete access to how the component is created under testing. This is not the case in a black box testing scenario. This project uses white box testing as the method for testing. The level of test is a system test, and the type of testing preformed is performance tests (Neely et al. [31]). The reason for using performance tests is to answer the research questions in this project. The focus is on the bottlenecks and flaws in the design as well as the how the architecture can cope with the estimated workload (answer to research question: RQ1, RQ3, and RQ4).

What has been tested?
The main features focused on during the testing of this project, was to identify any bottlenecks or performance issues. In addition, to see if the system meets the availability needed. The availability is difficult to test due to the time frame of the project. However, the tests had a decent test period to indicate whether the system is dependable.

Two different scenarios tests the system. The first step is to test everything locally on one computer to refine the design of the system. This has also been the platform used for implementing the system, and several iterations have made it possible to enhance different aspects (mainly performance). The second step is distributing the service to different servers to simulate distribution. The distributed scenario is done by setting up four remote servers.

Omissions
Security is not tested, due to the complexity it brings to the implementation and the project's limited time frame. Security is very important for the whole system. However, if the system could not perform its tasks to a minimal standard in a fast and dependable manner, there would not be a need to implement the system at all.

3.5 Development of the Prototype System

This project uses an iterative process to develop and implement the prototype system. The system architecture was first designed, and after designing the different components the implementation process started. The system was developed in C# and the first part of the process was to develop suitable components that could satisfy the architecture. After developing the components, the implementation of the slice of the system began. When finishing the slice, the testing process started. This involved tweaking the different classes and solution to increase the performance of the system. The iterative process is displayed in Figure 3.2.

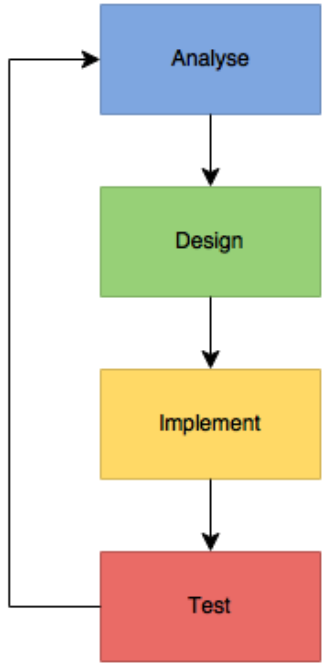

Figure 3.2: Visualisation of the Implementation Process

3.6 Summary

The chapter has shown the different methods used during this project. The project consists of three different contributions. The first is a series of interviews with ”SpareBank 1”. The second part is a questionnaire about the number of enterprises people have or has a customer relationship with. The last part is using design science research to create an architecture and implementation of a prototype version of IDMegler.

Chapter 4

Requirement Specification and Architectural Design

This chapter specifies the requirements of IDMegler, the architectural drivers, and shows the different roles and stakeholders. The chapter presents three different design options of IDMegler. These models are used to evaluate how the optimal distributed model could look like. The best architecture according to the evaluation is chosen for further investigation and the chapter then expands upon that model in details.

4.1 Requirement Specification

The requirement specification [32] is a guideline for expressing a system's attributes and what kinds of features the system needs to implement. The following section presents two sets of requirement specifications. The first specification specifies all features IDMegler would have to contain in a complete implementation. The second specification, specifies the subset of requirements that this project implements. The requirements are specified to be testable.

IDMegler system as a whole

Functional requirements

FR1: IDMegler shall be able to add new enterprises.
FR2: IDMegler shall be able to remove enterprises.
FR3: IDMegler shall be able to validate enterprises.
FR4: IDMegler shall be able to add a new customer to an existing enterprise.
FR5: IDMegler shall be able to remove a customer from an enterprise.
FR6: IDMegler shall be able to retrieve personal information about a customer.

FR7: IDMegler shall be able to accept requests about personal information from valid enterprises.
FR8: IDMegler shall be able to return personal information about one of their customers to an enterprise given a valid cause for needing this info.
FR9: IDMegler shall be able to find the right person in ”Folkeregisteret” based on enterprise and customer ID.
FR10: IDMegler shall be able to reset the IDs on all users of the system (both enterprises and their respective customer IDs).
FR11: IDMegler shall be able to redistribute the keys if they are deleted or otherwise are no longer valid to the enterprise who are entitled to them.
FR12: IDMegler shall be able to display a citizen’s entire history of logged transactions.
FR13: A citizen shall be able to safely log in and out of IDMegler.
FR14: A citizen shall be able to block enterprises who he no longer have a customer relationship with or otherwise are not supposed to get personal information about the customer according to Norwegian law.
FR15: IDMegler shall log every transaction an enterprise makes
FR16: IDMegler shall compress the logs generated to keep the valuable information but not keep excessive amounts of logs.
FR17: The logs shall not contain any sensitive information. Every log shall use numbers to represent the enterprise and customer, as well as numbers for placeholders for transaction types.
FR18: IDMegler shall use role based access control to ensure that only authorized users have access to its services.
FR19: IDMegler shall be able to retrieve and communicate with ”Folkeregisteret”.
FR20: IDMegler shall have an easy to use user interface.
FR21: IDMegler shall be able to communicate with different enterprises and get type codes for their transactions, to better the information of the logs.
FR22: For an enterprise to retrieve data, the enterprise needs to specify what it shall be used for.
FR23: IDMegler shall be able to identify a person based on enterprise and customer ID.

Non-functional requirements

Security

NFR1: The system shall use secure communication (HTTPS, SSL/TLS) when communicating with all of its components.
NFR2: The system shall be able to repel and detect d-dos attacks.
NFR3: The system shall use mechanisms to handle common web security flaws (OWASP).

Performance

NFR4: The system shall be able to handle all requests for personal information towards ”Folkeregisteret” from all businesses in Norway.

NFR5: The response time for an enterprise requesting information of one of its customers shall be less than 160 ms[1].
NFR6: The system shall be able to handle at least 4 500 transactions per second.
NFR7: The system shall be able to redistribute the keys from zero keys to all of them in less than a minute.

Availability

NFR8: The system shall be available 99.99999% of the time.
NFR9: The system needs to be operational when the keys in the system are being reset or updated.
NFR10: The system shall have an overlap where two different sets of keys are valid until every new key is in place to minimize any downtime.

These requirements are derived form a threat analyses of IDMegler [36], and the document specifying the idea [38].

In all of these requirements, the word IDMegler is used as the actor. The reason for not specifying the actors is that at this point it is not known what the roles will be. In some scenarios, the role might only be for a specific limited set of users (like administrators of IDMegler).

Subset of IDMegler (the requirements this project implements)

Functional requirements

FR6: IDMegler shall be able to retrieve personal information about a customer.
FR7: IDMegler shall be able to accept requests about personal information from valid enterprises.
FR9: IDMegler shall be able to find the right person in ”Folkeregisteret” based on enterprise and customer ID.
FR17: The logs shall not contain any sensitive information. Every log shall use numbers to represent the enterprise and customer, as well as numbers for placeholders for transaction types.
FR23: IDMegler shall be able to identify a person based on enterprise and customer id.

Non-functional requirements

NFR5: The response time for an enterprise requesting information of one of its customers shall be less than 160 ms.
NFR6: The system shall be able to handle at least 4 500 transactions per second.

The subset is much smaller than the full requirement specification. The reason for taking these specific requirements compared to the others is that this subset is sufficient to create a functioning system and test performance. As a result, the system implemented will help to answer the research questions in this report.

[1]This is based on the data provided from ”SpareBank 1” which is about 25% of the average time they have today. See chapter 6 for the data calculation

4.2 Architectural Drivers

4.2.1 Functional Requirements

There are many scenarios important to the system. The most important one for this project is: the enterprise requests for customer information using the enterprises own identification and a customer ID. This scenario is the most essential for testing availability, and performance of the system. It is also the function that will be most used. The project bases the quality attributes on this core scenario.

4.2.2 Quality Attributes

There are different parts to consider when developing a system; the internal and external quality attributes. This project defines different quality attributes during the design process, based on the requirements.

External

- Performance
- Availability
- Security

Internal

- Modifiability

Performance is a quality attribute important to IDMegler. In this project, the performance quality attribute about the response time of the solution is vital. The quality attribute includes the average, longest and shortest response time. The non-functional requirement NFR5 ”each response should be less than 160 ms”, is defined based on this quality attribute. Performance also affects the usefulness of the system. In a scenario where the response time for IDMegler increases, the usefulness of the system will decline. If the response time becomes too high, there is a chance that the enterprises will store the information internally and thereby misuse the intention of the application.

NFR8: ”The system should be available 99.99999% of the time,” specifies the availability quality attribute. This system has the potential to become a vital service for many enterprises in Norway. If that is the case, there can be hardly no downtime of the running system. Other systems will be dependent on IDMegler; therefore, availability is an important quality attribute.

Security as a quality attribute is here because of the need for secure communication. In addition, the system will likely be a major point of attack for any attacker who wants to cripple Norway.

Modifiability is only important for further changes or if anyone finds it interesting enough to expand the solution.

4.2.3 Technical Requirements

This project uses C# and ASP.net [2] technology. The ASP.net helps with implementing a fully functional REST API [28]. This has become one of the standards for developing web services. They are easy to set up and perform well. Use of REST API's will make it easy to implement the communication from both the enterprise side of the system and the citizen's side.

When the system retrieves data from "Folkeregisteret", the component should cache the information. This will reduce the number of times it needs to retrieve information about each person form "Folkeregisterete", in the case where the same person is requested several times within a limited time period.

Transactions

- Lookup
- Verify
- Send info
- Log transaction

Every form of transaction will be a lookup, which is the nature of IDMegler. The system will not manipulate any information. Data collection and access management is its only job. This means that the system needs to classify a series of different types of lookups to improve the insight citizens will receive from the report function[2] of IDMegler. There are two ways to categorise the enterprises use of the customer data. One way would be to make a standard set of usage categories every enterprise would have to follow. This will be easy to implement in IDMegler, however it could pose a problem when deciding the set of usage types. Another way would be to communicate with each enterprise and set up individual categories. The downside is the amount of time the functionality will take to maintain. Letting the enterprises define their own categories means that the categories must be updated over time. It will then be important to develop maintainable solutions for that purpose.

Databases
In general, there are two major different types of databases. The relational database management system (RDBMS) and NOSQL databases. The main difference is that in the RDBMS, the database structures all the data into schemas and the same kind of information is stored in the same area, with the same sets of values, whereas in the NOSQL databases, the data can be stored in any way. This could be key-value pairs, documents, or nodes (graph databases). NOSQL databases generally perform better and are highly scalable. One of the main disadvantages in the use of NOSQL is that not all of them implement the ACID (Atomicity, Consistency, Isolation, and Durability) abilities. In addition, there is a greater need for the programmer to keep the data coherent, as there are no schemas to ensure that the same data is uniform. Usually the greater flexibility and the improved performance have been the driving factors in increased use of NOSQL databases.

[2]FR12: IDMegler shall be able to display a citizen's entire history of logged transactions.

This project uses different kinds of databases. The project uses a MySQL database to simulate ”Folkeregisteret”. Locally, for optimization, the project have chosen to use NOSQL databases. Two different kinds of NOSQL databases have been used RavenDB [4] and Redis [5]. RavenDB implements the ACID abilities while Redis does not.

4.3 Roles and Stakeholders

To analyse the impact of IDMegler, it is considered how the system will affect its three main stakeholders: government, enterprises, and citizens. It is also considered how it will affect a major misuser: criminals. The goal should be to make life easier for the three former, but not for the latter. A person may embody more than one type of role. For instance, a person working for the government will also be a citizen. However, in this project a person only embodies one type of role at any given time. This project sees the roles as embodiments of the stakeholders.

Government
This project sees on the national and counties levels of the government when presenting this role. The government usually wants control. Control gives them the option to better safeguard and manage the role as the decision maker of a nation. IDMegler would give increased control in regards to the distribution of citizen's personal information. At the same time, a system like IDMegler poses a nationwide threat. The threat IDMegler poses is the vulnerability the nation would be in if the system crashed. In the end, the government would benefit from an implementation and law regulation of the system. Especially in the role as protector and decision maker of a nation.

Table 4.1: Pros and Cons for the Government Stakeholder

Pros	**Cons**
Better control over what is stored in the system	Must standardize current solutions for the new technology
Better integrity	IDMegler poses a threat that is the vulnerability the nation would be in if the system crashed
New standard for setting up new systems	
Increased trust (confidentiality)	

Enterprise
An Enterprises main purpose is to earn money or work towards a cause they believe in. How they do this is up to them and the different laws and regulations of the jurisdiction they operate in. In this context, there are few advantages (see table 4.2) to a system like IDMegler. The positive sides include the increased trust a citizen will have if they use the system, no need for maintaining the personal information, and increased integrity. The downside is that the enterprise loses potential valuable information that they could have used to increase their profit (Better targeting/ recommender systems).

Table 4.2: Pros and Cons for the Enterprise Stakeholder

Pros	Cons
Increased customer trust	The enterprise will lose some power over current assets when it comes to personal information. User history and direct links between individual costumers and transactions will be lost.
Will not have to worry about the integrity of personal information	The enterprise will need to adapt systems to a new interface. This will vary form company to company depending on how much the new interface will affect the company.
All new systems will have a standard for interacting with personal information	

Citizens

A citizen's role is to participate and contribute to society. In general, they are the stakeholder with the most to gain from IDMegler. Even though they are the most likely stakeholder to gain benefits, some resistance to the introduction of a system like IDMegler will exist. The positive side includes increased control through the insight service and increased awareness of the distribution of personal information. The downside for this stakeholder becomes the need to adapt to a new system, and the scepticism people initially have.

Table 4.3: Pros and Cons for the Citizen Stakeholder

Pros	Cons
Increased control	A new system to adapt to
Increased privacy awareness	Scepticism towards the credibility of the system
One place to manage private data	
Increased integrity	

Criminals

Criminal's main concern is to keep their crimes secret. In addition, they want to exploit systems for own personal gain. They are not a valued role in this project. The positive sides for the criminals include the increased control. By having greater control, it becomes easier to hide. The downside is that identity thefts become harder to perform.

Table 4.4: Pros and Cons for the Criminal Role

Pros	Cons
Increased control to hide from society	Harder to perform identity thefts

Others
In addition to the general roles and stakeholders, the author of this project, the supervisors of this project, and future people continuing developing this system are important stakeholders.

4.4 IDMegler Designs

This section presents three different designs of ”IDMegler” to find the most optimal solution. The section introduces each design superficially and expands on the best one in more detail.

4.4.1 Peer to Peer

The structure can be seen in Figure 4.1.

This structure works like a peer-to-peer network. Each component handles every function IDMegler will have to take care of. This means that at some point during the day each component has to synchronize every update from the other peer components. This will ensure that IDMegler does not lose any information. However, it will delay and make it more complicated to introduce new enterprises and customers into the system. As shown in Figure 4.1 the design enables multiple instances of the peer-to-peer component.

Pros

- Very stable (The design stores the information in multiple locations)
- No need for backup (Backup is done automatically)

Cons

- Many attack surfaces
- Many duplications
- Lots of communication between the different components
- If one is corrupted it could potentially spread
- Adding new enterprises or customer into the system will be complicated

4.4.2 Master/Slave

The overall structure can be seen in Figure 4.2.

The idea behind this structure is to split the tasks of IDMegler into two separate components, Insight Key Data Management component (IKDM (master)) and Log Request Information Management component (LRIM (slave)). The architecture opens for several

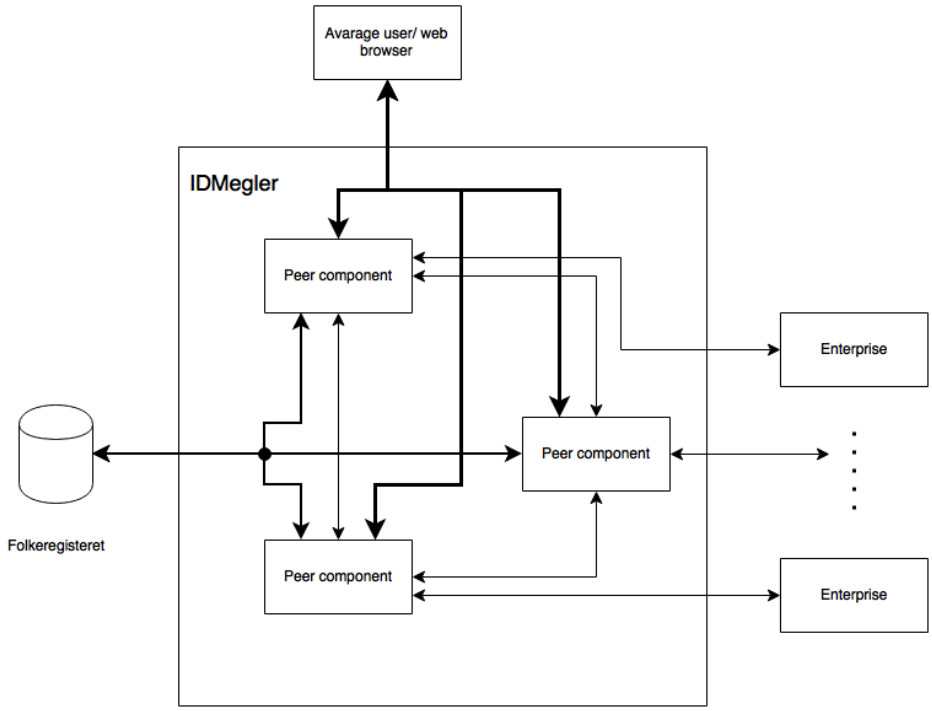

Figure 4.1: Peer-To-Peer structure of IDMegler

instances of the LRIM components as indicated in Figure 4.2. The IKDM component will handle the insight service and contain all the keys and logs. The LRIM component will manage the task of getting information from ”Folkeregisteret” to the enterprise and make the logs. The LRIM component then sends the logs to the IKDM component where the IKDM component stores them. The IKDM component will not have direct contact with ”Folkeregisteret”.

Pros

- Dividing of responsibility
- Easy to expand (redistribute new keys, add new costumers)
- Easy to distribute (New servers for enterprises with the extra need for quick response time)
- Few things to back up (mainly the IKDM component)
- Shortest path possible for accessing all information

Cons

- IKDM component is a possible bottleneck

- Big attack surface

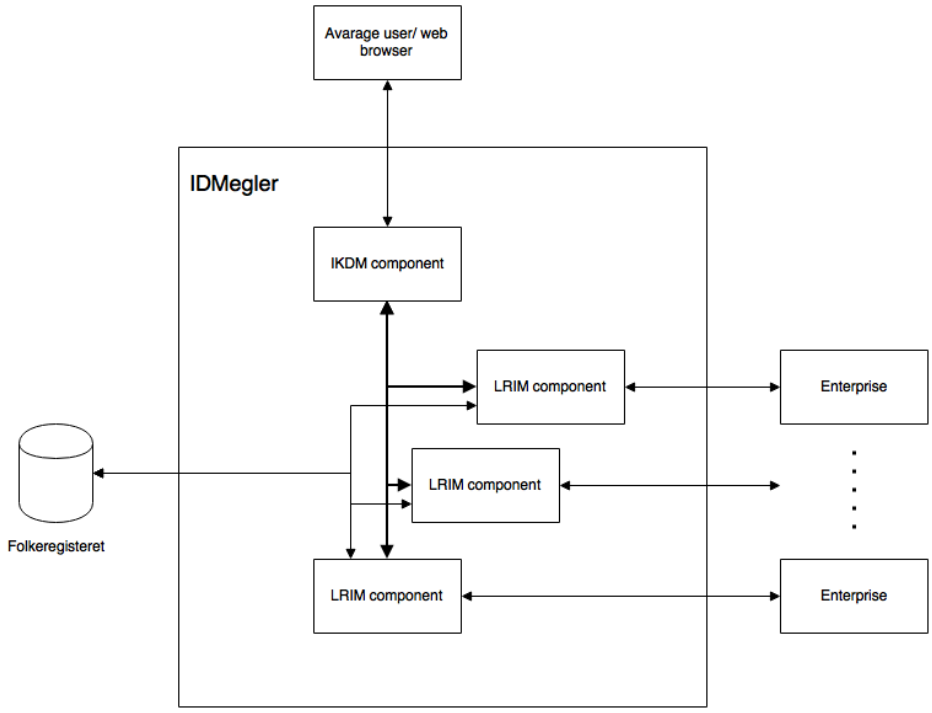

Figure 4.2: Master/Slave Structure of IDMegler

4.4.3 Model-View-Controller

The overall structure can be seen in Figure 4.3.

The idea is to have two fronts or controller components whose only role is to redirect and manage the incoming requests. Each IDMegler component handles all features (logging, finding the right person inside ”Folkeregisteret”, and report the logs to the users). The UserFront takes incoming requests and retrieves all the relevant logged data from each mini-IDMegler component. The BusinessFront handles the incoming requests for the enterprises and determines which mini-IDMegler component is capable of responding to the request. When an enterprise is added, it will be added to one and only one mini-IDMegler. Therefore, no connection exists between the mini-IDMegler's. As shown in Figure 4.3 the design enables multiple instances of the mini-IDMegler component.

Pros

- Few calls between servers
- Easy to expand

- Few connections
- Only two disposed fronts (attack surfaces)

Cons

- Hard to maintain (introduction of new keys, renew keys)
- Lots of duplication (databases, information, keys)
- Only one access point for enterprises (potential bottleneck/slow access)
- Possible long response time for users (UserFront will have to retrieve lots of information from the different components)
- Needs lots of backup to secure everything in the case of a shutdown

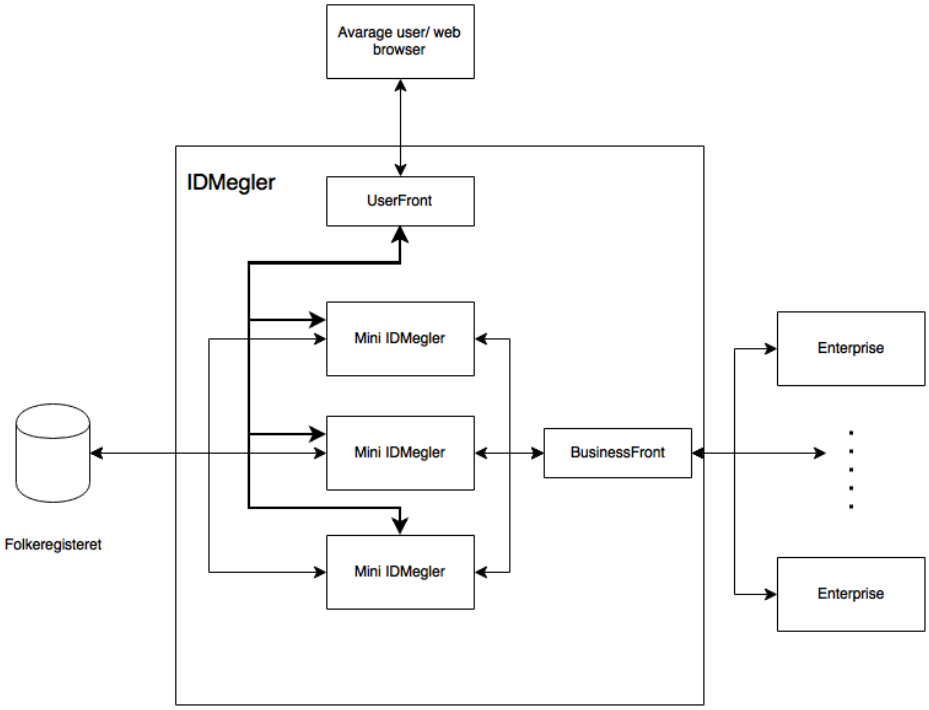

Figure 4.3: Model View Controller Structure of IDMegler

4.5 Chosen Design

After weighting the different designs pros and cons, the Master Slave design was deemed to have the best qualities and attributes. The main features of this design is that it divides the responsibilities and functions into two components. The chose design opens for several

instances of the LRIM component. This will make the system highly scalable. In addition, by using several LRIM components it becomes possible to cache key ID's in the individual components, which will enhance performance. Lastly, the design is the one with the best performance for data requests from the enterprises. If done correctly the main feature of retrieving information for enterprises will be highly scalable. The division will make it easier to focus on core aspects of the core functionalities. This will hopefully make the functions of IDMegler more clear and specific.

4.5.1 IDMegler's Process Flow

In IDMegler, there are two different users of the system. One is the citizens retrieving information. The second one is the enterprise who wants information about a citizen. The two following diagrams show one flow diagram for the citizen and one for the enterprise.

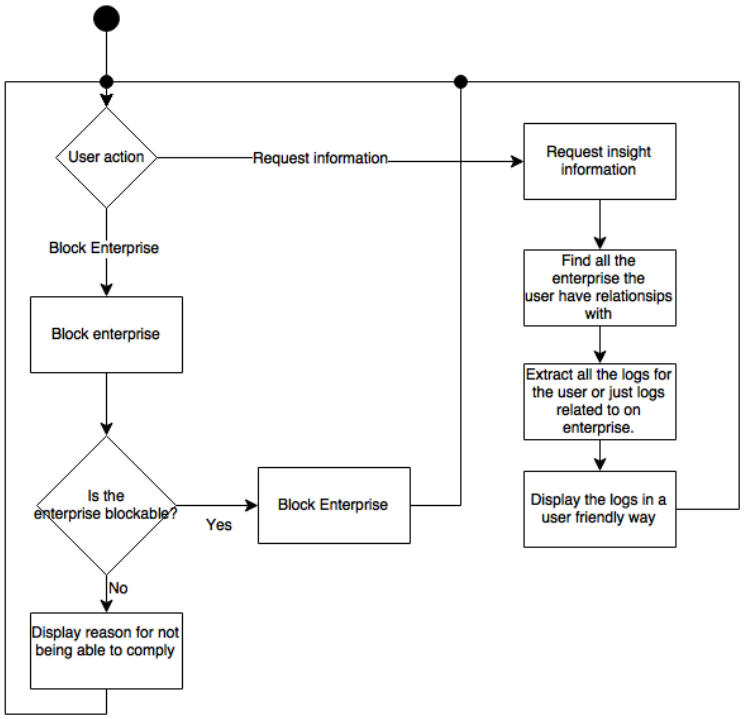

Figure 4.4: Process Flow Diagram for Citizen's Perspective

As seen in Figure 4.4, there are primarily two different actions that a citizen will use IDMegler for. One is to block access to data for enterprises the citizen does not want to have access. The second one is to get an overview of which enterprises have access to personal data, and when they have requested information, and why.

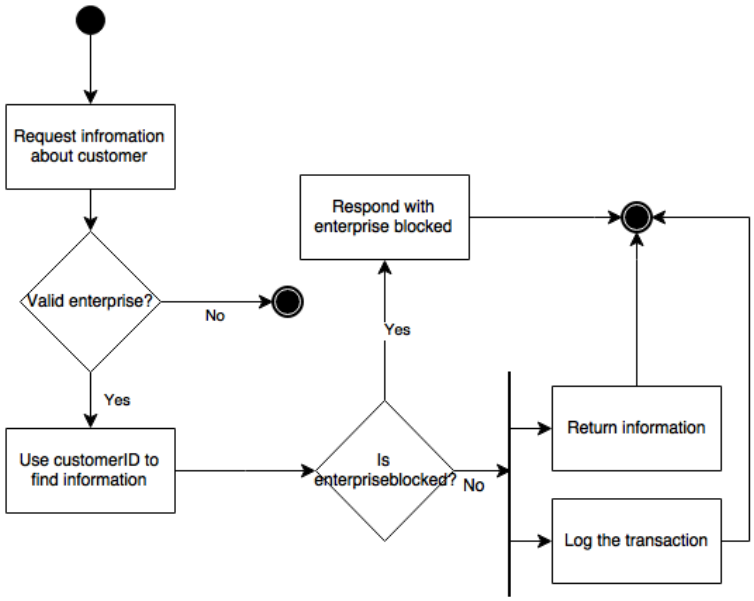

Figure 4.5: Process Flow Diagram for Enterprise Perspective

Figure 4.5, shows the basic process of retrieving information about a customer. As can be seen this process can terminate at different points and will make sure that only authorized enterprises gets access to information about people.

4.5.2 IKDM Component

This component will have to handle a couple of different functions. As the IKDM component, this is where IDMegler will permanently store all information. When IDMegler is running, the IKDM component is responsible for proving the information the LRIM components needs to manage their function in IDMegler. IKDM is also responsible for providing the insight service to the citizens. This includes managing the logged transactions to provide the right citizen with his transaction history. In addition, IKDM also handles the features when a citizen blocks an enterprise for accessing his personal information.

Structure
The IKDM component is composed of mainly two different parts. This is the IDKM server program manageing the dataflow and the database. Figure 4.6 shows the dataflows and the components within IDKM.

Database
The IKDM component will have a database to store all the information it needs to run IDMegler. This database will manage data for enterprises, relationships between enterprises

and citizens, log data, and user information. Using the database as the one hub with information makes it the only component that needs a complete backup. All other components in this system are replaceable and disposable. The following list, lists the different kinds of information that the database will contain.

Information in database

- Enterprise information

  ID

  Data about the enterprise

- Link between enterprise and citizen

  EnterpriseID

  CustomerID within enterprise

  CustomerID in ”Folkeregisteret”

  Information about blocked enterprises

- Logs

  Requests from enterprises

  User logs

  Administrator logs

- User info (for people to log in and out)

Dataflow
The IKDM component will only communicate externally with the LRIM components and the citizens (most likely through a web browser). However, the flow of data will be between different components. Figure 4.6, shows the different external and internal dataflows and how the IKDM component will handle them.

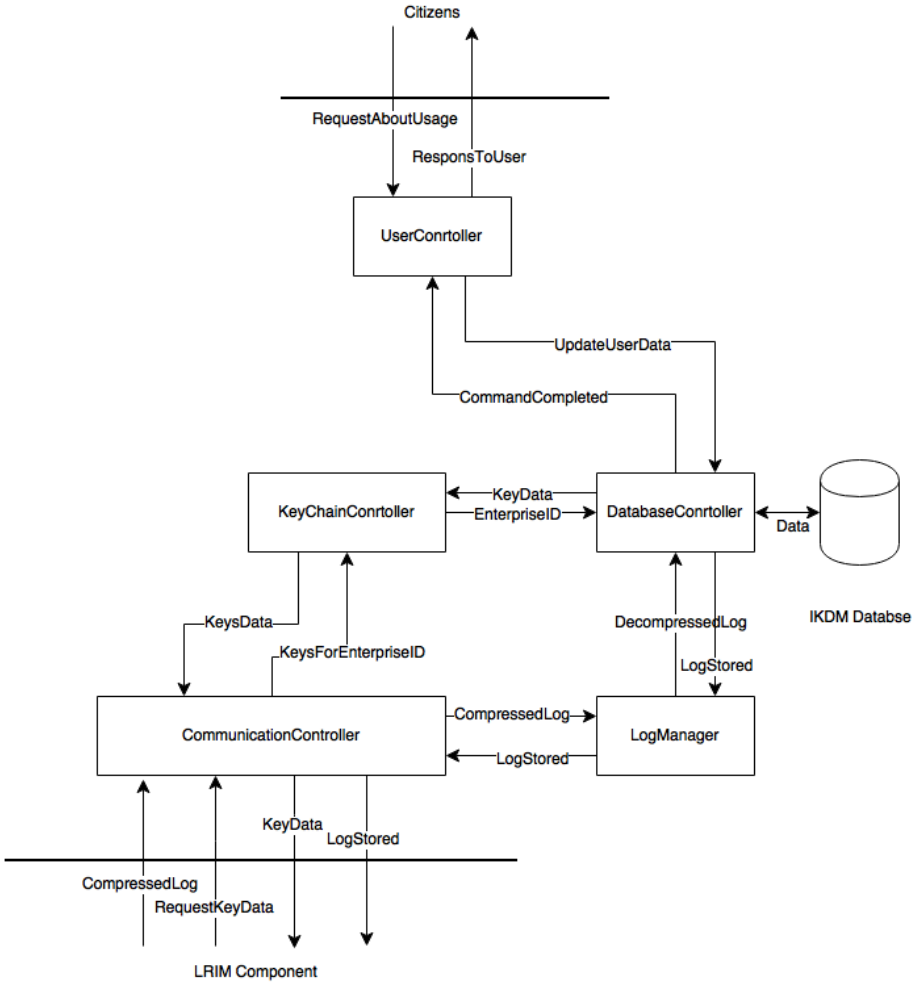

Figure 4.6: Dataflow for the IKDM Component

4.5.3 LRIM Component

This component will have to handle the most crucial part of IDMegler, namely the extraction of information from ”Folkeregistert” to the enterprise, make sure that the logs are made, and prevent unauthorized enterprises access.

Structure
The LRIM components structure is as the IKDM component, divided into two. It contains a database and a server program for handling its functions. Figure 4.7 shows the different internal components. LRIM will have to communicate with ”Folkeregisteret”, IKDM, and

the enterprises. Figure 4.7 shows the dataflow, and how the different internal components uses the data.

This components main feature is to manage the request from the enterprises, and use the keys stored in a local temporary database to find the information that the enterprise requests. The first time an enterprise requests information about a customer, the LRIM component makes contact with the IKDM component and retrieves all the data about that enterprise customer links (keys). This includes the enterprise link to customers, the customer's ID in "Folkeregisteret", and information about customers who have blocked the enterprise. The enterprise is then assigned to this LRIM component until the component is reset. The LRIM component then stores the information retrieved form the IKDM component in a local temporary database. When the LRIM component handles the request from the enterprise, the component creates a log of the transaction and sends it to the IKDM component.

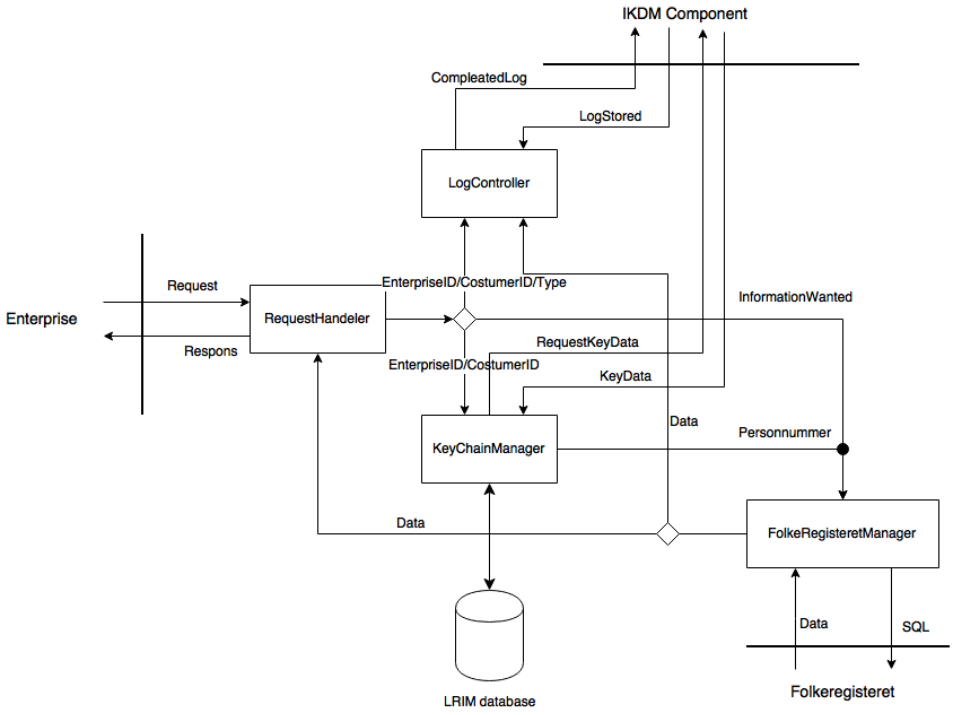

Figure 4.7: Data Flow for the LRIM Component

Database

Information in database (local subset copy from IKDM component database)

- EnterpriseID
- CustomerID within enterprise

- CustomerID in ”Folkeregisteret”
- Information about blocked enterprises

The database in the LRIM component will have links between customerIDs of the enterprise, customerIDs in folkeregisteret, and information if the customer has blocked the information for the enterprise. This enables the LRIM component to retrieve information from ”Folkeregisteret” with its local information.

Sequence Diagram
Figure 4.8 shows a sequence diagram of how the LRIM component handles a request and returns the information to the enterprise (this sequence diagram is only valid after the LRIM component has retrieved information form the IKDM component). The reason for showing this particular sequence diagram is to give a better understanding of how the main functionality in IDMegler with this architecture will work.

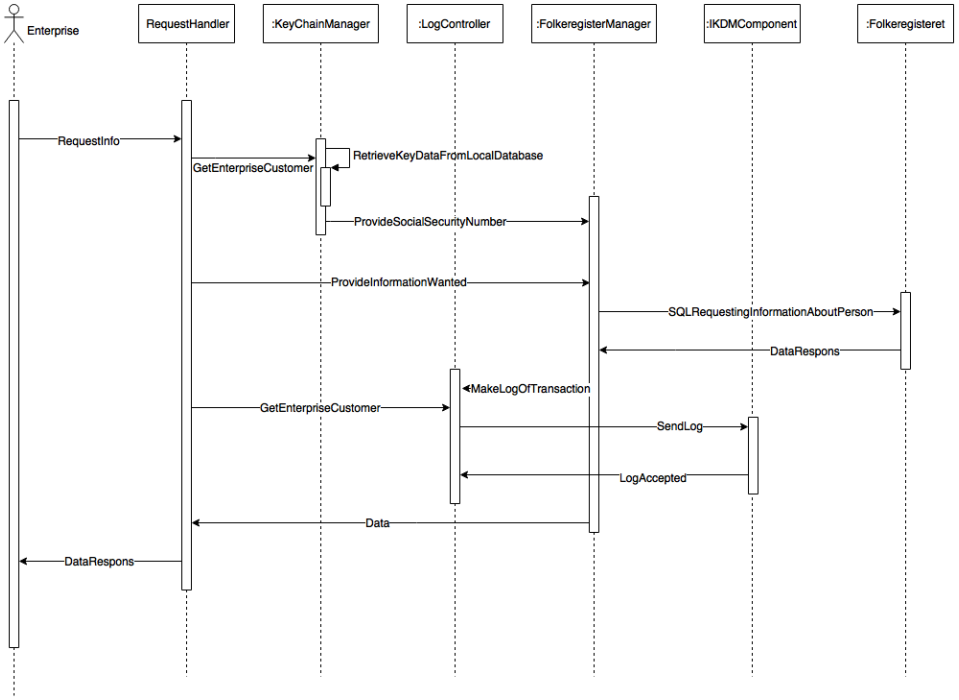

Figure 4.8: Sequence Diagram for the Process of Retrieving a Customer’s Information

4.6 Summary

This section introduced the architecture that this project implements, as well as some of its advantages and disadvantages. The suggested structures main advantage is delegating

its responsibilities. By delegating the responsibilities, the architecture allows the different components to be specialized. This ensures that the system scales easy and is more flexible.

After discussing the different aspects of the architecture, the chapter introduced the components in detail. The project does this by using a structural view, process flow, dataflow, and sequence diagrams. The project shows the objectives the system must accommodate, through the requirement specification.

The architecture is not perfect, there are trade-offs. The figures and diagrams can be misunderstood, and the description of the system have possible improvements. Chapter 5 shows the implementation and design, and includes class diagrams to further increase understandability of the implemented design.

Chapter 5

Design and Implementation

The following sections describe the choices made when implementing the architecture and to ensure a good solution. The overall solution is developed in C# and is available for download at https://github.com/erirei/IdMegler.

5.1 Components/Services

The implementation contains four main components. The components are: The IKDM server, the LRIM server, the enterprise simulator, and the ”Folkeregisteret” simulator. The enterprise simulator and the ”Folkeregisteret” simulator are helper components for simulating the complete system.

5.1.1 IKDM Server

The IKDM server component's core functionality is to:

- Store the keys for the enterprises and their customer.
- Store the logs generated using the system.
- Make logs retrievable for the different citizens.

To retrieve information about the different users (citizens) the system implements a REST API. Early in the implementation, the solution used an SSL-stream to communicate between this component and the LRIM servers to secure the communication. However, due to difficulties deploying the solution to the test servers, the solution dropped the SSL/TLS connection. A document NOSQL database called RavenDB [4] stores the information.

The main criteria considered when implementing the solution where:

- Security
- Performance
- Availability
- Modifiability

The criteria is a direct response to the quality attributes discussed in Chapter 4.2.2.

Class implementation
The IKDM server contains a set of different classes. The project is a made from .NET's default REST API (ASP) which simplifies the REST operations. With the REST aspect there are two classes used for retrieving logs of a specified user. The *UserController* class and the *LogDisplayed* class. Using a controller and a model class is necessary for implementing the REST functionality in ASP.Net. Further, the *DocumentStoreDatabaseController* class handles the communication with the RavenDB and all actions concerning retrieval or updating the document store. By using one class to handle the communication simplifies the development process. The *SSLServerConnection* class handles all incoming messages from the different LRIM servers and determines the right responses. *KeyChainManager* and *LogController* are helper classes used to divide the set of tasks from the other main classes. The division makes sure that individual classes gets different tasks. Individual tasks simplify the code and makes it more manageable. Lastly, there is a set of database (RavenDB) classes used to generate the models for how the document database stores the information.

To better the understanding of the relations between the classes, the class diagram (Figure 5.1) shows how they are related.

UserController
UserController implements the *APIController* class that enables the program to recognize it as a part of the REST API. It only stores one function called *GetAllLogsForUser*. The function takes an integer as input and returns a list with all the logs in the form of *LogDisplayed* class model. The method returns an *IHttpActionResult* that is an asynchronous call. *DocumentStoreDatabaseController*'s public method *GetAllLogsForUserBySocialSecurityNumber* retrieves the information.

DocumentStoreDatabaseController
The API controller needs this class and thus, this class uses the singleton pattern to avoid many instantiations. The API controller class gets instantiated automatically when it returns an *IHttpActionResult*. The singleton pattern avoids creating unnecessary many connections with the document database, which improves performance. On instantiation, the class makes a connection to the document database on localhost:8080. This enables the use of the RavenDB server. By making the connection on instantiation, the class can store the communication. Storing the communication, removes the need for several connections. The class continues by implementing three main methods for storing and retrieving information from the database: *getKeysForEnterprise*, *addNewLog*, and *GetAllLogsForUserBySocialSecurityNumber*. These are the main functionalities that the program needs, to

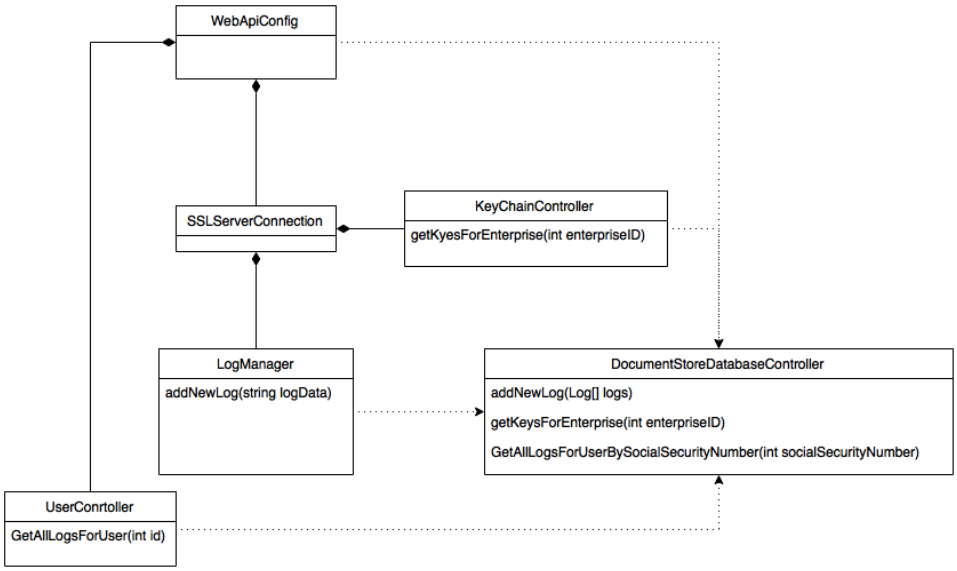

Figure 5.1: Class Diagram for the IKDM Server

operate with the database. Other possible solutions would be to move these functionalities into their own classes to separate functionality. The solution does not do this because of the small set of operations required.

The method *getKeysForEnterprise* takes and integer as input, representing the ID for the enterprise and returns a list with key objects. Each key stores the social security number and a customer ID. The class retrieves the information by getting one document from the database. This reduces the calls to the database, which is the result of using a document-oriented database and enhances performance.

The *AddNewLog* method finds the key-documents in the database where its EnterpriseID equals the incoming logs EnterpriseID and its CustomerID equals the incoming logs CustomerID. Then the method adds the new log to its list of logs, storing the enterpriseID, customerID, date, and type. During the implementation, several redesigns of this method were necessary. In the first implementation, the method passed only one log as the argument, which resulted in to many queries to the database. When tested, the database could not keep up with the requests. In the second attempt, the method takes a list of logs as the argument. By looping over the logs, the number of updates became reduced to a single update. However, this ex-ousted the number of allowed queries at any given time in the database. The final solution solves the performance problem by using the load operation in the database api, which also enforces ACID.

The *GetAllLogsForUserBySocialSecurityNumber* takes an integer as input and returns a list of *LogDisplayed* classes. The method retrieves the user document with the matching social security number, and retrieves all the keys referenced in the user's document. Using the load function in RavenDB, the database only performs one call. This increases the

performance of the system. Alternative methods would be to fetch each key and map it to the user, but would result in a greater workload on the database. Then by iterating through the keys and their list of logs, *LogDisplayed* adds them to the return list. The method then returns the list when the iteration is complete.

This class was rewritten because of the original document database had a limit on documents stored. The first database (SiaqoDB) were a client-side document-store. Its performance was better and a much better fit for the purpose. This is because only one IKDM server is active in the architecture at any given time. This resulted in rewriting all the code in this class, and redesigning the document store to better suited the principle of RavenDB.

SSLServerConnection
SSLServerConnection manges the SSL communication server side, and uses the helper classes *KeyChainManager*, and *LogManager*. The *SslStream* class implements The SSL communication via C#'s own functionality. The alternative would have been to implement the SSL communication myself. During the implementation process, the project did research on several other forms of communication, but none of them included SSL. When deploying the system the servers ran into problems with the SSL authentication. To be able to perform the test the only option was to remove the SSL.

When constructed, the class creates the *KeyChainManager* class and takes *LogManager* as an argument in the constructor. Then proceeds by opening the certificate store and picks the first certificate. Then it instantiates a TCP Listener on port 8800 and makes a thread for handling incoming client sockets. When the class detects a socket, it calls the private method *ProcessClient*. *ProcessClient* opens up a new *SslStream* that accepts any certificates. When a client connects, it authenticates itself as the server. If the authentication succeeds, the class starts to listen to the incoming message. The class does this through the method *readMessage*. The *readMessage* method takes the incoming bytes and through a string-builder generates the message. *StringBuilder* is a fast and efficient way to build strings from a character stream, and enhances performance. The string builder loops over the decrypted chars until the client sends the message '<EOF>'. This stops the loop and removes the '<EOF>' string at the end of the message. The *decipherMessage* method then retrieves the message. All the strings used to communicate uses a specific syntax.

Incoming messages to the server from the client

- ':' parts the action to understand what the client want the server to do
- '#' distinguishes between different instances
- ',' splits values

Incoming messages would have the syntax: {server action}:{data}
{data} =>{instance}#{instance}#{instance}...
{instance} =>{value},{value},{value}...

Example of a string incoming to the *SSLStrem* could be: 'newLog:1,1,1#2,1,3#2,3,2'.

Outgoing messages from the server to the client

- ':' distinguishes between different instances
- '#' distinguishes between different data objects
- ',' splits values
- '—' separates sub data
- '/' distinguishes sub data values

Outgoing message form the server syntax: {Data object}#{Data object} ...
{Data object } = >{instance}:{instance} ...
{instance} = >{value},{value}...
{value} = >{sub data}/{sub value} — {sub data}/{sub value}...

The ':' character splits the message, and the solution uses a switch to decide what action to take. There currently exists two different scenarios. Either the client wants to retrieve all the keys for a given enterprise (indicated by "getKeys" string), or add new logs to the server (indicated by "'newLog' string). Since the message is string based, it becomes natural to design a way to distinguish the different ways to respond to the client's requests. Other syntaxes are possible, but differ little in their behaviour.

Another way to handle the communication could be to send serialized objects. This would be more work, and might be slower in the end because it would send more data. By using the string-based approach, it becomes possible to make compact logs.

If the message is "getKeys", the class calls the method *generateAndGetKeys*. This invokes *keyChainController* to return a string representing the keys corresponding to the enterpriseID. The method *writeMessage* sends the string to the client by using *writeToClient* method. This composes the message into a byte array and encodes it into UTF8. If the incoming message is 'newLog' the *LogManager* calls the method *addNewLog* and passes the data part of the string as the argument. When responding or otherwise operating with data the solution compresses it into a small string. This makes the message between both the client and server small. This reduces the time it will take to communicate, which in turn enhances performance. Currently this is the biggest bottleneck in the solution.

KeyChainController
The *KeyChainController*'s responsibility is to translate the list of Keys generated by the *DocumentStoreDatabaseController*. The method *getKeysForEnterprise* makes this possible. This takes an integer as argument representing the enterpriseID of the enterprise that the method wants to retrieve the keys from. First, the method retrieves the keys through the database controller, then iterates over the keys. Using the syntax for outgoing messages it represents all the different keys as well as which customerID has blocked different enterprises. The string ends with the message '<EOF>', which indicates the information's transfer is complete. Then returns the string generated.

LogController
The *LogController* has one main task, which is to take the string of compressed logs,

transform it into log objects, and then pass it onto the *DocumentStoreDatabaseController*'s *addLog* method. The syntax splits the string into the different data for the objects. First value is type, second is enterpriseID, and the last one is customerID. Then the log itself generate the date of creation. This means that the time of the transaction will not be 100% accurate. This makes the string the class needs to transfer shorter, which reduces the time it will take to send it over the Internet. An alternative could be to parse the string in the *DocumentStoreDatabaseController*, but would in turn increase the amount of tasks the class would be responsible for which is not a good design principle.

RavenDB implementation
This database is a NOSQL database that classifies as a document store. This means that the database stores all information in separate documents. As a result, thinking in the way of RDBMS is not possible. Each document can contain any amount of information. However, there are no requirements of relationships between the documents. This result in each documents design needs to be designed to give answers the majority of queries the program is likely to preform. There exist functionality that enables referencing other documents and this implementation uses it when necessary.

The reason for choosing the document database approach and not NOSQL is that it in many scenarios a NOSQL database will reduces the amount of time it takes to retrieve and store information. In addition, the database needs a lot fewer transactions between itself and the server. Performance in NOSQL will often outshine an RDBMS's performance. In addition, the RavenDB implements the ACID abilities. NOSQL databases usually does not implement ACID. However, this is important because it enhances the durability.

The database in this scenario stores three different document types. Enterprise, User, and key. The user document has the properties: list of key references and social security number. The enterprise document has the properties: name, ID, and list of key objects. The Key document has the properties: ID, CustomerID, EnterpriseID, List of logs, social security number, and list of blocked transaction types. (Se Figure 5.2). It is important to note that the enterprise document does not contain the logs. When the program is running, it uses the key object in the enterprise document to find the reference to the key document with the right customerID. Therefore, the logs are stored in the key document.

5.1.2 LRIM Server

The LRIM server component's core functionality is to:

- Provide enterprises access to information
- Log the enterprises and the type of transactions they perform
- Map keys for enterprises
- Communicate with the IKDM server and send over the logs
- Communicate with 'Folkeregister' simulator to retrieve the actual wanted information

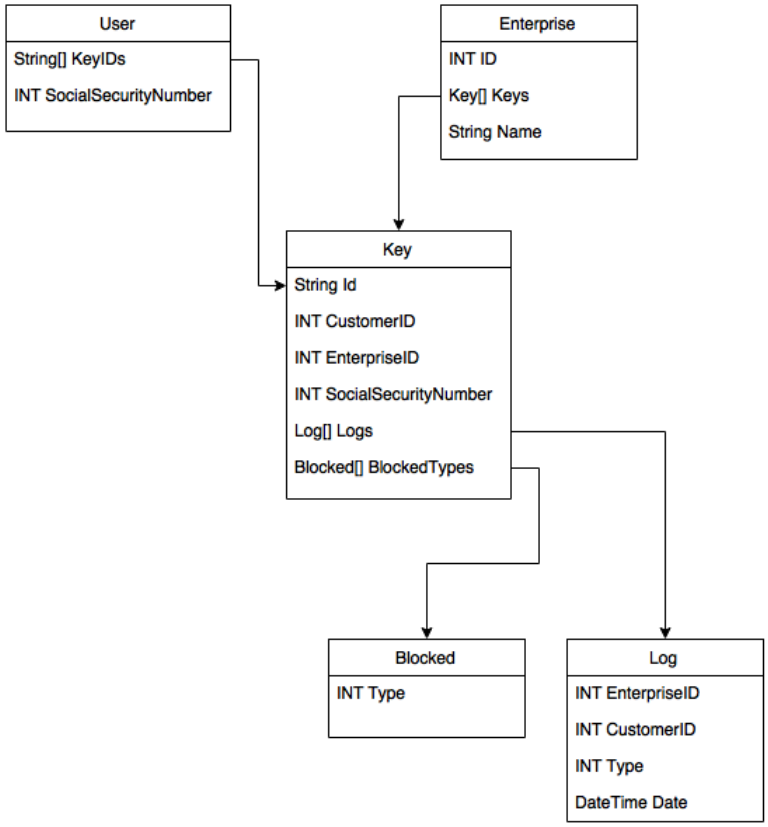

Figure 5.2: Document Database for the RaveDB

The LRIM server uses .Net's REST API (ASP [2]) implementation to handle requests form the enterprises. This is a quick way of handling requests from enterprises in an asynchronous way, which enhances performance. For mapping the different keys to the right enterprise, the server gets all the keys from the IKDM server by using the SSL communication. Then a NOSQL database called Redis stores the keys. This is a fast database. However, it does not support ACID operations. Since the data in this database is not supposed to be other than a lookup for the keys, it does not need these attributes. The SSL makes sure that the information transfers between the servers is secure. The LRIM server uses the same connection to transfer the logs when the enterprise uses the REST API. When communicating with 'Folketregsteret' it connects to the database using MySql's C# api for queering towards a MySql database.

The main criteria considered when implementing the solution were:

- Security
- Performance
- Durability (dependability)
- Modifiability

The criteria is a direct response to the quality attributes discussed in Chapter 4.2.2

Class implementation
The LRIM server contains four classes to handle retrieving information for the enterprises and generating the logs (*RequestHandler*, *KeyChainManager*, *FolkeregisteretManager*, and *LogController*), a class for retrieving data from a MySql database, a *SslClientConnection* class for communicating with the IKDM server, and a controller and model class to implement the REST API. The *RequestHandler* manages what to do when a request arrives. *LogController* generates and sends the logs. *KeyChainManager* communicates with the Redis database and finds the keys to use. *FolkregisterManager* uses the database class to retrieve information from the "Folkeregister" simulator. In addition, the solution contains data classes to store information (*RequestObj*, *DBData*, and *Customer*).

To get a better understanding of the relations between the classes, the class diagram (Figure 5.3) shows how they are related.

DBManager
The DBManager is an implementation of regular MySql database operations. This class uses the library provided by MySql to connect and query towards the database. The class does this by taking the connection string as argument in the constructor of the class. By storing the string, the class does not have to receive this information again. Other solutions would be to pass it as the argument when performing a database operation. This would be a good idea in a scenario where the solution would have to connect to different databases.

To perform all the queries to the "Folkeregister", the class contains the method *preformPrepeardStatementToFindPersonWithData*. This takes arguments in the form of a statement, a social security number, and a string representing what data to retrieve. Then by

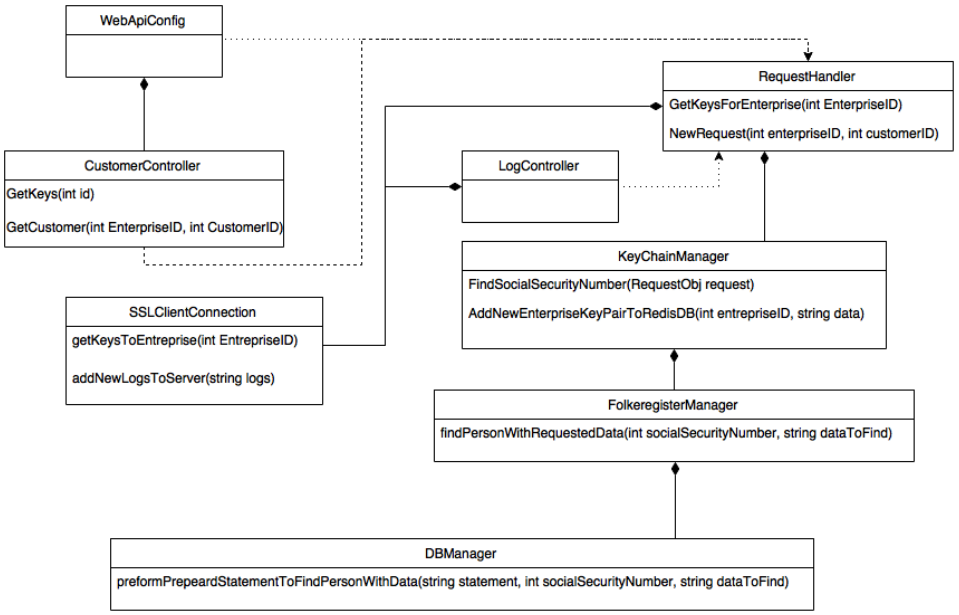

Figure 5.3: Class Diagram for the LRIM Server

using the *using* statement, to be sure to release the connection after use, the method opens a connection and makes a prepared statement query. During early implementation, the method returned the connection, this resulted in the connection not releasing as it should and clogged up the system. This meant that the mapping of the information would have to take place in this class. To accommodate any kind of generic response the solution implements a DBData class to store any kind of information. This later translates the data into its right type. The method does this through a loop over the response data form the queries, which adds the data to two different lists. One storing the type and the other the information.

SSLClientConnection
SSLClientConnection uses the .NET *sslStream* and *tcpClient* classes to communicate with the IKDM server. On creation, it opens the certificate store and stores the certificate to send as authentication when connecting to the IKDM server. This class implements two main functions for communicating with the IKDM server: *getKeysToEnterprise* and *addNewLogsToServer*.

getKeysToEnterprise takes an integer representing the enterpriseID as input. It works by first using a *tcpClient* and connecting with the server, then it opens a SSL stream and by using the helper method *ConnectToServer* authenticates as client. It further proceeds to encode the message sent to the IKDM server. The method does this to be able to retrieve a message with the responding keys and find out if the citizen blocks the retrieval of information. The method then tries to read any message from the server and passes it into the method *readMessage*. This helper method is the same as the one implemented

in the IKDM server. When the data has arrived, the method invokes the static method *AddNewEnterpriseKeyPairToRedisDB* in *KeyChainManager*. The reason for using a static method is to reduce the coupling between the classes.

addNewLogsToServer takes a string as argument, with the string representation of the logs, and starts the method by generating a new *TcpClient* to send the server, for then to authenticate the process as a *sslStream*. Then it writes the logs to the IKDM server.

The *SSLClientConnection* class uses the same principle as the SSL class in the IKDM server. The SSL was also removed here because of during deployment the server were not able to use the certificates generated.

FolkeregisterManager
FolkeregisterManager handles all transactions and messages between the ”Folkeregiser” simulator. The class does this by instantiating a *DBManager* class with the right connection string in this class's constructor. This class main function is to create a customer object representing the data an enterprise wants to retrieve. The class manages this by calling the public method *findPersonWithRequestedData*. This method takes an integer as the social security number for the citizen wanted, and a string with the data requested. This method uses the private method *findDataInDatabase* that passes the same arguments and uses the *DBManager* instance to call the method *preformPrepeardStatementToFindPersonWithData*. A method called *createCustomerObject* gets the return data from *preformPrepeardStatementToFindPersonWithData*. Afterwards it iterates over the information and by using the description of the fields from the data object generated from the *DBManager*, the fields in the customer object appends the right data to the right variables. The method then returns a customer object with the right information.

KeyChainManager
KeyChainManager manages the lookup for matching the enterpriseID with their respective customerID in the Redis[5] database. This class stores the *FolkeregisterManager* class. The class stores it to be able to retrieve the customer object from the Redis database using the social security number. The constructor of this class takes the class as an argument. This class contains two main methods both about the management of the keys in the solution. The first is *FindSocialSecurityNumber* and the second is *AddNewEnterpriseKeyPairToRedisDB*.

FindSocialSecurityNumber takes a request object as input. Then by using the Redis defined way of connecting to a Redis database, it uses the *HashGet* function built into the Redis library, to retrieve a specific hashed value, by using the field as the representation of the customer id and the hash name is representing the enterpriseID. This returns the social security number stored in the specific location. If it manages to retrieve the data, the *FolkeregisterManager*'s method *findPersonWithRequestedData* gets the customer object. During implementation of this method, there were problems with forgetting to close the connections to the Redis database. The solution was to use Redis's standard way of operating with connections. By using a static lazy connection to the database.

AddNewEnterpriseKeyPairToRedisDB is a static method used for adding all keys for an enterprise into the database. The method takes the enterpriseID and a data string repre-

senting the keys and information as arguments. By using the syntax for messaging with the IKDM server, it separates the different objects by splitting the string at ':'. Then it deletes the hash in the database if it exists. The method does this to make sure that there are no lingering keys in the database. Because the deletion and writing is super-fast this causes little to no delay. In addition, a method will only run when updating current information or during a reset. Then the method continues by iterating over each string representing a key, and adds them to the Redis database.

LogController
LogController handles sending the logs to the IKDM server when the LRIM server gets a request. This class takes both an *SslClientConnection* and *RequestHandler* class as arguments in the constructor. The class saves the *SslClientConnection* for later use, and uses the *RequestHandler* class for listening for *newRequest* events fired for that class. By using events, the class handles the logging part of the server. This ensures that it only operates when necessary and enables the use of threads to increase performance. This class only uses the private method *sendNewLogToMainServer* that the *newRequest* event invokes.

sendNewLogToMainServer started in the beginning to send the arguments from the event and pass them into the SSL connection through *addNewLogsToServer*. However, this resulted in the exhausting of the number of sockets that the server could have at any given time under a few request per second (as low as 50 transactions per second). To fix this problem the solution now checks the number of newly added logs. If the number is less than 500 it adds the new log to a list. If the number exceeds 500, it makes a new copy of the list of logs. Using the syntax for messaging the method compresses the logs into a single string by looping over the log list. The method then sends the list to the server by invoking *SslClientConnection*'s method *addNewLogsToServer*. The sending is done in a separate thread to enhance performance.

CustomerController
CustomerController implements the *ApiController* that makes it usable as a REST api class in the .NET module. It stores two methods *GetKeys* and *GetCustomer*. The first takes an integer as the argument representing the enterprise's ID. Then uses the *RequestHandlers* method *getKeysForEnterprise* to get the keys for an enterpriseID with the corresponding ID as requested. The second method *GetCustomer* takes both an enterpriseID and a customerID both represented as an integer as arguments than passes them onto the *RequestHandlers* method *NewRequest*, which returns a customer object. The method returns the object if it is not null. Both methods in this class returns *IHttpActionResult* that are asynchronous methods. In addition, since the class implements the *ApiController* every time a request occurs the solution generates a new instance of this class. This enhances performance by running each request in their own thread.

RequestHandler
RequestHandler main purpose is to handle any incoming requests and use the other classes to invoke the right response to the requests. Since the API calls triggers this class's method, it uses the singleton pattern to avoid initializing at every request. This ensures that there is only one set of the other classes and that they only initializes them when the server

starts. This enhances performance. This class uses the *KeyChainManager* class and the *SSLClientConnection* class, which both are passed as arguments in the constructor. This class also implements an event called *NewRequest*. This event takes a self-defined event argument that holds a request object.

RequestHandler has two public methods, *getKeysForEnterprise* and *NewRequest*. The first takes an integer as argument representing the enterpriseID and then calls the *getKeysToEnterprise* in the *SSLClientConnection* class. *NewRequest* takes an integer for the enterpriseID and customerID as arguments. Then based on the input, makes a request object. This object is than passed as argument to the *KeyChainManager* class's method *FindSocialSecurityNumber* that returns a customer object. The method then stores the object. If the customer object found is not null the method invokes the event *NewRequest* and then passes the request object as the event argument, and at the end returns the customer object.

Redis
This database is a NOSQL key-value store database. The database is popular and have a fast reading and writing speed. The database has high performance and makes it easy to store the keys. Redis involves some different structures in which to store the data, this solution uses the hash set. It makes it possible to store the keys in each hash by using the enterprise ID as the hash value and the customer id as the field, then the social security number as the value. There should be no problems with the individual fields in the database since it supports fields up to 2^{32} field value pairs. This is also the case for the number of hashes.

5.1.3 "Folkeregisteret" Simulator

The project uses this component for placing the data about the citizens in another physical location than where the servers are running. The component archives this by making a database (RDBMS) MySql server with a table containing the fields that are relevant for the citizens to have stored in the register.

The reason for making it as simple as having a database is that this is not the focus of the project. So by distributing the data it will simulate some of the time it will take to get the information from an external source. The downside is that this is not the real representation of what the system will have to communicate with. Thus, an increased delay is possible. However, for the sake of not wasting time, and simulate some of the time it will take the database to retrieve a person's information, this solution was an acceptable trade-off. In addition, since there is no way of knowing how a future solution might look, this simple approach will be enough.

5.1.4 Enterprise Simulator

This is a simple program made for testing purposes. The data that the program monitors is: the amount of requests sent, if the request where successful or not, the time it takes

for a response, the time it takes to complete the test, as well as the longest, shortest, and average response time.

The focus when making the simulator is to have as precise and robust measures as possible. In addition, the focus was to be able to simulate several enterprises at any given time.

Class implementation
This solution uses the standard application project provided by C# in visual studio. By implementing a simple GUI for setting the number of requests per second and the number of enterprises, it enables the simulation of several enterprises sending requests scenario. The classes implemented are a simple XAML/XAML.cs class for the GUI aspect called *MainWindow* and the classes, *EnterpriseSimulator*, and *Enterprise* as well as an *EventStatsArgs* class. The thought behind the set-up is to use the *EnterpriseSimulator* class to instantiate up to several *Enterprises* and each *Enterprise* runs its own thread and performs a set amount of requests per second, each running asynchronous. When a request gets its response, the solution invokes an event with the data of the request. The *EnterpriseSimulator* class stores the data.

An alternative way to make the simulator would be to not use the concept of simulating several enterprises in the same program and instead use several instances of the same program. Another approach to the request response scenario would be to not do them asynchronous but wait for the response before instantiate a new one. This would result in the data reflecting the largest amount of requests/ response's possible. In addition, it would hinder the system from crashing, but not able to specify a specific number for requests per second.

The downside of the current solution is that the implementation of the response time is thread-based so there is room for some errors if the system uses all of the CPU.

MainWindow
This class is the standard class used to make a GUI window (Figure 5.4) appear when running the program. The C# WPF classes makes up the design of the GUI elements. This class sets the data context for the xaml file to a new *EnterpriseSimulator* instance and binds click-events for the start and stop buttons. The fields are: transactions per second, and number of businesses.

EnterpriseSimulator
This class main function is to store the data. The class displays the data in the GUI, starts and stops the simulation, and instantiate the enterprises when the simulation starts. The class uses several property fields to display the data, and uses the method *StartSimulation* to start the simulation. The method *StartSimulation* uses the set frequency taken from the GUI and calculates the amount of milliseconds the enterprises have to wait between each request. Then using the number of *Enterprise*s, it instantiates that number of enterprises, subscribes to the *Enterprise* events *ResponseOk* and *ResponseNotFound*, and adds the *Enterprise* to a list of *Enterprises* which the simulator uses under the simulation. *ResponseOk* and *ResponseNotFound* are invoked when the events occurred and then update the variables stored in the *businessSimulator* class. To stop the simulation, the program

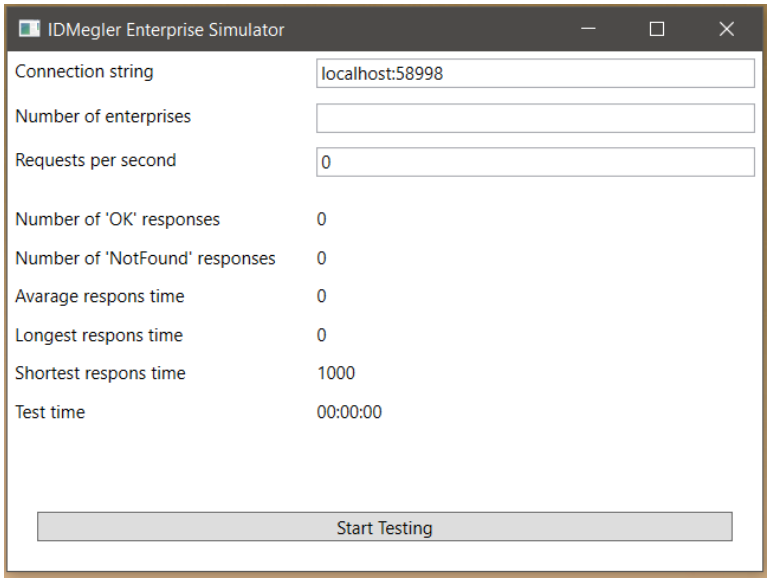

Figure 5.4: Enterprise Simulator GUI Interface

uses the method *StopSimulation*. This method iterates over the list of *Enterprise*s and sets the boolean controlling the loops for sending requests in the *Enterprise* objects. Then it clears the *Enterprise* list.

Enterprise
This class performs the requests to the LRIM server. The class dose this by starting a thread in the constructor that generates http requests to the server by a given frequency. In the constructor the frequency, which enterpriseID to use, and the connection string to use when sending requests are set. Then the boolean for looping over request is set to true and the constructor creates and starts the new thread looping the requests. The thread uses the method *loopRequestCustomer* that creates a new task of type *RequestCustomer* and then sleeps for the time calculated from the frequency. The task *RequestCustomer* is a method that stars by creating a stopwatch object and starts it, then by using the *HttpClient* class in the C# library makes a new client, and then it generates a random number for finding a customer. The client then uses the asynchronous method *getAsynch* that takes a url and makes a request, for then to wait for the response. The thread stops the stopwatch when the response arrives. Depending on the return message, the thread invokes a new event with the data of the response, either *ResponseOK* or *ResponseNotFound*.

5.2 Patterns

During the implementation, the solution uses different patterns to ensure certain attributes.

- Singleton
- Lazy initialization
- Mediator
- iterator
- Master/Slave

The singleton pattern ensures that there is only one instance of a certain object at any one time. This makes it possible to call the methods in the object without having to couple the instance.

The lazy initialization is a pattern used to save memory. This pattern makes sure that the object or resource is not instantiated before the system needs it. The pattern is used by the *KeyChainManager* class to find information in the Redis database.

The mediator pattern makes sure that the behaviour of different components do not differ. The patter is useful to make sure that there is a definition on how a class should work. The pattern is used automatically when creating interfaces.

The iterator pattern is for going through a collection of objects and manipulate them in one session. It is a performance-enhancing pattern. This pattern is used under every time it is necessary to traverse a list to enhance performance.

Master/Slave pattern is for performance. The master uses the slave components to perform actions on its behalf to solve a problem. This implementation uses the pattern in the form of the distribution of the IKDM(Master) and LRIM(Slave) components.

5.3 Shortcomings of the Implementation

In general, the solution presented is not complete. There are different aspects that the solution does no implement, like the security aspect. This is a choice made to reduce the task of the implementation, but can affect the testing. Testing security would be interesting to look into as future work. The solution does not implement the authentication for each user of the insight service; the solution also lack full support of blocking certain enterprises for access. The project has started some of the work to make it possible, but the LRIM server needs to implement it.

IKDM server
The IKDM server's implementation contain some noteworthy shortcomings. The implementation removed the SSL/TLS security due to problems with the deployment of the system to the windows servers. This means that the solution does not have any form of security. This means that there is a possibility when testing the system that the performance it will show is better than what it will be in a fully implemented system. Another defect is that the socket listing on the logs from the LRIM component does not restart when overloaded. The last thing to note with the implementation of the IKDM server is the performance and operation of the RavenDB. The current solution works; however, it

feels less than satisfactory. The database uses lots of memory and processing power, and is not as quick as the author was led to believe.

LRIM server
The LRIM server's implementation is not complete. The server provides features to make it possible to block a enterprise, but the solution does not implement them. In addition, the server currently only makes the same kinds of transaction types when logging a transaction. The server does not handle many connections at the same time and crashes if it gets overloaded. The system should implement a pool to handle workload from different enterprises. The Redis database works well and is a good way to store the keys. The communication between the LRIM server and the IKDM server uses TCP. TCP might not be the best solution.

"Folkeregister" simulator
The "Folkeregister" simulator is not a main concern for the project, because it is not part of the internal design of IDMegler. This means that it is implemented as simple as possible. It is a single table in a MySQL database. The database is not the real thing a system like IDMegler would have to communicate with, but it is an ok representation. The downside is that it is possibly faster than the real "Folkeregister" would be.

Enterprise Simulator
This program is a simple simulator and is designed to function as a simple test tool. There are some uncertainties in the implementation. To check the response time and other data for the request it uses the stopwatch. Because each request runs on an individual thread there are potential for error in the data collected. Another problem with the simulation, to get it to work when testing over a long period of time, the simulator uses only one connection per enterprise. This means that under testing the simulator only have to make the connection one time and keep it. This means that the overall time to complete a transaction might be lower then if it made the connection each time.

5.4 Set up

The tests system contains four 2012 windows servers. Each server represents one of the parts of "IDMegler".

'Folkeregister' simulator
Facts
This part consists of a MySql database.

Issues
After installation, the connection from the local MySql workbench were unable to connect to the remote database.

Solution
The problem was that the WMI rights on the server were not set correctly, after updating the WMI rights the local computer were able to connect to the MySql database.

IKDM server
Facts
This server needed to support IIS 4.5 or higher. In addition, the server needed valid certificates located in the local machine, and the IKDM server needed to be deployed in its entirety (database included)

Issues
The remote deployment feature in visual studio did not work. The server did not have any certificates installed that could be used. When creating a certificate, the solution were not able to use the test certificate. The database's deploy feature in the implementation did not work (the database should have a feature to create a dump-file with the content of the database).

Solution
To deploy the program the server program was deployed locally, and transported as a package to the server. Then using the IIS Manger and web deploy, the server was imported. To be able to use the RavenDB, the solution became to download a fresh database and implement a test function in the server that reapplied the test data in the database. To create test certificates "certmaker.exe" was downloaded, and used to create test-certificates and place them in the local machine.

LRIM server
Facts
This server should install the solution created for the LRIM component and run the Redis database.

Learning from the mistakes from the main server, IIS, web deploy, and windowsSDK where all installed from the beginning. There were no other problems when deploying to this server when the decision for removing the SSL/TLS was made.

Enterprise simulator
Facts
The enterprise simulator only needed to run a simple .exe file. This posed no issues

Troubleshooting
Major issues
Under deployment and creation of the certificate, the servers did not accept them in the SSL/TLS connection. The error produced were: "The credentials supplied to the package were not recognized".

solution
A lot of effort were made into finding a solution for the certificates not working. After several tries (including making a certificate authority to sign the certificate and adding the private key to the certificate), the decision was made to remove the SSL/TLS from the solution to avoid using the certificates, because of the time limit of the project.

5.5 Summary

This chapter have presented the implementation of the architecture, and the choices taken during the implementation of the system. The chapter went into detail about the different parts of the implementation and explained how they work. The chapter does this by going into detail about the definition of the different classes. At the end, the chapter introduced the different patterns the implementation use, and presented a reflection about the shortcomings of the different components of the implementation.

Chapter 6

Results

This chapter presents the raw results of data gathered and generated throughout this project. It starts by showing the data obtained from "SpareBank 1". The chapter proceeds to show the data from the survey, and lastly, the data generated from the tests on the implemented prototype of IDMegler.

6.1 Qualitative Data from "SpareBank 1"

The following sections shows the data gathered from "SpareBank 1".

6.1.1 Transactions

The Figure 6.1 shows the total amount of transactions where information about people were used, from "SpareBank 1"s external channels during January 2016. The data provided is taken from "SpareBank 1 SR" which makes up approximately 25% of "SpareBank 1". This implies that all data must be scaled up by a factor of four.

From the Figure 6.1, the total amount of transactions occurring during a month is $2987631+$ $2987631 + 2569320 =$ **8 554 582**. This means that the total workload for "SpareBank 1" is approximately: $8.55million \cdot 4 = 34.5$ million. Calculated down to transactions per second, the number becomes **13** (rounded up).

This is only for the external side. To estimate the amount for the internal channels this project assumes 25% more traffic internally than externally. This assumption is based on best guess and gut feelings. By using the number calculated and the factor of 1.25, the total amount of transactions per seconds becomes: $13 + 13 \cdot 1.25 =$ **30** (rounded up).

ÅrKvartalMnd	Jan-16
Radetiketter	**Antall Innlogging**
Bedrift Mobil	11 455
Bedrift Portal	5 383
Portal (PM/BM ukjent)	962 788
Privat Mobil	2 000 734
Privat Portal	7 271
Totalsum	**2 987 631**

ÅrKvartalMnd	Jan-16
Radetiketter	**Antall Innlogging**
BankID	2 680
BankID20	907 134
BankIDAPP	38
BankIDMobil	65 590
Innlogget	2 012 189
Totalsum	**2 987 631**

ÅrKvartalMnd	Jan-16
Radetiketter	**Antall Viljes Ytringer**
pm651 - Oversiktsside PersonForsikring i nettbanken.	27 082
pm700 - Min Oversikt - Enhver REST-ressurs hvor bruker ber om data knyttet til et oppgitt sett kontonumre	2 541 576
pm826 - Opprette rolle/fullmakt	298
pm827 - Slette rolle/fullmakt	90
pm846 - Til steg 1 - Kontaktinformasjon	274
Totalsum	**2 569 320**

Figure 6.1: Number of Transactions for ”SpareBank 1 SR”s External Channels

6.1.2 Performance

”SpareBank 1” were able to provide data taken from a performance test for testing one of their systems.

Table 6.1: Data from ”SpareBank 1” Test Data and Response Time.

Call description	**Number of calls (during 72 hours)**	**Average (ms)**	**90% of the calls respond within (ms)**
01/01 readCustomer 1,0 - Default information returned	1202509	1620	3005
01/02 readCustomer 2,0 - By public id	1202438	1808	4092
01/03 readCustomer - Only customer information	60116	789	1439
01/04 readCustomer - Only postal addresses	60113	803	1472
01/05 readCustomer - Only other addresses	60107	675	1275

As seen from the Table 6.1, the average response time is always over 675 ms. This will be the number used when calculating the threshold values for IDMegler’s response time. Because the data provided by ”SpareBank 1” represents the whole stack trace, IDMegler’s treshold value must be a fraction of the number from ”SpareBank 1”.

These performance tests numbers where generated through a test based on experience with what ”SpareBank 1” normally have to handle. Which is 500 000 transactions each day and peaks at 800 000. The test ”SpareBank 1” did were done with this in mind, and with 800 000 transactions each day over 72 hours, calculated down, the number becomes 600 per minute or 10 per second.

6.1.3 General Questions

During the communication with ”SpareBank 1” some general questions were asked, which will be summarized here.

Insight requests
Insight request are when a customer asks an enterprise about what kind of data they have stored about them. The enterprise is legally obliged to provide the information to the customer. The number of insight requests ”SpareBank 1 SR” receives yearly is 5-10. By taking into consideration, that this is 25% of the ”SpareBank 1”, the number is likely between 20-40 yearly. This number is relatively low. Unfortunately, ”SpareBank 1” could not give any concrete feedback on how this number has changed over time. However, given the increased interest in privacy, they expect the number to increase.

When handling an insight request ”SpareBank 1”s policy is to response as quickly as possible, with a limit of 30 days. The people in charge of the insight requests are the privacy-federation within the bank.

Data and technology
The company is currently not using any NOSQL databases. The standard response format they use are XML via SOAP. This information is of interest to get an idea of the technology used in the industry.

6.1.4 Number of Enterprises that Uses Information About Persons.

When asking ”Datatilsynet” about how many enterprises use information about persons, they directed the project to a service[1] which enables anyone to search for companies using information about persons. As of 2016.04.10 the number of enterprises on the list was 12397.

6.2 Quantitative Survey Data

The survey has 106 participants. The focus of the survey was to get data of the number of customer relationships people have today. Among the participants 37,7 % were female and 62,3 % male (Figure 6.3). The majority of the participants were of the age group 20-29 presenting 48,1 % of the population (Figure 6.2). The main phone type of the population

[1] The service is reachable at: https://melding.datatilsynet.no/melding/report_search.pl

were android as seen in Figure 6.4. And the main education type were master degree (Figure 6.5).

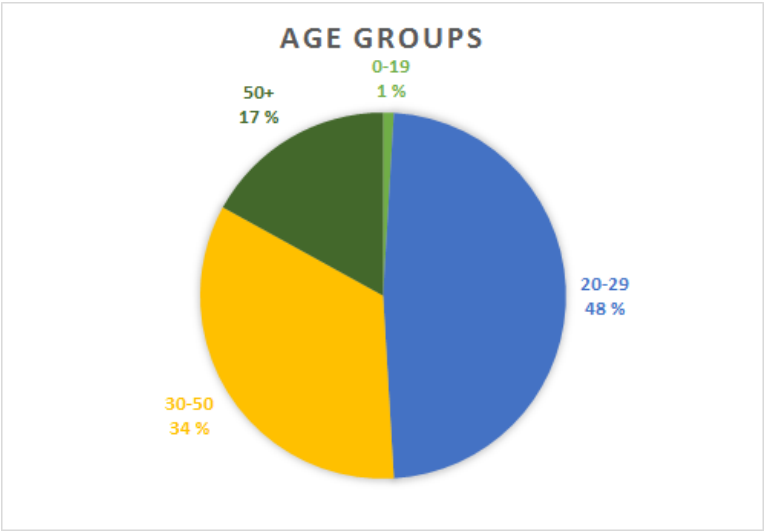

Figure 6.2: The population of the participants

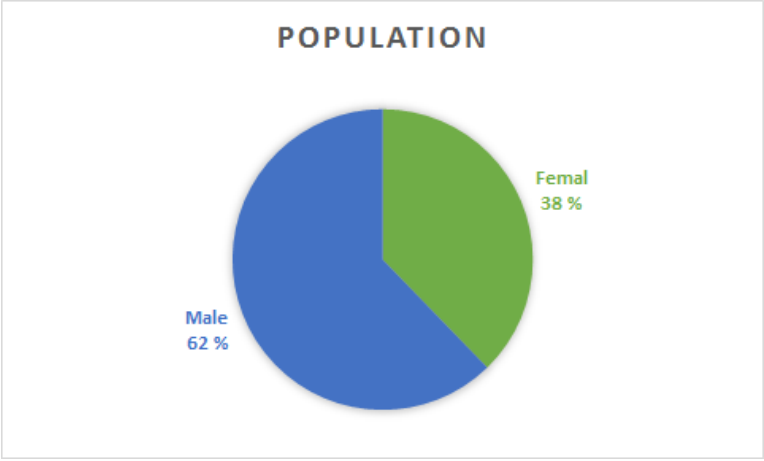

Figure 6.3: The Number of Participants and their Age Groups

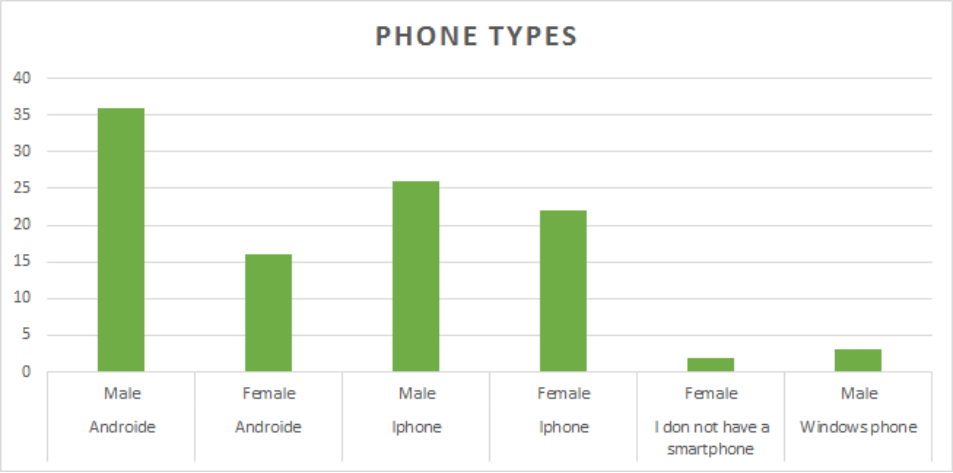

Figure 6.4: The Different Types of Phones

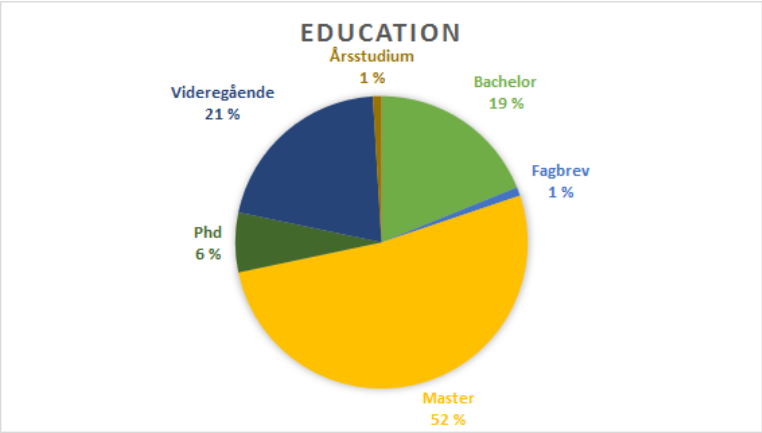

Figure 6.5: Education Types

An interesting observation of the survey were the number of memberships or customer relationships the participants have, which ranged from 10 - 64. The average of memberships were 19.69. It shows that there is no big difference in number of memberships between male and female, as shown in Figure 6.6. The same is the case for the number of customer relationships based on sex and age as seen in Figure 6.7.

Figure 6.6: Average Memberships Divided by Sex.

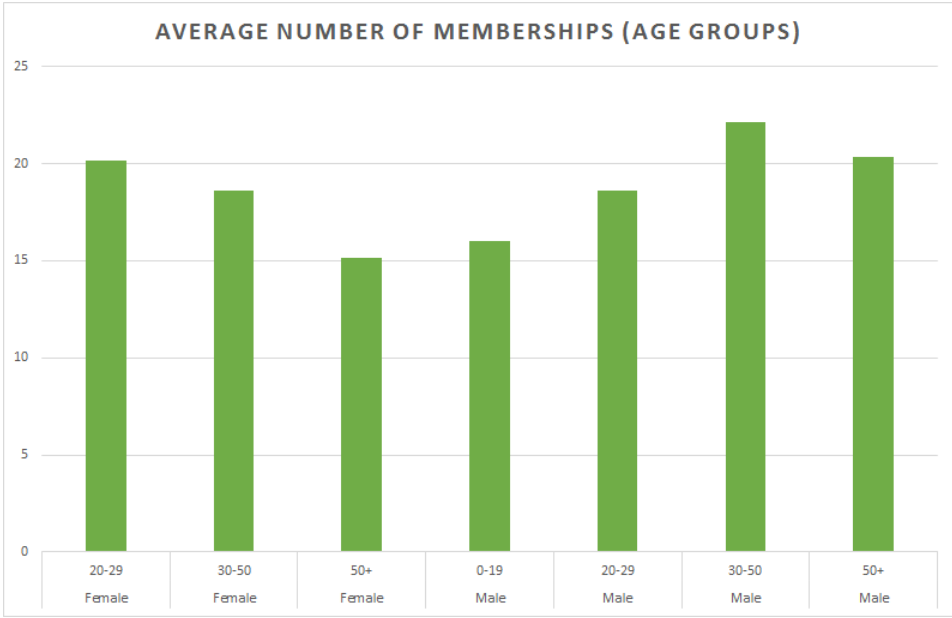

Figure 6.7: Average Memberships Divided by Sex and Age Group.

What this survey concludes is that the number of memberships and customer relationships are quiet high. This is not very surprising, but an important fact to emphasise.

6.3 Test Data

6.3.1 Calculated Threshold Values

To verify whether the system is capable of handling the workload, threshold values are calculated based on collected data.

Performance
This project calculates some threshold values, by using the data from "SpareBank 1" and the number of enterprises who use personal information. When it comes to performance, there are two qualities to consider, the number of transactions the system should handle, and the response time for the transactions. As stated earlier in this chapter (6.1.2), the lowest average response time for the different transactions measured by "SpareBank 1" was 675 ms. Given that this is for the whole stack tree, the response time that the system should be able to deliver is stipulated to 25% of 675 ms which is approximately **160 ms**.

The number of transactions is calculated through extrapolation. By making three categories of enterprise usage of information about people. Each of the categories gets an estimated amount of transactions each day. Then put percentages of the enterprises into each category. The categories this project presents are; daily, monthly, and occasionally. The occasionally category would estimate five transactions daily, the monthly category would do 100 transactions daily, and lastly the daily category would use the transaction volumes given by "SpareBank 1" (30 transactions per second). This can all be summarized in the table 6.2.

Table 6.2: Categories of Transactions Occurrences

Category	Transactions
Daily	2 592 000[2]
Monthly	100
Occasionally	5

Daily
The total number of enterprises estimated was **12 397** (chapter 6.4). It is obvious that not many enterprises will fall into the category of daily. This will typically be banks and other major enterprises that is paid to handle information about people. Based on this assumption the percentage of enterprises in this category will be 0.5%. The total number of enterprises becomes approximately: $12397 \cdot 0.005 = 100$ enterprises.

Monthly
The number of monthly enterprises will likely be relatively low. This is because a majority of enterprises chooses to use other firms to outsource the favour of sending out monthly

[2]This number is calculated multiplying the "SpareBank 1" number of transactions per second (30) by the amount of seconds in a day.

bills, paychecks etc. Based on this assumption this project estimates the percentage of enterprises to be 1% which is approximately: $12397 \cdot 0.01 = 200$ enterprises.

Occasionally
By power of elimination, the rest of the enterprises will be occasionally. Which is 98.5 % of the enterprises, which gives: $12397 \cdot 0.985 = 12200$ (rounded down).

Total number of transactions
This means that the total number of transactions IDMegler would at least have to manage each day is: $100 \cdot 2592000 + 200 \cdot 100 + 12200 \cdot 5 = 259281000$.

The threshold value for number of transactions per second calculated for this project is: 3 000. 3 000 transactions per second is the average amount stipulated. The peak is stipulated to be 50% more than the average. The threshold value for this project then becomes **4 500** transactions per second.

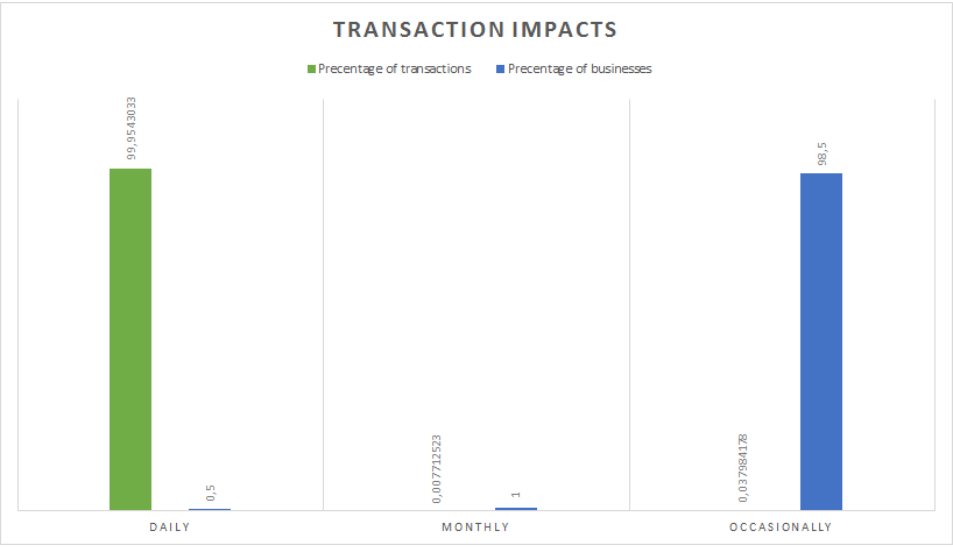

Figure 6.8: Transaction Impact

Figure 6.8 shows that the majority of the enterprises traffic is almost neglectable. The real traffic for IDMegler comes from the enterprises in the daily category.

6.3.2 Tests

Looking at the threshold values, there are different scenarios where the servers would have to handle different kinds of enterprises. Some servers will have to handle many different enterprises, while others will have many transactions coming from one enterprise. This calls for this project to test two different scenarios: One server with a lot of transactions form one enterprise, and one server getting few requests from many enterprises.

The purpose of the performance tests is to test if the system handle the stipulated amount of transactions per second, and at the same time keep the response-time within the calculated threshold value. Because the system does not use any form of secure channels in the implementation, it is expected that there will be an increase in the response time for the actual implementation of IDMegler. This is based on the assumption that complexity adds time to perform an action, and security adds complexity.

Test scenario

Name	Inputs	Duration
Performance test 1.1	Run the tests with 1 transaction per second and 10 enterprises.	10 min
Performance test 1.2	Run the tests with 1 transaction per second and 50 enterprises.	10 min
Performance test 1.3	Run the tests with 1 transaction per second and 100 enterprises.	10 min
Performance test 1.4	Run the tests with 1 transaction per second and 200 enterprises.	10 min
Performance test 1.5	Run the tests with 1 transaction per second and 500 enterprises.	10 min
Performance test 2.1	Run the tests with 10 transaction per second and 1 enterprise.	10 min
Performance test 2.2	Run the tests with 20 transaction per second and 1 enterprise.	10 min
Performance test 2.3	Run the tests with 50 transaction per second and 1 enterprise.	10 min
Performance test 2.4	Run the tests with 100 transaction per second and 1 enterprise.	10 min
Performance test 2.5	Run the tests with 200 transaction per second and 1 enterprise.	10 min
Performance test 3.1	Run the tests with 10 transaction per second and 5 enterprises.	10 min
Performance test 4.2	Run the tests with 20 transaction per second and 5 enterprises.	10 min
Performance test 4.3	Run the tests with 50 transaction per second and 5 enterprises.	10 min
Performance test 4.4	Run the tests with 100 transaction per second and 5 enterprises.	10 min
Performance test 4.5	Run the tests with 200 transaction per second and 5 enterprises.	10 min
Performance test 5.1	Run the tests with 10 transaction per second and 10 enterprises.	10 min
Performance test 5.2	Run the tests with 20 transaction per second and 10 enterprises.	10 min
Performance test 5.3	Run the tests with 50 transaction per second and 10 enterprises.	10 min
Performance test 5.4	Run the tests with 100 transaction per second and 10 enterprises.	10 min
Performance test 5.5	Run the tests with 150 transaction per second and 10 enterprises.	10 min

Table 6.3: The Tests Preformed on the Set-up

As can be seen from table 6.3, there are two different kinds of tests. The first scenarios sees how well the system handles many enterprises and only one transaction per second. This checks that the server will be able to handle many different enterprises using the same server. The second part of the test scenarios sees how well one server handles many transactions per second in three different cases. One where it is only one enterprise sending the requests, one where five enterprises sends requests and lastly one where ten enterprises sends requests.

6.3.3 Preformed tests

The tests preformed uses the following guidelines:

- All response times are measured in milliseconds.
- All of the test scenarios are done three times to make sure the data collected was accurate.

Table 6.4: One Request Per Second to Many Enterprises

Requests/ second	#Ok	# Not found	Enterprises	Average re- sponse time	Longest re- sponse time	Shortest re- sponse time	Total test time
1	6000	0	10	4.6	178	1.64	10:00.0
1	6000	0	10	7.96	3014.6	1.58	10:00.0
1	6000	0	10	10.62	3033.64	1.57	10:00.0
1	30000	0	50	13.2	3043.064	1.45	10:00.0
1	30000	0	50	17.84	3029.39	1.45	10:00.0
1	30000	0	50	11.29	3026.27	1.50	10:00.0
1	60000	0	100	17.08	3150.19	1.51	10:00.0
1	60000	0	100	19.38	1025.70	1.46	10:00.0
1	60000	0	100	16.43	3038.57	1.40	10:00.0
1	119913	0	200	31.07	3038.16	1.45	10:00.0
1	120000	0	200	15.15	3031.10	1.40	10:00.0
1	119912	0	200	27.99	3020.87	1.37	10:00.0
1	299495	0	500	60.43	3020.72	1.48	10:00.0
1	299704	0	500	39.01	3024.72	1.42	10:00.0
1	299464	0	500	79.66	1325.83	1.39	10:00.0

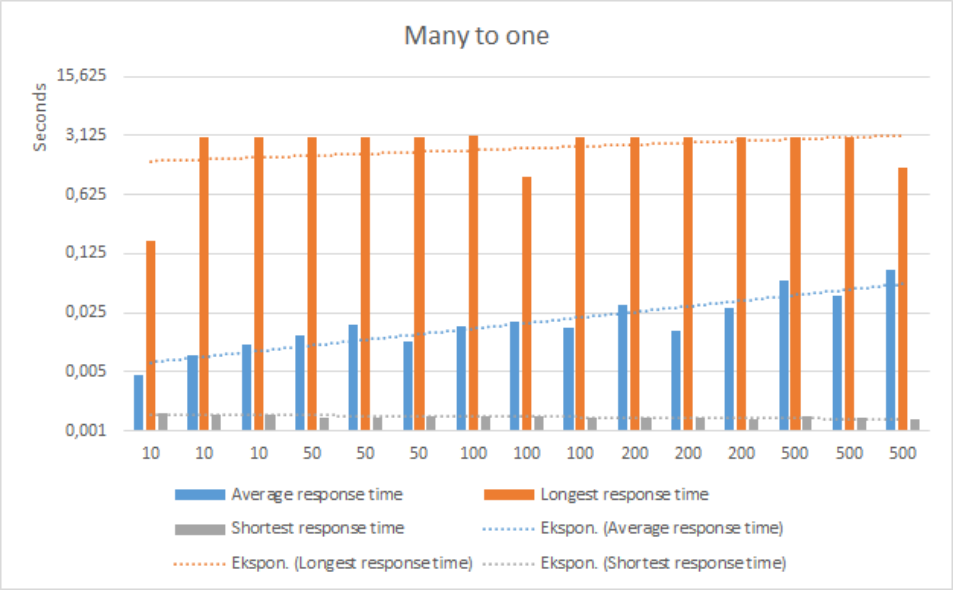

Figure 6.9: Many to One Response Time

Table 6.5: One Enterprise to Many Requests

Requests/ second	#Ok	# Not found	Enterprises	Average re- sponse time	Longest re- sponse time	Shortest re- sponse time	Total test time
10	5952	0	1	3.62	341.97	1.71	10:00.0
10	5953	0	1	4.01	605.67	1.63	10:00.0
10	5954	0	1	3.42	312.76	1.77	10:00.0
20	11852	0	1	4.14	931.39	1.56	10:00.0
20	11847	0	1	3.24	329.56	1.57	10:00.0
20	11853	0	1	4.21	981.53	1.57	10:00.0
50	28864	0	1	2.89	319.98	1.39	10:00.0
50	28876	0	1	2.93	316.00	1.41	10:00.0
50	28884	0	1	4.18	1152.95	1.38	10:00.0
100	55081	0	1	2.99	3019.09	1.34	10:00.0
100	54985	0	1	5.41	3040.95	1.36	10:00.0
100	54916	0	1	3.19	346.72	1.32	10:00.0
200	100748	0	1	3.6	3017.13	1.31	10:00.0
200	100674	0	1	5.15	3029.45	1.34	10:00.0
200	99104	0	1	3.92	3026.26	1.30	10:00.0

An interesting observation to notice is from testing ”1 enterprise, 20 transactions” (blue marked lines in table 6.5) and ”1 enterprise, 50 transactions” (green marked lines in table 6.5) it appears that the average response time goes down. When testing ”1 enterprise, 50 transaction”, the IKDM server used up all of its resources (CPU and RAM) to handle the logs sent from the LRIM server. This made the IKDM server shut down its port for receiving logs. When testing ”1 enterprise, 50 transactions” the IKDM server did not receive the logs and then did not have to use processing power to handle the logs, which explains the reduction in response time. This means that the LRIM component is capable of handling the workload but not the IKDM component.

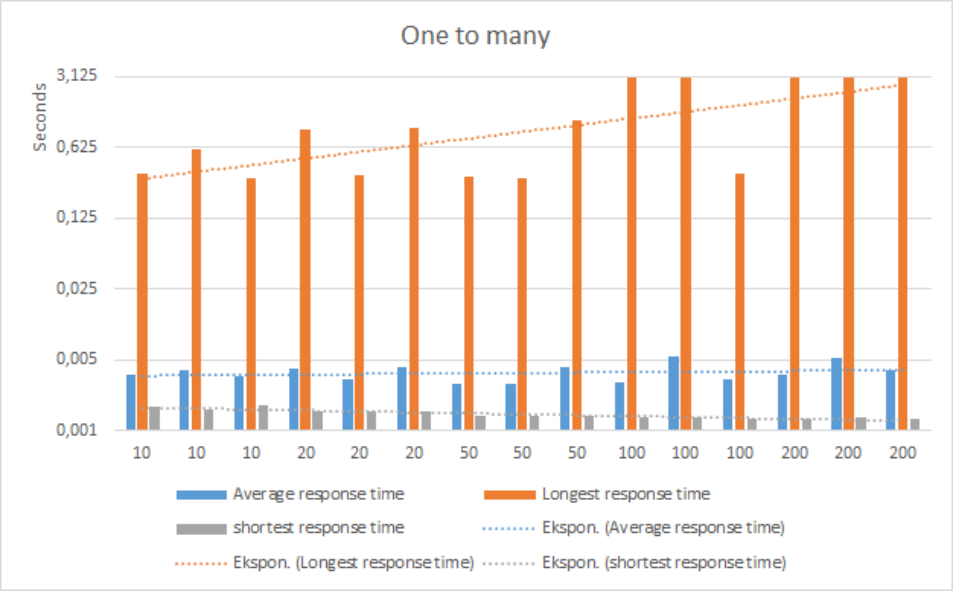

Figure 6.10: One to Many Response Time

Table 6.6: Five Enterprises To Many Requests

Requests/ second	#Ok	# Not found	Enterprises	Average re- sponse time	Longest re- sponse time	Shortest re- sponse time	Total test time
10	29770	0	5	3.84	3017.92	1.38	10:00.0
10	29759	0	5	3.98	3013.76	1.36	10:00.0
10	29770	0	5	5.39	3027.73	1.36	10:00.0
20	59178	0	5	3.77	3025.88	1.36	10:00.0
20	59185	0	5	4.79	3022.29	1.33	10:00.0
20	59205	0	5	3.27	3032.42	1.37	10:00.0
50	143744	0	5	7.36	3017.47	1.35	10:00.0
50	143710	0	5	5.63	533.96	1.35	10:00.0
50	143645	0	5	5.07	3019.56	1.35	10:00.0
100	269559	0	5	14.36	3019.15	1.33	10:00.0
100	268810	0	5	27.2	1306.99	1.36	10:00.0
100	268974	0	5	28.33	1413.79	1.34	10:00.0
200	435727	0	5	68.36	1426.50	1.39	10:00.0
200	440117	0	5	167.69	3415.00	1.34	10:00.0
200	433118	0	5	376.22	7779.79	1.37	10:00.0

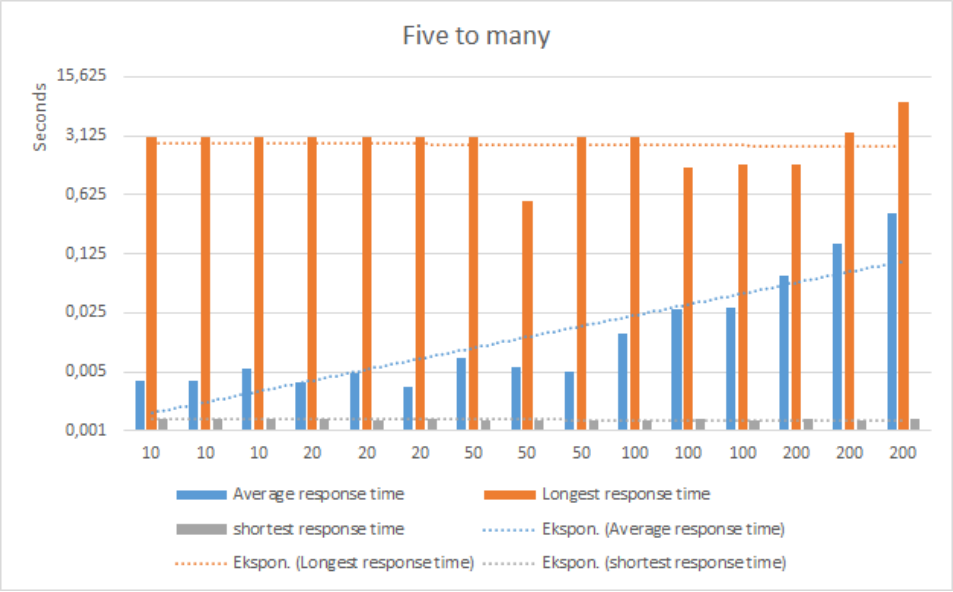

Figure 6.11: Five to Many Response Time

Table 6.7: Ten Enterprises to Many Requests

Requests/ second	#Ok	# Not found	Enterprises	Average response time	Longest response time	Shortest response time	Total test time
10	59521	0	10	3.99	335.28	1.41	10:00.0
10	59511	0	10	7.97	3022.52	1.37	10:00.0
10	59537	0	10	3.88	3014.95	1.35	10:00.0
20	118166	0	10	7.29	3021.11	1.35	10:00.0
20	118052	0	10	4.87	328.44	1.36	10:00.0
20	118156	0	10	4.77	3010.83	1.32	10:00.0
50	283370	0	10	20.75	871.42	1.37	10:00.0
50	283072	0	10	17.40	3020.73	1.33	10:00.0
50	281783	0	10	29.83	3018.57	1.35	10:00.0
100	485098	0	10	111.2	2735.31	1.37	10:00.0
100	506449	0	10	62.77	3022.75	1.38	10:00.0
100	502633	0	10	91.42	4462.63	1.37	10:00.0
150	664774	0	10	23528.24	73919.63	1.42	10:00.0
150	665202	0	10	24585.29	73374.42	1.84	10:00.0
150	657479	0	10	16858.46	55531.33	2.47	10:00.0

In the latest two set-ups (”10 enterprises, 100 transactions” and ”10 enterprises, 150 transactions”) from table 6.7, it is clear that the server cannot respond properly to all of the requests. By looking at the number of OK requests and the duration of the tests, there seems to be an upper limit for requests the servers can handle. By looking at the ”10 enterprises, 100 transactions” first, the amount of requests per second becomes: $485098/6000 = 81, 506449/6000 = 84$, and $502633/6000 = 84$. This is approximately 15% less than the request load which is 100 transaction. This is not a result of how the tests are done; the threads generating the requests are not aborted, but simply stopped. The remaining requests which had been sent showed up in the recorded data after the ending of the test period of 10 minutes. The number is even worse for the ”10 enterprises, 150 transactions” scenario: $664774/6000 = 111, 665202/6000 = 111$, and $657479/6000 = 109$. This is a reduction of approximately 27%. The interesting part is that in the second scenario the server is capable of making over 100 requests per second. However, the response times are horrendous and far outside of any threshold values set for a satisfactory system.

During the testing of the scenario with 10 enterprises and several requests per second, it was first planned to send 200 requests per second form each of the 10 enterprises in the last scenario. However, the enterprise simulator crashed after a while due to the amount of requests generated and the server were not able to handle all of them. To get data on higher settings this number was reduced to 150 as seen in the table 6.7. It can be said with certainty that the system with one server handling enterprise requests cannot handle 2 000 requests per second for a very long time.

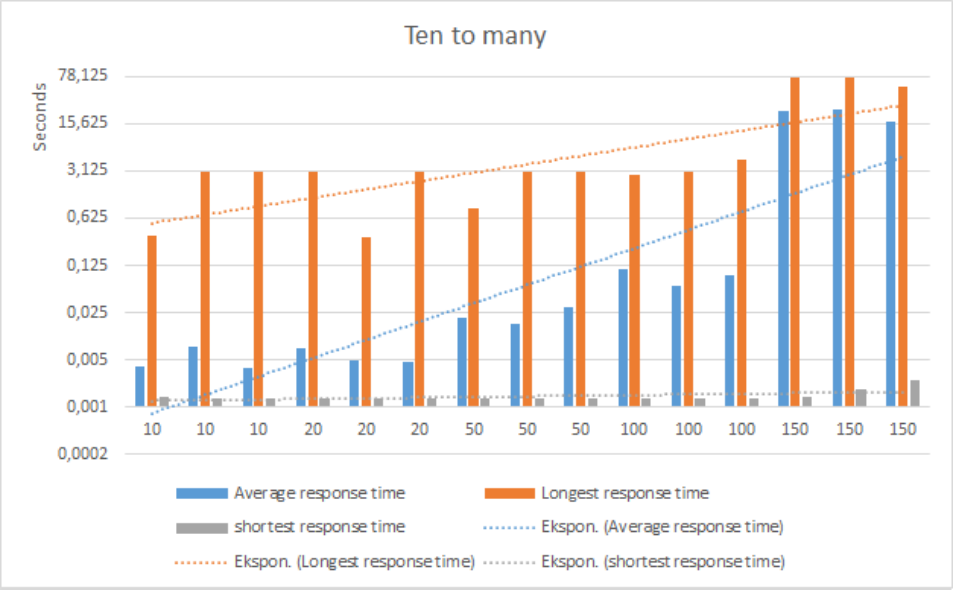

Figure 6.12: Ten to Many Response Time

During testing, a problem arised with the server running the IKDM-IDMegler component. The IKDM component used too much processing power and memory so the socket used for retrieving the logs shut down. With 4 GB of memory and 1 CPU (Intel(R) Xeon(R) CPU E5-2650 0 @ 2.00GHz), the server was capable of handling 25 transactions per second. After doubling the memory and CPU power, the server managed 50 transactions per second. It becomes apparent that the server would need 30 times the CPU power and 240 GB of ram to handle the workload for the tests done in this project. This clearly implies a bottleneck that needs to be addressed. One solution could be to raise the memory and processing power on the server. Another approach would be to reduce the number of logs sent from the LRIM component. By storing the logs temporary in the LRIM component, it would be possible to compress the logs. This way the value of the logs would still be present, but the amount of logs sent to the IKDM component would be reduced.

The tests imply that the system and architecture should be able to handle the workload. According to the tests, one LRIM server is capable of handling around 1 000 requests per second. If scaling up the amount of LRIM servers, there should be close to no limit to how many requests per second the system and architecture is capable of handling.

6.4 Summary

This chapter have described the data of all tests, interviews, and surveys done in this project. Then presented the calculations and threshold values that the system should be

able to handle, described the performed tests on the implemented system, and briefly discussed the findings.

Chapter 7

Discussion

This chapter discusses all results, findings, and contributions throughout the project. A reflection and retrospective of the DSRM Process is given at the end.

7.1 Data Collection

The data collection in this project was partly successful. ”SpareBank 1” was able to give the number of transactions from their external channels but not from their internal channels. This means that an unknown amount of transactions had to be estimated. In addition to this, there were no data from smaller enterprises nor from other sectors than ”SpareBank 1”. The project should have contacted more enterprises of different size from various sectors, to achieve a full set of data about their numbers of personal data transactions. In this project, those numbers were estimated by assuming the other enterprises were less likely to use information about persons. This could be a source of error in the estimation. However, based on the performed tests, the system should be in good shape to handle a much higher number of transactions per second then the total estimated number in this project.

7.2 Survey

Survey objective
It is important to make sure that there is a purpose to any survey. The reason for this survey was to get an idea of the number of enterprises a person is linked to and to confirm the assumption that people lack control over their own information. It was important to design the survey in a way where people could give as much valid information as possible. People should ideally answer questions, which are relevant to them to get the best results.

The survey tries to cover the most areas of enterprises a citizen can be linked to. However, the variety of enterprises is so vast that the coverage has limitations.

Researcher bias
The researcher bias is a challenge. Another challenge is designing the survey with good coverage of the problem domain. The questions asked becomes a result of the authors own experience. This makes the questions bias towards the authors own knowledge of enterprises. A more thorough research about possible enterprises would have increased the quality of the questions. The survey ends in an open question encouraging the participants to name all enterprises they remember to have, or have had, a customer relationship with, to accommodate for the lack of knowledge about possible enterprises. However, only 24/106 people choose to answer the question with very varying amounts of detail.

Survey sample
One of my two supervisors and I shared the survey on Facebook. This makes the sample of people relatively centred on people in the same age group and life situation as my supervisor and me. Ideally, the sample size should have included many other types of people, and not mainly students and people of higher academic background.

Conclusion
The survey manages to give an indication of the knowledge people have about which enterprises they are related to. Additionally the survey is able to point out that people have big variation in number of enterprise relationships they are aware of. The numbers range from 12 to 61, and this is just a subsample. Of the 24 who answered the open question, 8 said that they expected the number to be much higher but were unable to remember any other enterprises. This makes it safe to say that very few people have complete control over the distribution of information.

7.3 System Design and Architecture

7.3.1 Reflections

Metrics
This project focused on two metrics:

- Performance (time from request to returned data)
- Requests handled per second (the amount of transactions per second the implementation can handle)

Response time is important when using IDMegler. The request time between the enterprise and the response time from IDMegler need to be a lot lower compared to the response time measured at the enterprise (this will be a fraction of what the enterprises likely measures today given that IDMegler will always be part of a system chain). It needs to be a constant factor and not operate differently based on the request load.

The amount of request per second IDMegler can handle is crucial for the system to be successful. If IDMegler is not able to handle the total workload of the entire Norwegian nation, the system will not be beneficial. This means that the system must be able easily to handle the peak workload stipulated for the entire nation.

REST
REST is an easy and good solution for sending requests and making web services. This allows testing of response time from different physical locations. In addition, it is easy to implement. There exist many frameworks to help with implementation.

Database
There are multiple scenarios where a database is important. The IKDM service needs a temporary quick lookup table as well as the ability to cache any incoming data. By using NOSQL databases, the read/write speed is increased immensely compared to a RDBMS.

The IKDM service needs to store big amounts of data. This emphasises the importance of negating any data loss, especially the key values and the logs. This makes the use of NOSQL databases less optimal since there is a higher risk of losing data compared to a relational database. However, by using databases like RavenDB, which implements ACID, this no longer becomes a problem.

7.3.2 Architecture

The IDMegler architecture is described in various different diagrams and process flows (Chapter 4.5). The overall structure first presented, gives a physical view about the distribution possibilities. The process flow diagrams for the citizens and enterprises is shown in more detail displaying the main processes of IDMegler. The dataflow diagrams give an indication of the internal components in IDMegler. Lastly, the sequence diagram shows in detail how a request from an enterprise will look like in an implementation of this architecture. In general, the majority of the diagrams repeat the different components. This ties the diagrams together, and it is meant to increase readability and comprehensibility.

The system design and architecture were a major part of the project. The architecture was developed through a series of iterations. Part of the redesign was accomplished during the implementation. In retrospective it would have been better to analyse the system design and architecture to a greater extent before starting the implementation. The reason for the early start of implementation was to ensure that it was possible to generate a system capable of enduring tests that would generate relevant information. During the implementation, many problems were discovered and several changes were necessary. As problems were discovered, it became apparent that the initial design and architecture were not capable of addressing all of the problems.

Increased focus on the early parts of the specification and design would be crucial if the project was to be executed again. However, it is often hard to make the right choices in the design phase, because at that point in time you have to make crucial decisions when you know the least about your system.

7.3.3 Problems

Cold start-up
There are two different ways for the system to handle the cold start: Lazy and active.

Lazy: The LRIM component only retrieves key-chain information from the IKDM component when requested from the enterprise. The different enterprises should already be loaded into the LRIM component.

Active: All information about all the enterprises and their respective customers should be available when started.

A cold start poses a problem and could potentially be slow (If x numbers of servers request hundreds of millions entries from the IKDM server). However, this was not tested in this project, but could potentially be tested in future work.

7.3.4 Implementation

A lot of work can still be done to improve the implementation, and to add functionality to IDMegler. During the implementation, some simple security elements were coded into the implementation, but removed due to problems with certificates.

Further work can be done in the area of security and in adding more functionality to the solution.

Some of the code could have been written better and there is a lack of comments in the code. The program in itself is not hard to understand but some documentation (exceeding Chapter 5) should be added.

7.3.5 Functional Requirements

In Chapter 4.1 the functional requirements this project should implement were specified. The requirements were:

FR6: IDMegler shall be able to retrieve personal information about a customer.
FR7: IDMegler shall be able to accept requests about personal information from valid enterprises.
FR9: IDMegler shall be able to find the right person in ”Folkeregisteret” based on enterprise and customer ID.
FR17: The logs shall not contain any sensitive information. Every log shall use numbers to represent the enterprise and customer, as well as numbers for placeholders for transaction types.
FR23: IDMegler shall be able to identify a person based on enterprise and customer id.

All of the functional requirements were implemented fully except for FR7. The implementation is capable of accepting requests from enterprises, however, the project did not implement any functionality to check if the enterprise is valid.

7.3.6 Test and Performance

Findings

- The architecture of the system is capable of handling the workload
- There is a bottleneck with the number of logs generated
- The system needs more complete tests

The architecture of the system is capable of handling the workload
The findings of the tests were that the architecture of the system was able to handle the workload. A single LRIM server is capable of responding to 1 000 transactions per second. The total amount estimated to the complete Norwegian marked is 4 500 transactions per second. The average response time for a server handling 1 000 transactions per second was around 100 ms. This was tested on a server who had 10 enterprises which each made 100 requests per second. The estimate for a big enterprise (”SpareBank 1”) was 32 requests per second. In a scenario where a company of that size has their own server, the average response time should in general be around 3 ms.

There is a bottleneck with the number of logs generated
The problem when testing the system was that the IKDM server was incapable to handle the number of logs. However, this does not indicate that the LRIM server is incapable of handling the transactions. The way the system makes and collects the logs needs a redesign. A possible solution might be to save the logs within the LRIM servers. When a citizen wants to look at his transaction history, the system retrieves the logs from the various LRIM servers. A second solution could be to let the enterprises store the logs locally, and require that they can deliver the logs on request from IDMegler. A third possible approach could be to find a good way to compress and reduce the number of logs at the LRIM server. This would not reduce the value of the information for the citizens, because his interest is not in knowing all the logs in detail but a summary and explanation of what the enterprises use their personal information for.

The system could need more conclusive tests

- The system needs a full availability test over longer periods
- The master/slave part of the system is not fully tested for scalability (only one slave (LRIM))
- The tests scope should optimally be bigger
- The system does not implement any security and this is therefore not tested

The tests do not include any availability test. This means that even though the data from the tests give a good indication of the capability of the architecture; the quality attribute is

not fully tested. The tests try to include some availability, by having a decent test period. However, to say anything specific about the availability, the implemented prototype would demand a test that runs at least 72 hours. The reason for not making complete availability tests was that it is difficult to perform and takes a lot of time. Since this project had a limited time frame, these tests were excluded from the test scope.

The project did not test the master/slave architecture with more LRIM components. The set up for all the tests included only one LRIM (slave) component and one IKDM (master) component. This means that there is no data for how the prototype system would behave in an environment with more LRIM components. A test with more LRIM components might identify the improved version of ”Folkeregister” to be a bottleneck as well. In the scenario that Norway decides to implement and enforce IDMegler, it will likely be in the nation’s interest to make sure that ”Folkeregisteret” can handle the full load of transaction. As a matter of fact, this uncertainty will likely not pose a problem.

It would be interesting to address the situation were the IKDM component would handle the logs when more than one LRIM component is sending them. The master would still have problems with the amount of logs, but it would nonetheless be an interesting scenario that might shed some light on possible ways to handle the bottleneck.

The system tests scope should have a larger and more realistic set of testdata. This project did tests where every request only requested data about 10 different individuals stored in the prototype system. In addition, all of them belonged to the same enterprise. By the nature of how the implementation adds logs, a more thorough test would likely increase the time it takes to add them. When adding the logs to the IKDM server (the logs are sent in batches of 500), the server sorts the logs based on which enterprise the logs come from. The server retrieves the document (the document holds all the information about the enterprise) from the NOSQL database (RavenDB) for each enterprise and write the logs into the document. Therefore, if many logs from different enterprises arrive, more documents need to be retrieved from the database, which will increase the time it takes to add the logs. This would increase the need for making a smart logging system in IDMegler.

There is no security in the implemented prototype system. This project had no focus on the security aspect of IDMegler. However, security in general, adds up to the complexity of a system. With more complexity the performance of the system usually decreases.

The results of the test were promising. The number of transactions per second by one LRIM server was able to handle 1 000 transactions per second and preformed with a lower response time than the threshold calculated in Chapter 6. This is 20% of the estimated transactions Norway would need. This implementation of IDMegler, does not consider a scenario where an enterprise wants to retrieve a group of its customers. How to handle such a scenario will need further discussion. How often such a feature is needed would also have to be address. By improving the feature for handling logs and adding security to the system, the prototype would be close to a deployable implementation of IDMegler.

7.4 Validity

Here the process of the project is evaluated, against the DSRM process.

7.4.1 DSRM Process

Peffers and Tuunanen [34] presents the DSRM process. This process is a way to capture the essence of design science research. The 6 different activities will be reflected upon here, to see whether or not this project follows the method.

Activity 1: Problem identification and motivation
The first activity is supposed to identify what the design problem is, as well as to evaluate the idea for solving the problem. The first part of this project identifies the problem through the motivation and explanation of IDMegler. The motivation is justified through the will to improve the privacy and problems with profiling.

Activity 2: Define the objectives for a solution
This activity is supposed to define the benefits of the new solution and compare it to the former design. At the start of this project, the research questions were defined. The threshold values the system would have to accomplish, were developed throughout the project by gathering information from various sources. The estimation of the threshold values was done at the same time as the implementation, but ideally, it would have been better to do the estimations prior to the design and development.

Activity 3: Design and development
This activity should create the artefacts and define the artefacts functions. This project creates the artefacts in Chapter 4, and in Chapter 5, the implementation is described. This project did not follow this strictly, given that throughout the implementation, certain restrictions were discovered. This lead to a redesign of some of the artefacts.

Activity 4: Demonstration
This activity is meant to be a showcase of the artefacts created. This was presented in chapter 6. By showing how the system performs and what the system is capable of, the key features are highlighted.

Activity 5: Evaluation
This activity evaluates the artefact. This project tests the artefact in Chapter 6 and the results are discussed in Chapter 7.

Activity 6. Communication
This activity is supposed to communicate the weaknesses in the design and the improvement potentials. This part is done in Chapter 7.

Overall, the project uses all the different activities of the DSRM process. During the different activities there have been redesigns of the solutions and of artefacts created. Even though the design process has been followed, in retrospect the project should follow the principles more systematically if done again.

Chapter 8

Conclusion and Future Work

This project has worked with architecture, data collection, implementation, and testing of a suggested system called IDMegler. The tests performed suggest that the architecture developed through this project is satisfactory and should be able to handle the workload. However, different test scenarios remain due to the limited time of the project and there is a bottleneck with the number of logs that have to be solved. It would be interesting to find out how the system handles multiple LRIM components, and how it behaves over a longer period of time to verify the stability of the system.

8.1 Research Questions Conclusion

The four research questions defined in Chapter 1 were as follows:

RQ1: What are possible bottlenecks and limitations of IDMegler?
RQ2: What is the minimum amount of traffic a system like IDMegler should be able to handle?
RQ3: How can the architecture be designed to cope with the workload?
RQ4: How can the architecture fulfil the functional requirements of the solution?

RQ1: What are possible bottlenecks and limitations of IDMegler? One major bottleneck was identified during testing. The sheer amount of logs generated was too massive for the IKDM component to handle. This was an interesting find but not a big surprise. A way to handle this problem could be to store the logs with the different LRIM components and then let the system retrieve the logs from the LRIM components when a citizen wants to look at his history. A second solution could be to let the enterprises store the logs locally, and require that they can deliver the logs on request from IDMegler. A third possible approach could be to process the logs at the LRIM component and send a more compact number of logs to the main component.

Security is very important for IDMegler. There is a possibility that security could be a bottleneck for performance in IDMegler. When adding security to a system, the complexity of the system is increased. Usually, complexity adds time when performing a transaction. As a result, the time it takes to perform each request in IDMegler could increase. This would have to be tested further to determine if this is a bottleneck.

In the discussion, it has been pointed out that performance from a national wide ”Folkeregister” could be a bottleneck for very high peaks of transaction volumes. Nevertheless, in the scenario that Norway decides to implement and enforce IDMegler, it will likely be in the nation's interest to make sure that ”Folkeregisteret” can handle the full load of transaction.

RQ2: What is the minimum amount of traffic a system like IDMegler should be able to handle? A stipulated requirement for transaction traffic were calculated. This number ended up being 4 500 transactions per second. The transaction requirement was mainly based on the information provided by ”SpareBank 1”. In addition, a threshold based on response time data from ”SpareBank 1” was calculated to 160 ms for IDMegler.

RQ3: How can the architecture be designed to cope with the workload? Based on the tests preformed, the implemented architecture of IDMegler was able to accommodate the workload. If the tests were extended to a more diverse set of scenarios, it would be possible to conclude with more certainty. However one LRIM component were able to handle a load of about 1 000 transactions per second with a good average response time. This means that for all of Norway's needs, according to the estimation, IDMegler would only need five to six servers. An absolute architecture needs to be able to handle the security aspect, have a good enough interface for the enterprises, a good enough user interface for the citizens, and to be incorporated into Norwegian laws and regulations. The project conclude based on the tests performed that the purposed architecture is capable of coping with the workload.

RQ4: How can the architecture fulfil the functional requirements of the solution? The implemented architecture were able to fulfil the subset of the functional requirements (FR6, FR7, FR9, FR17, FR23). The implementation is only a subset of the total requirement specification, so more work on different features should be done. It was not feasible to implement every aspect of IDMegler in this project, but there is nothing in the architecture that would hinder further implementation of the functional requirements. The architecture also needs to be tested more thoroughly on the availability aspect. All of these improvements are possible further works.

An important thing to discuss is, how IDMegler will affect an existing enterprise? An enterprise would have to make adjustments in all its current systems to be able to connect to IDMegler. The enterprise would have to isolate and identify all of its processes and find the ones where they need to communicate with IDMegler. In addition, there will be big costs connected to developing solutions and updating their existing processes to work with IDMegler. All of this will take time and be expensive. This means that it is achievable but it will be a significant decision. More information of the total cost and social and economic benefits of IDMegler is needed to determine if this kind of system is something to invest in.

8.2 Future Work

As mentioned throughout the project there are certain areas that would be interesting to further investigation and development. The following list shows the different possible future works. The future works is divided into three categories; technical, political, and development of the problem domain. The technical work could look into expanding on the implementation and architecture to improve technical aspects. The political work could look into how the IDMegler is received as a good investment for the society, both economic and political. Lastly, the problem domain work could do research about the cost benefit of implementing IDMegler at different enterprises. It would also be interesting to discover societies need for IDMegler. Some of these future works might be better performed outside the IT domain.

Technical

- Test the dependability quality of the implementation
- Develop and test the security aspect of the implementation (Test plan and penetration testing)
- Work out a better way to handle the logs
- Implement more of the functionality of IDMegler and test how it will affect the performance
- Create a good user interface for the insight service

Political

- Find out how the decision makers looks upon IDMegler
- Find out what the costs of IDMegler will be
- Categorize the social and political pros and cons of IDMegler
- What potentials has IDMegler to being a valid solution for Norway

Development of the problem domain

- Categorise different enterprises in different sectors and their use of personal information to get an insight of their needs for data requests (transactions)
- Expand on the questionnaire to categorise further peoples distribution of personal information and their general opinion of privacy

References

[1] Personal data act. https://www.datatilsynet.no/English/Regulations/Personal-Data-Act-/, 2015. accessed 08-December-2015.

[2] Asp .net. http://www.asp.net/, 2016. accessed 02-June-2016.

[3] Platform for privacy preferences. https://www.w3.org/P3P/, 2016. accessed 05-June-2016.

[4] RavenDB. https://ravendb.net/, 2016. accessed 01-June-2016.

[5] Redis. http://redis.io/, 2016. accessed 02-June-2016.

[6] Gunes Acar, Christian Eubank, Steven Englehardt, Marc Juarez, Arvind Narayanan, and Claudia Diaz. The web never forgets: Persistent tracking mechanisms in the wild. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 674–689, New York, NY, USA, 2014. ACM.

[7] Mark S. Ackerman, Lorrie Faith Cranor, and Joseph Reagle. Privacy in e-commerce: Examining user scenarios and privacy preferences. In *Proceedings of the 1st ACM Conference on Electronic Commerce*, EC '99, pages 1–8, New York, NY, USA, 1999. ACM.

[8] Solon Barocas and Helen Nissenbaum. Big data's end run around procedural privacy protections. *Commun. ACM*, 57(11):31–33, October 2014.

[9] Richard Baskerville, Jan Pries-Heje, and John Venable. Soft design science methodology. In *Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology*, DESRIST '09, pages 9:1–9:11, New York, NY, USA, 2009. ACM.

[10] Tania Basso, Regina Moraes, Mario Jino, and Marco Vieira. Requirements, design and evaluation of a privacy reference architecture for web applications and services. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, SAC '15, pages 1425–1432, New York, NY, USA, 2015. ACM.

[11] Imen Ben Dhia, Talel Abdessalem, and Mauro Sozio. Primates: A privacy management system for social networks. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, CIKM '12, pages 2746–2748, New York, NY, USA, 2012. ACM.

[12] Peter Bodorik and Dawn Jutla. Architecture for user-controlled e-privacy. In *Proceedings of the 2003 ACM Symposium on Applied Computing*, SAC '03, pages 609–616, New York, NY, USA, 2003. ACM.

[13] Piero Bonatti and Pierangela Samarati. Regulating service access and information release on the web. In *Proceedings of the 7th ACM Conference on Computer and Communications Security*, CCS '00, pages 134–143, New York, NY, USA, 2000. ACM.

[14] Jan Camenisch, abhi shelat, Dieter Sommer, Simone Fischer-Hübner, Marit Hansen, Henry Krasemann, Gérard Lacoste, Ronald Leenes, and Jimmy Tseng. Privacy and identity management for everyone. In *Proceedings of the 2005 Workshop on Digital Identity Management*, DIM '05, pages 20–27, New York, NY, USA, 2005. ACM.

[15] Daniel J. T. Chong and Robert H. Deng. Privacy-enhanced superdistribution of layered content with trusted access control. In *Proceedings of the ACM Workshop on Digital Rights Management*, DRM '06, pages 37–44, New York, NY, USA, 2006. ACM.

[16] Chi-Yin Chow and Mohamed F. Mokbel. Privacy in location-based services: A system architecture perspective. *SIGSPATIAL Special*, 1(2):23–27, July 2009.

[17] Roger Clarke. Whats privacy. *Australian Law Reform Commission Workshop*, 28, 2006. http://www.rogerclarke.com/DV/Privacy.html#Defn.

[18] Sebastian Clauβ, Dogan Kesdogan, and Tobias Kölsch. Privacy enhancing identity management: Protection against re-identification and profiling. In *Proceedings of the 2005 Workshop on Digital Identity Management*, DIM '05, pages 84–93, New York, NY, USA, 2005. ACM.

[19] David Crowe and Wasim A. Al-Hamdani. Google privacy: Something for nothing? In *Proceedings of the 2013 on InfoSecCD '13: Information Security Curriculum Development Conference*, InfoSecCD '13, pages 27:27–27:32, New York, NY, USA, 2013. ACM.

[20] Alfredo Cuzzocrea. Privacy and security of big data: Current challenges and future research perspectives. In *Proceedings of the First International Workshop on Privacy and Secuirty of Big Data*, PSBD '14, pages 45–47, New York, NY, USA, 2014. ACM.

[21] Hilde Visthoff Drange. Personvernsarkitektur. Master thesis, NTNU, 2013.

[22] Vegard Fossestøl. Personvernsarkitektur. Technical report, NTNU, 2014.

[23] Saikat Guha and Srikanth Kandula. Act for affordable data care. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, HotNets-XI, pages 103–108, New York, NY, USA, 2012. ACM.

[24] M. Hansen, A. Schwartz, and A. Cooper. Privacy and identity management. *IEEE Security & Privacy*, 6(2):38–45, March 2008.

[25] Alan R Hevner. A three cycle view of design science research. *Scandinavian journal of information systems*, 19(2):4, 2007.

[26] Steve Kenny and Larry Korba. Applying digital rights management systems to privacy rights management. *Computers & Security*, 21(7):648–664, 2002.

[27] Alfred Kobsa. A component architecture for dynamically managing privacy constraints in personalized web-based systems. In Roger Dingledine, editor, *Privacy Enhancing Technologies*, volume 2760 of *Lecture Notes in Computer & Science*, pages 177–188. Springer Berlin Heidelberg, 2003.

[28] L. Li and W. Chou. Design and describe rest api without violating rest: A petri net based approach. In *Web Services (ICWS), 2011 IEEE International Conference on*, pages 508–515, July 2011.

[29] Dheerendra Mishra. An accountable privacy architecture for digital rights management system. In *Proceedings of the Sixth International Conference on Computer and Communication Technology 2015*, ICCCT '15, pages 328–332, New York, NY, USA, 2015. ACM.

[30] Min Mun, Shuai Hao, Nilesh Mishra, Katie Shilton, Jeff Burke, Deborah Estrin, Mark Hansen, and Ramesh Govindan. Personal data vaults: A locus of control for personal data streams. In *Proceedings of the 6th International COnference*, CoNEXT '10, pages 17:1–17:12, New York, NY, USA, 2010. ACM.

[31] Andy Neely, Mike Gregory, and Ken Platts. Performance measurement system design: A literature review and research agenda. *International Journal of Operations & Production Management*, 15(4):80–116, 1995.

[32] Joaqun Nicols and Ambrosio Toval. On the generation of requirements specifications from software engineering models: A systematic literature review. *Information and Software Technology*, 51(9):1291 – 1307, 2009.

[33] Srinivas Nidhra and Jagruthi Dondeti. Blackbox and whitebox testing techniques-a literature review. *International Journal of Embedded Systems and Applications (IJESA)*, 2(2):29–50, 2012.

[34] Ken Peffers, Tuure Tuunanen, Marcus A. Rothenberger, and Samir Chatterjee. A design science research methodology for information systems research. *Journal of Management Information Systems*, 24(3):45–77, 2007.

[35] John Sören Pettersson, Simone Fischer-Hubner, Marco Casassa Mont, and Siani Pearson. How ordinary internet users can have a chance to influence privacy policies. In *Proceedings of the 4th Nordic Conference on Human-computer Interaction: Changing Roles*, NordiCHI '06, pages 473–476, New York, NY, USA, 2006. ACM.

[36] Erik Reimer. Privacy architecture idmeglers impact. Technical report, NTNU, 2015.

[37] Forrest Shull, Janice Singer, and Dag IK Sjøberg. *Guide to advanced empirical software engineering*, volume 93. Springer, London, 2008.

[38] Carl-Fredrik Sørensen. Id megler. Memo, 2010.

[39] J. M. Such, E. Serrano, V. Botti, and A. García-Fornes. Strategic pseudonym change in agent-based e-commerce. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 3*, AAMAS '12, pages 1377–1378, Richland, SC, 2012. International Foundation for Autonomous Agents and Multiagent Systems.

[40] Jaideep Vaidya. Privacy in the context of digital government. In *Proceedings of the 13th Annual International Conference on Digital Government Research*, dg.o '12, pages 302–303, New York, NY, USA, 2012. ACM.

Appendix

Questionnaire Questions Appendix A

The questionnaire was written in Norwegian and included the following questions with the following alternatives.

- Hvor mange banker har du hatt, eller har du et kundeforhold til? (0, 1, 2, 3, 4, annet)
- Hvilke forsikringsselskap bruker du, eller har brukt? (SpareBank 1 Forsikring, Gjensidige, NAF, If, Tryg, Storebrand, annet)
- Er du medlem av noen fagforeniger? (ingen, Tekna, Nito, LO, annet)
- Er du medlem av Obos, Tobb, eller andre lignende Boligbyggelag? (ingen, Tobb, Obos, annet)
- Hvilke sosiale medier burker du? (Twitter, Facebook, Instagram, Tumblr, Google+, Snapchat, LinkdIn)
- Er du eller har du vært medlem av en studentsamskipnad? (Jeg er medlem, Jeg har vært medlem, nei)
- Har du hatt korospondans med noen statlige etater? (ingen, Lånekassen, Militæret, Nav, annet)
- Hvor mange idrettslag/treningssentre er du medlem av? (0, 1, 2, 3, 4, annet)
- Har du korrespondert med noen veldedige organisasjoner? (ingen, Leger uten grenser, Kirkens nødhjelp, Røde kors, SOS barnebyer, Plan Norge, annet)
- Er du medlem av et trossamfunn? (ja, nei)
- Bruker du paypal? (ja, nei)
- Bruker du Vips eller mCASH? (ingen, Vips, mCASH)
- Har du benyttet deg av nettbutikker? (ingen, Finn, NettOnNett, Komplett, Ebay, Amazon, annet)
- Finnes det andre organisasjoner du er eller har vært medlem av som har lagret informasjon om deg elektronisk? Rams opp så mange som mulig, fordelt på er og har. (fritekst)
- Kjønn (mann, kvinne)

- Alder (0-19, 20-29, 30-50, 50+)
- Høyeste utdanning (Videregående, Fagbrev, Bachelor, Master, Phd, annet)
- Hva slags smartphone har du? (Iphone, Android, Windows phone, jeg har ikke smartphone)