Implementing a telemedicine system for remote medical consultation

SIMON ENLUND

ALEXANDER JOURKOVSKI
Abstract

Telemedicine is the combination of information technology (IT) and health care and is an area that has seen increase during the last few decades due to breakthroughs in IT. Today there are still parts of the health care system that can be counterproductive with regards to the time and commitment necessary. One such part is medical consultations for private individuals where many of the simpler ones are still carried out in-person.

Presented in this thesis is the implementation of a prototype telemedicine system that could be used as a substitute for in-person doctor-patient consultation. Necessary 3rd party components and their implementations into one system are described on a general use level. The result is a web application that supports login and payment via trusted sources, booking of meetings and carrying them out with video and audio.

The prototype is a continuation of a telemedicine web application idea that was put on ice by a company that already provides solutions for health controls and blood analysis.

Keywords
Telemedicine, Distributed System, Web Application, Java EE
Abstract

Telemedicin är kombinationen av informationsteknik (IT) och hälsovård och är ett område som har sett en ökning de senaste årtiondena på grund av genombrott inom IT. Idag finns det fortfarande delar av hälsovården som kan vara kontraproduktiva med avseende på den tid och det åtagandet som krävs. En sådan del är medicinska konsultationer för privatpersoner där många av de simplare mötena fortfarande sker i person.


Den presenterade prototypen är en fortsättning på en telemedicinsk webbapplikationsidé som lagts på is av ett företag som redan erbjuder lösningar för hälsokontroll och blodanalys.

Nyckelord
Telemedicin, Distribuerat System, Webbapplikation, Java EE
# Table of Contents

1 Introduction ........................................................................................................................................ 3  
   1.1 Background ................................................................................................................................ 3  
   1.2 Problem ....................................................................................................................................... 4  
   1.3 Purpose ........................................................................................................................................ 4  
   1.4 Goal ............................................................................................................................................. 5  
       1.4.1 Benefits, Ethics and Sustainability .......................................................................................... 5  
1.5 Methodology and Methods ............................................................................................................... 6  
       1.5.1 Quantitative vs Qualitative ..................................................................................................... 6  
       1.5.2 Philosophical assumption ........................................................................................................ 6  
       1.5.3 Research method ..................................................................................................................... 7  
       1.5.4 Research approach .................................................................................................................. 7  
1.6 Delimitations ................................................................................................................................... 7  
1.7 Outline ........................................................................................................................................... 8  

2 Technical Background ....................................................................................................................... 9  
   2.1 Java Enterprise Edition (Java EE) .................................................................................................. 9  
       2.1.1 Architecture ........................................................................................................................... 9  
       2.1.2 Containers ............................................................................................................................... 10  
       2.1.3 Beans ..................................................................................................................................... 11  
       2.1.4 Servlet .................................................................................................................................... 11  
       2.1.5 JavaServer Pages ..................................................................................................................... 11  
       2.1.6 JavaServer Faces ..................................................................................................................... 12  
       2.1.7 Context and Dependency Injection ......................................................................................... 13  
       2.1.8 Bean Validation ...................................................................................................................... 13  
   2.2 OpenTok ....................................................................................................................................... 13  
       2.2.1 Sessions .................................................................................................................................... 13  
       2.2.2 Routing .................................................................................................................................... 13  
       2.2.3 Tokens ..................................................................................................................................... 14  
       2.2.4 Publishing ............................................................................................................................. 14  
       2.2.5 Subscribing ............................................................................................................................. 14  
   2.3 Security ....................................................................................................................................... 15  
       2.3.1 Databases ............................................................................................................................... 15  
       2.3.2 Authentication ......................................................................................................................... 16  
       2.3.3 Authorization .......................................................................................................................... 17  
       2.3.4 Encryption .............................................................................................................................. 17  
       2.3.5 Web ....................................................................................................................................... 17  

3 Methods ........................................................................................................................................... 22  
   3.1 Research Strategy ......................................................................................................................... 22  
   3.2 Data Collection ............................................................................................................................. 22  
   3.3 Data Analysis ............................................................................................................................... 22  
   3.4 Quality Assurance ...................................................................................................................... 23  
   3.5 Development Method .................................................................................................................. 24  
       3.5.1 Scrum (Agile) .......................................................................................................................... 24  

4 Implementation ................................................................................................................................... 26  
   4.1 Overview ...................................................................................................................................... 26  
   4.2 Login .......................................................................................................................................... 27  
   4.3 Booking ...................................................................................................................................... 28  
   4.4 Payment ..................................................................................................................................... 29  
   4.5 Meeting ..................................................................................................................................... 31
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.6 Database</td>
<td>33</td>
</tr>
<tr>
<td>4.7 Security</td>
<td>33</td>
</tr>
<tr>
<td>5 Results</td>
<td>35</td>
</tr>
<tr>
<td>5.1 VideoPortal</td>
<td>35</td>
</tr>
<tr>
<td>5.2 Features</td>
<td>36</td>
</tr>
<tr>
<td>6 Conclusions</td>
<td>37</td>
</tr>
<tr>
<td>6.1 Summary</td>
<td>37</td>
</tr>
<tr>
<td>6.2 Discussion</td>
<td>37</td>
</tr>
<tr>
<td>6.3 Future work</td>
<td>39</td>
</tr>
</tbody>
</table>
1 Introduction

Healthcare is a field that has existed for a long time because there has always been a need for helping others with their physical and mental symptoms. Throughout the history of healthcare and medicine a number of obstacles have hindered their ease of use. One of the biggest of these obstacles is the ability to present and transport medicinal information. By solving this issue one can help and provide aid over distances.

One of the first reported cases of a crossroad between information technology and medicinal care was in 1906 when Wilhelm Einthoven managed to transport electrocardiograms via a closed-circuit television network with the use of his “telecardiogram”.[1] This allowed an electrocardiogram to be recorded at a distance from the patient. Another case is “The Nebraska Project” in the 1950s when doctors performed interviews with psychiatric patients located in other facilities up to 150km away.[1]

Today we refer to this crossing of information technology and health care as telemedicine. Telemedicine is formally defined as “the use of medical information exchanged from one site to another via electronic communications to improve a patient’s clinical health status” according to the American Telemedicine Association.[2] Today we apply this area to more advanced systems. This can happen via a variety of applications or services including two-way video, email, smart phones, wireless tools and other forms of telecommunications technology.[2]

1.1 Background

Examinations require that both the patient and the doctor are present in one place. This is not always possible or feasible. Today we use two-way communication with video, audio and text over networks in multiple ways during our day to day lives. A subset of these physical or mental examinations are simple enough to be able to be completed using these methods as well.

Two-way communication over networks has several possible implementations. Two of the primary ones are peer-to-peer between clients and client-server with multiple clients. These are used in order for parties to transmit information between each other. With peer-to-peer the clients connect directly to each other to establish a connection and with a client-server model the clients connect to each other via a server that is not always visible to the user.[5] In order to implement these in bigger projects they are commonly used as a part of a distributed system.

A distributed system appears to its end users as a single and coherent system but is in reality an assortment of independent computers working together. These systems are often used in order to implement multiple back end services for a front-facing application or web service. Examples of popular back end additions are databases and authorization.[6]
The use of distributed calling systems in telemedicine for regular consultations has recently become prominent. There are a few products that have come out recently and this is why a push for these kinds of products has appeared. An example of such a product is 'kry.se' but the problem with these products in regards to this thesis is that they are closed and proprietary. This means that the technology and systems that they employ are not openly shared.

![Image of 'kry.se' front page](image)

**Figure 1. The front page of 'kry.se'.**

1.2 Problem
The problem tackled in this thesis is how to implement a distributed telemedicine system for use as a substitute for in-person doctor-patient examinations.

Most of the questions regarding necessary features and graphical design were already decided on or were irrelevant for the current stage of the project. This thesis is about the continuation of the foundation laid by the original specifications.

The question to be answered in this thesis is: how could a telemedicine system enable remote medical consultations and how would that technical solution be implemented?

1.3 Purpose
The purpose is to research and present an implementation of a prototype telemedicine system using a distributed architecture and to describe how it
was implemented. It will be explained on a general level to provide a clear view of the entire prototype and what components were necessary.

1.4 Goal
The intended goal for this thesis is a proof of concept prototype of a distributed telemedicine system that customers and medical personnel can use for remote consultation via a web interface. The system should be able to support a robust login, preferably via a known or trusted source. It should also provide means to arrange or book meetings between the two parties and a way to do them via both video and audio. Customers should be able to pay for their planned meetings online via a trusted source and have a quick overview of them. All the information both sent and stored should be handled securely in order to conform to set standards. Initially it should at least try to conform to 'Personuppgiftslagen' (The Personal Data Act) and 'Patientdatalagen' (The Patient Data Act).

1.4.1 Benefits, Ethics and Sustainability
Potential benefactors of this thesis are developers who are new to these areas or students who wish to further their knowledge in this or similar systems. This could lead to an increase in interest in telemedicine and in turn a potentially positive benefit to the field of healthcare.

One of the most obvious concerns with regards to ethics is the physician-patient privilege and medical data confidentiality.[4] These relate to the act of securing and encrypting information passed via the two involved parties to protect it from incorrect hands. This works for the associated physician in the same way it works today without telemedicine but information however also has to be passed through a series of networks and potentially stored for future use. In order to preserve the confidentiality the data needs to be properly secured.[7]

There is also the issue of these telemedicine systems' abilities to potentially diminish the trust and intimacy between patient and physician that one would otherwise get when being examined inside an official healthcare facility. Even the small act of sitting in the waiting room may grant an assurance before meeting a doctor. Instead you are anywhere outside of these facilities and then find yourself talking to a person via webcam which may not feel as personal.[8]

Consultations made through the system have to follow standard regulations regarding healthcare, licenses and necessary educations. This is easier to enforce in physical examinations due to them being in official healthcare facilities. There are workers that constantly take notice of the people working there, there are also continual identification checks and clockings.

When it comes to sustainability there are two prominent possible issues. The first is competition. The healthcare area has big monetary potential and there are many companies vying for them. There are and will always be competing products that will push themselves to be better or more appealing to
customers. This is the result of the competitive market. The other possible issue is for technology that the product is used with or for to become outdated or replaced. This applies to both the client-server machines but also to the components of the system.

1.5 Methodology and Methods

Before starting work on a thesis like this it is important and essential to recognize the different methods and methodologies to be used.[3] These can be decided on in retrospect but that would be detrimental to the study. These different methods are chosen suitably to the study’s subject matter. The different philosophical assumptions, research methods and research approaches that we will choose from are detailed below.

1.5.1 Quantitative vs Qualitative

Unless specified otherwise, the text in this chapter is based on [3].

The first choice to be made is a decision between research method categories. This will narrow down the choice of methods going forward.

- **Quantitative** – These methods require a larger number of data samples and focuses on quantity over quality in order to test and validate the project's hypothesis. This method is best used when measuring variables that can prove or disprove a chosen hypothesis.

- **Qualitative** – Contrary to the quantitative research methods the qualitative methods focus on a more narrow set of data that produce more reliable results. Instead of measuring numbers it emphasizes behaviors and opinions to develop computer systems or reach tentative theories.

Because of the nature of this thesis' project we will use qualitative research methods to find the data required to develop a prototype system.

1.5.2 Philosophical assumption

Unless specified otherwise, the text in this chapter is based on [3].

After selecting an appropriate research method category we can continue on to philosophical assumptions. These are decided on early in order to set the tone of the research.

- **Interpretivism** – This assumption tries to comprehend phenomena by investigating and trying to understand other peoples interpretations of them. It is most often used in projects that examine opinions, perspectives and experiences. These use cases include developing projects with necessary human-computer interaction.

Our choice for this thesis is interpretivism because of the need to interpret and understand other people's interpretations of systems, concepts, ideas and
components. This is a good fit when developing new software systems as most of the data will be found in reports, documents and books.

### 1.5.3 Research method

*Unless specified otherwise, the text in this chapter is based on [3]*.

An appropriate research method is also to be considered. This is the method that the actual research will be conducted with. These involve starting the research, carrying it out and completing it.

- **Conceptual** – The conceptual research method concerns itself with creating new concepts or working on existing concepts. Literature can be studied to interpret and find frequently used concepts with focus on theoretical data and not on realized objects or events.

Our choice for this thesis is the conceptual method since it is the most suitable for development of prototype systems.

### 1.5.4 Research approach

*Unless specified otherwise, the text in this chapter is based on [3].*

A final choice is made to be used for drawing conclusions from the resulting data and hopefully be able to decide on a definitive conclusion.

- **Inductive** – The inductive approach draws probable conclusions from analyzing empirical data in the form of observations and patterns. It is often used in conjunction with data gathered by qualitative methods to construct theories.

For this thesis there is no initial hypothesis and qualitative research methods have already been chosen. This leads to us using the inductive approach since the objective of this thesis is to discover what needs to be done to achieve the sought software system.

### 1.6 Delimitations

The project that this thesis is based on was supplied by a company that handles real customers and products. This means that certain details and concepts need to be followed. This includes certain features and ideas that the company want to be implemented and followed but nothing except core functionality was set in stone. This limits the amount of extra features that will be considered for the project due to the specification. There are also several areas and features that will need to be excluded due to the nature of the project and its goals as well as the possible time frame that needs to be followed.

Developers at the company had already tried to produce a prototype a long time ago which had since been put on ice. This led to the new project already having certain frameworks and features decided on, particularly the ones that
are already paid for or are supplied through existing partnerships. This excludes the need to find certain alternatives. There are also exclusions based on costs, specifically from third party components.

Delimitations also need to be placed on the research that is made before and during the project. The project and this resulting thesis are very technical in nature and as such there is not much room for researching personal opinions or biased points of views. The research is instead focused on specifications and documentations that seek to be impartial and are true to the material. They should also be very specific to the questions that arise and that need to answered instead of being very general and broad. This means that the information and data that is researched should be about the parts that are actually pertinent to the project and not irrelevant.

1.7 Outline

The chapters of the thesis, excluding the current one, will are described below.

• **Chapter 2: Technical Background** – This chapter will detail the theoretic background for the main systems and components that are used throughout the project and described in the thesis. This is the base for understanding how the project system will work and function.

• **Chapter 3: Methods** – This chapter will detail some of the methods that can or will be relevant to this thesis and the choices that were thought to be most fitting. These methods cover the research, data collection, analysis, quality assurance and development.

• **Chapter 4: Implementation** – This chapter will detail the implementation of the distributed system. The theoretic information gathered in chapter 2 will be discussed in a more coherent and practical manner.

• **Chapter 5: Results** – This chapter will detail the results that have formed from the problem and the questions stated in chapter 1.2.

• **Chapter 6: Conclusions** – This chapter will detail the conclusions gained from reflecting on the results of this thesis report and how it influences the information from the introductory chapter. Positives, drawbacks, evaluations of the degree project and even future work will be described.
2 Technical Background

This chapter will detail the technical background for the main systems and components that are used throughout the project and described in the thesis. This is the base for understanding how the project system will work and function.

2.1 Java Enterprise Edition (Java EE)

Java EE is an enterprise software platform designed to help with the creation of “large-scale, multi-tiered, scalable and secure network applications”.[9] Enterprise applications are applications for large enterprises with the intent of solving the problems that arise with working on such scales.[9]

Creating such applications often induces complexity both in the systems and during development. The Java EE platform aims to reduce this complexity by providing the necessary tools in order for the developer to be able to focus more on functionality.[6]

2.1.1 Architecture

Unless specified otherwise, the text in this chapter is based on [10].

In enterprise applications the functionalities are divided into their own tiers. These are usually separated into a client tier, a middle tier and a data tier. In Java these are in respective order; the Client Machine tier (Client tier), the Java EE Server tier (Web tier and Business tier) and the Database Server tier (Enterprise Information Systems tier).

![Figure 2. The four individual layers of a Java EE application.[10]](image)
• The client tier consists of dynamic web pages that contain various types of markup language and web browsers that render them.

• The web tier consists of either Servlets or web pages that are created with JSF or JSP.

• The business tier consists of enterprise beans that are running necessary business logic. Examples of these are banking, retail or finance.

• The Enterprise Information Systems tier consists of enterprise information systems that the Java EE web application might need access to.

2.1.2 Containers

Unless specified otherwise, the text in this chapter is based on [18].

There are four types of containers within the Java EE specification. These are the EJB Container, Web Container, Application Client Container and the Applet Container. Containers provide an executable environment that supports a subset of the APIs and specifications defined by the Java EE platform for the 'applications' deployed within.

![Diagram of the four containers according to the Java EE specifications.](image)

Figure 3. The four containers according to the Java EE specifications[18].

- EJB Containers manage the execution of Enterprise JavaBeans.
- Web Containers manage the execution of Servlets and web pages.
- Application Client Containers manage the execution of normal Java SE applications.
- Applet Containers manage the execution of applets.
The containers also handle several non-configurable services such as component life cycles, database connection resource pooling and access to APIs.

### 2.1.3 Beans

Java Beans are classes that are encapsulated into a single object. These are meant to be easy to reuse by others without understanding their inner workings. The variables in a bean are private and are reached with mutator methods and accessors. These are methods that are used to control and manipulate changes to variables, usually referred to as “setters” and “getters”. Beans also implement zero-argument constructors and are serializable.\(^{[19]}\)

There are also Enterprise Java Beans. These are implemented in the applications business layer and handles business logic. In order to accomplish these tasks they provide extra services that a regular bean does not. These extra services include transaction management and security authorization. An enterprise bean is either a session bean or a message-driven bean.\(^{[20]}\)

- A session bean performs a task for a client and when the client finishes the bean and its data are gone.\(^{[20]}\)
- A message-driven bean combines the functionality of a session bean and a message listener. This allows message-driven beans to receive asynchronous messages from business components.\(^{[20]}\)

### 2.1.4 Servlet

*Unless specified otherwise, the text in this chapter is based on [21].*

A servlet is a Java class web component used to generate dynamic web pages via a request-response programming model. These are managed by Servlet Containers as a part of a web server or application. A typical request-response may look as follows:

1. A web server receives a HTTP request from a client and it is handed off to the servlet container.
2. The Servlet container then sends objects that represent the request and response to the appropriate servlet.
3. The servlet looks at the request object received from the container to determine HTTP POST parameters and other relevant data. It then generates the correct data and sends it back to the requesting client via the response object.

### 2.1.5 JavaServer Pages

*Unless specified otherwise, the text in this chapter is based on [18].*

JavaServer Pages (JSP) is used to generate dynamic web pages. The JSP pages are text-based documents that can contain both static data and JSP elements. The static data is usually in HTML or XML and the JSP elements determine
how the page constructs the dynamic content. This allows developers to write Java code directly inside HTML code.

2.1.6 JavaServer Faces

Unless specified otherwise, the text in this chapter is based on [23].

JavaServer Faces (JSF) simplifies development of Java web applications by providing a component-based MVC (Model-View-Controller) framework. As a result JSF applications become more manageable because the user-interface code (View) and the applications data and logic (Model) is separated. Front end “Faces” Servlets (Controller) handle all the user interaction to thwart improper or unauthorized access of the pages. [22]

JSF pages (XHTML) are built using tags which either associate with translations from JSF tags to HTML tags or by directly using the HTML tags. The JSF tags have the additional functionality of being able to communicate and handle the contents of managed beans. This allows the server to store information received and managed by the front facing page. Unlike the hands-on approach to request-response with Servlets in JSP, the servlets in JSF act unseen to the developer and development time can be spent more efficiently.

Figure 4. The phases of a JSF request. [23]

- **Restore View**: Retrieves an older component tree if the page has been displayed before. If it has not then the component tree is created. If the HTTP parameters are missing then JSF skips to the Render Response phase.
- **Apply Request Values**: Parameter values are stored for the components that they belong to by identifying them individually.
- **Process Validations**: The store parameter values are checked by the validation rules. These are set in a managed bean and if any of the validations fail then the previous page is sent again but with an additional error message.
- **Update Model Values**: Local values in managed beans are updated by using the setter methods.
- **Invoke Application**: The method specified in the action attribute of the appropriate HTTP request component is called.
• **Render Response**: The JSF components are translated to regular XHTML code and appropriate values are received from the get methods for the relevant managed bean.

### 2.1.7 Context and Dependency Injection

*Unless specified otherwise, the text in this chapter is based on [23].*

In order to manage the life cycles of objects and their relations with each other Java EE incorporates the CDI container. The result of this is not having to create and discard objects at the correct times and the ability to intercept calls between them.

### 2.1.8 Bean Validation

*Unless specified otherwise, the text in this chapter is based on [23].*

An important part of any application with input and outputs is to validate the data that is passed. These validations are used in multiple layers and run the risk of storing duplicate code for the same validation logics. This can be circumvented by using Bean Validation. Bean Validation is a standard framework used for validation in Java EE. All field, methods parameters and return values can be validated using specific annotations or in XML configuration files. The validator sets can then be shared by all the layers of an application.

### 2.2 OpenTok

OpenTok is a WebRTC (Web Real-Time Communication) platform for embedding live video, voice and messaging into websites and mobile apps.[11] The OpenTok platform consists of two parts: client-side libraries and server SDKs.[11] These are available in several different programming languages. For the thesis project the JavaScript client-side library and the Java server SDK were used to implement the real-time communication between users.

#### 2.2.1 Sessions

*Unless specified otherwise, the text in this chapter is based on [12].*

OpenTok Sessions can be viewed as virtual lobbies. These are the hubs that clients connect to in order for communication to be established. Once a session has been created it is assigned a unique ID and will never expire. These IDs are then used to join specific sessions together with other specified users.

#### 2.2.2 Routing

*Unless specified otherwise, the text in this chapter is based on [13].*

There are three different ways for the media streams to be sent between users. The first is Relayed (Peer-to-Peer) sessions where the streams are sent directly between users. The second is a variant of the Relayed sessions where communication directly between users is not available and a relay server is
used as an intermediator. The third is Routed sessions where streams are explicitly sent via OpenTok Media Servers which enables features like archiving but sacrifices latency.

2.2.3 Tokens

*Unless specified otherwise, the text in this chapter is based on [14].*

OpenTok Tokens are used in conjunction with specific session IDs in order to connect to the sessions. Each user is assigned a unique token with a specified user role and the token needs to be supplied in order to be authenticated. These tokens are themselves unique for one session. If a session is viewed as a lobby then the tokens are the individual invitations that the users need in order to enter. Unlike session IDs the tokens expire.

2.2.4 Publishing

*Unless specified otherwise, the text in this chapter is based on [15].*

In order to send a stream of video or audio through a session you need to publish the stream to the session. This is done by passing a Publisher object as a parameter to the publish method. When this is done the session will identify and allow the user to stream their data to the session and others can see it by subscribing.

2.2.5 Subscribing

*Unless specified otherwise, the text in this chapter is based on [16].*

When a user publishes a stream to the session, the other users will be notified and receive a “streamCreated” event. The users can then choose to subscribe.
to the published stream and begin to receive the data. In short publishers send
streams of content and subscribers receive it.

A user can both publish and subscribe which is often the case if two people are
video conferencing with each other. They both want to see and communicate
with one another.

2.3 Security
A vital and important part of any software system is its ability to protect itself
from attacks. This is done to protect users, systems and the business.

2.3.1 Databases
Unless specified otherwise, the text in this chapter is based on [26].

Almost every application deployed by companies today needs to store relevant
data. This is data about users, processes and variables that are relevant to the
application or company. This might be sensitive or confidential data, both
from a user protection point of view but also a business point of view. The
most common of these are interfaced via relational database management
systems such as SQL (Structured Query Language).

The most common vulnerability against these kinds of databases are so called
injections. An SQL injection is an attempt to run SQL statements on the server
via user input fields. This could be manifested through a username or
password input during login. These vulnerabilities are possible due to the
input from the fields being concatenated into the server’s own SQL
statements.
An example of such an attack would be a query on the server that looks as follows:

```
SELECT * FROM Users WHERE UserID = <input> AND password = <input>);
```

In this obviously vulnerable scenario the inputs from the login form is placed directly into the UserID and password comparisons which could lead to the following if a user would supply “190 or 1=1--” as input for their username:

```
SELECT * FROM Users WHERE UserID = 190 or 1=1;
```

This would result in the comparison “1=1” which is always true to make the rest of the statement true and the query would present all the rows from the table Users.

The more common countermeasures to these kinds of attacks are parametrization, sanitation and permissions.

- **Parametrization** handles user input in a different manner than normal string concatenation. The user input is treated literally and not as part of the actual SQL statement that will be executed. These inputs are added to the SQL query at execution time in a controlled manner.

- Sanitation is used to validated and check the input that the user supplies before it is added to an SQL statement. It will try to strip away anything that can be interpreted as SQL and not regular user input.

- Permissions limit what can be accessed in the database in the case of an attack being successfully completed. These permissions are set on the SQL users that run the command.

### 2.3.2 Authentication

*Unless specified otherwise, the text in this chapter is based on [27].*

Authentication is the process of identifying individuals. This is usually done via usernames and password and tries to ensure that the person logged in is who they claim to be. These can be done through self-implemented methods but they will take a lot of time and produce significant security risks. A better alternative is to make use of 3rd party authentications services. These are already established and have gone through years of development and testing.

The chosen authentication method for the project in this thesis is BankID which is the leading electronic identification in Sweden. This is done by sending requests to a BankID provider that then validates the user and redirects them back. There is also an authentication stage during purchase of time slots in the system via PayEx that handles the payment process. PayEx are authorized by Finansinspektionen (Financial Supervisory Authority) in Sweden.
2.3.3 Authorization

Authorization is the process of granting or denying access to resources or features. This is usually done in tandem with authentication to form a two-step process. Once a user is authenticated they will be granted appropriate permissions via authorization to use parts of the system.

In the system some users will be given roles of users, doctors or admins. This is done by authorization in the web application via the Java EE platform.

2.3.4 Encryption

Encryption is the process of transforming data to make it illegible or unavailable to unauthorized access. This is done to protect the confidentiality of digital data that is stored on computer systems or transmitted via the internet and other computer networks. Encryption is a popular topic of discussion because it is practically mandatory in almost all computer systems and as such there are a lot of resources, algorithms and programs devoted to it.

Today most encryption is already done through the components used in software systems such as the databases. Additionally there are also encryption options available for communication on the web layer such as SSL (Secure Sockets Layer) which is the standard for establishing encrypted links between web servers and browsers.

2.3.5 Web

There are several security risks for attacks in web-based applications. These are used to acquire confidential information, trick users to enter confidential information or dupe users. These attacks are made possible in the clients web browser. Two of the most widely known attacks are XSS (Cross-Site Scripting) and CSRF (Cross-Site Request Forgery).

XSS are injection attacks made on web applications that are otherwise trusted. Potentially harmful or malicious code is sent from a web application usually in the form of JavaScript which is then executed in the clients browser. These are made possible through inputs from clients that in turn are used in outputs back to clients and contain scripts that the receiving web browser does not have a way of detecting as malicious or foreign. There are three general types of XSS: Stored, Reflected and DOM-based.\textsuperscript{[29]}

**Stored XSS** are attacks where client input has been permanently stored on a target web application’s server. These inputs are usually in the form of comments or messages from users. Such inputs are stored server-side and
then sent back in the dynamic content that the web application serves to users requesting the site.\[29\]

![Image of a Stored XSS attack](image)

**Figure 7. An illustration of a Stored XSS attack.**\[28\]

1. An attacker uses an input field on a website to inject a malicious string into the webserver's database.
2. A victim requests the page from the webserver
3. The page containing the malicious script retrieved from the database is sent to the victim.
4. The victim's browser executes the script contained in the response.

**Reflected XSS** are attacks where a user constructs a URL that contains a malicious string and sends it to potential victims. The server will then include the malicious string as a part of the response. It is not necessarily stored in a database since it is used in the requested site to dynamically refer to previous input perhaps only once.\[29\]
Figure 8. An illustration of a Reflected XSS attack.\[^{28}\]

1. An attacker constructs a malicious URL containing a script.
2. A victim is made to send a request with the URL to the webserver.
3. A page containing the malicious script is received from the webserver.
4. The victim’s browser executes the script contained in the response.

\textit{DOM-based XSS} are attacks which are similar to the reflected XSS attacks described previously. The difference lies in the way that the malicious string in the URL is handled by the server. In a reflected XSS attack the server will retrieve and handle the malicious string as it inserts it into the requested page. In a DOM-based XSS attack however the malicious string will instead be retrieved and executed in the clients own browser as a part of a legit script sent by the server.\[^{29}\]
1. An attacker constructs a malicious URL containing a script.
2. A victim is made to send a request with the URL to the webserver.
3. The webserver receives the request but does not include the malicious string.
4. The victim's browser executes the legitimate script from the server which then retrieves the malicious script and inserts it into the page.
5. The victim's browser executes the malicious script that it inserted into the page.

Since XSS attacks are code injections most of the protection available today is via secure input handling. For web applications these are done in two different ways. The first is encoding which escapes the user input which results in the browser interpreting it as data and not as code. The other way is with validation which filters the user input from unwanted data so that the browser can interpret it as harmless. Another method for handling risks in user input is to interpret the data in the context that it was added which would result in several tailor-made input handling scenarios. These would then be able to correctly sanitize or escape any malicious input for the specific cases.

The other widely known attack is CSRF (Cross-Site Request Forgery). A CSRF attack is done through a malicious website that assumes its victim is already authenticated to another website. The victim will visit the malicious website. The malicious site then routes a potentially harmful request to the target site from the victims browser via a hidden element in the page. The target site will have to assume that it is from the already authenticated user and then handle the potentially harmful request unbeknownst to the victim. The impact of such
an attack could be a request that is granted for the victim to transfer funds or send an involuntary form submission.\textsuperscript{[30]}

These vulnerabilities lie in the affected web application and not in the victim’s browser or the site that is hosting the CSRF attack. Aside from the user employing common knowledge such as always logging out of web applications when they are done there are also countermeasure that can be employed in the actual web application.\textsuperscript{[30]}

The web application can implement hard-to-guess tokens which need to be included in the user’s request. In order to work around this prevention method an attacker would have to get a hold of the victims own token and insert it into the request in the malicious website before sending it to the victim. These are used by JSF in the Java EE platform via ViewState in order to combat CSRF attacks.\textsuperscript{[30]}

Figure 10. An illustration of a DOM-based XSS attack.\textsuperscript{[31]}
3 Methods

This chapter will detail the methods that were chosen as the most appropriate to this thesis. These methods cover the research, data collection, analysis, quality assurance and development.

3.1 Research Strategy

Unless specified otherwise, the text in this chapter is based on [3].

The research strategies are considered guidelines for applying the actual research. In comparison to the methods in chapter 1.5, these are less about the philosophical approach and more about the research as it is carried out.

For qualitative research the most commonly used methods are Surveys, Case Study, Action, Exploratory, Grounded and Ethnography.

- Grounded Theory – An inductive theory discovery method that allows for development of a theoretical account of the featured topics. It is based in data and the systematic analysis of it.

For this thesis the most fitting option is grounded theory. This is because of the need to study existing data and systems in order to develop a new system derived from the information gained during the research.

3.2 Data Collection

Unless specified otherwise, the text in this chapter is based on [3].

The data collection methods are used in the early research stage to get the prerequisite and necessary data with the use of sampling.

For qualitative research the most commonly used methods are Questionnaire, Case Study, Observations, Interviews and Language and Text.

- Language and Text – Used for translating the meanings in documents, texts and meanings.

The only apparent choice for this thesis is language and text. This is due to the nature of the project which necessitates for data to be collected from specifications, tutorials and documentation from and for existing systems.

3.3 Data Analysis

Unless specified otherwise, the text in this chapter is based on [3].

After data has been collected it needs to be analyzed. There are several different methods for inspecting and modeling data in an effort to better understand the collected information. This is done in order to be able to draw conclusions.
For qualitative research the most commonly used methods are Coding, Analytic Induction, Grounded Theory, Narrative Analysis, Hermeneutic and Semiotic.

- **Analytic Induction and Grounded Theory** – Iterative methods that alternate between collecting and analyzing data until no contradictions to the hypothesis remain. The result is a validated theory of the hypothesis and the grounded theory.

Due to most of the thesis work being focused around software development and systems research the choice for data analysis is analytic induction and grounded theory.

### 3.4 Quality Assurance

*Unless specified otherwise, the text in this chapter is based on [3].*

After results have been found through analyzing the collected data it still needs to be quality assured. This is done to make sure that it does not incorrectly represent or is incorrect.

For qualitative research validity, dependability, confirmability, transferability and ethics need to be applied and discussed.

- **Validity** – The instruments used in the quantitative research need to be verified that they are in fact measuring what they are supposed to and in qualitative research, the need to be validated is to make sure that the research was conducted following established rules.

- **Ethics** – Regardless if you use quantitative or qualitative research there is still the issue of ethics. The research needs to assure that it adheres to the principles of being moral and humane. This includes privacy of participants, not using coercion and respecting confidentiality.

- **Dependability** – The qualitative counterpart to reliability when ensuing results need to be consistent. It is the process of using auditing to evaluate the legitimacy of the conclusions.

- **Confirmability** – The research needs to have been done without any interfering personal bias potentially affecting the results.

- **Transferability** – The ability of the research to be passed on as a base for other researchers. This often mandates detailed descriptions of the work.

Due to this thesis’ focus on qualitative data and methods the required quality assurances are validity, dependability, confirmability, transferability and ethics.
3.5 Development Method

There are also choices to be made after the research has been completed with regards to the development stage. This is a well-discussed and researched area because of the potential impacts on development efficiency and their results. These are often chosen depending on the size and complexity of both the projects and the teams working on them. There are several methods available but only the one that was chosen for this thesis will be discussed.

The chosen development method for this thesis is based on Scrum with daily meetings and targeted iterations.

3.5.1 Scrum (Agile)

*Unless specified otherwise, the text in this chapter is based on [25].*

The Agile development method is becoming more and more prominent amongst development companies and teams. As the name implies it is supposed to be modeled to be done fast, quickly and limber. Requirements in Agile projects are assumed to be able to change and can be very fluid. This requires the development method to be very adaptive to these changes and to be able to handle them frequently. This is where the Agile method is supposed to shine.

Scrum is a subset of Agile and is distinguished from other agile processes by specific concepts and practices. The two most common processes employed in Scrum are the daily Scrum meetings and sprints.

![Figure 11. The stages of sprints in Scrum.](image)

- Scrum meeting – The scrums are done every day and discuss the work that has been done and the work that needs to be done. These meetings are led by a Scrum master who moderates and takes decisions on the project development. Goals and tasks of priority are distributed here.

- Sprint – These are the individual iterations on a set of backlog items that the team has committed to. The sprints are usually one to four weeks long. This is where the work is done on specific tasks such as functions or bugs.
The Agile development methods are suited for fluid projects where the requirements are not necessarily set in stone before development begins. It is good for handling unexpected changes to the product and when specific parts of the product need to be worked and planned for individually during the development process.
4 Implementation

This chapter will detail the implementation and system architecture of the distributed system. The theoretic information gathered in chapter 2 will be discussed in a more coherent and practical manner. Any irrelevant or unnecessary code will be omitted from the code snippets and replaced with “...”.

4.1 Overview

![Diagram of VideoPortal architecture](image)

Figure 12. Overview of VideoPortal architecture.

The system is comprised of three generalized component groups divided into the front end and the back end. These three generalized component groups are the Java EE platform components, the database server components and the OpenTok components.

System modeling is per the MVC (model-view-controller) pattern and communication between the front end and the back end is via either beans in the Java EE platform or through Ajax calls. The web pages are generated with the JSF framework using XHTML and the web application server is GlassFish Server 4.1.

The back end consists of a web application server that hosts the primary Java EE components, a database server running the relational database management system MySQL and the sessions/token handling of OpenTok.
The front end consists of front-facing web pages that displays the relevant information from the back end and the client browser that handles the streaming/joining with OpenTok.

4.2 Login

Authenticating to the prototype during login is done via BankID provided by a company called Signicat.

Figure 13. The BankID login page via Signicat.

There are two steps to the process: the initial redirect to Signicat BankID and the resulting request with SAML. Since this is an in-development prototype the system uses the pre-production system of Signicat and BankID.

Figure 14. The BankID login process with Signicat.
The first stage of login for a user is the redirect to Signicat for use of BankID authentication. This is done with a '.sendRedirect()' on the current context:

```java
((HttpServletResponse)
```

After the user has been redirected for BankID authentication, the next login step is retrieving the returning request and the SAML response. SAML (Security Assertion Markup Language) is used when exchanging authentication and authorization data between two parties.

```java
HttpServletRequest request = (HttpServletRequest)
context.getExternalContext().getRequest();

if (request != null) {
    String assertion = request.getParameter("SAMLResponse");

    SamlResponseData samlResponseData = samlFacade.readSamlResponse(assertion, url);

    if (samlResponseData != null) {
        String personalNumber = samlResponseData.getSubjectName();
        ...

        ((HttpServletResponse)
context.getExternalContext().getResponse()).sendRedirect("https://localhost:8181/videoportal/secure/start.xhtml");
    }
}
```

### 4.3 Booking

Booking of meetings is done via a simple calendar view. It is generated via a JSF component called Schedule from a UI framework for Java EE called PrimeFaces. A client may book a time using the calendar and specify the reason for the meeting.
Figure 15. The booking page.

The implementation of such a calendar JSF component is done in the XHTML on the back end after adding the framework as a dependency and can look as below:

```xml
<p:schedule id="schedule"
    value="#{bookingView.eventModel}"
    widgetVar="myschedule"
    clientTimeZone="local"
    locale="se"
    slotMinutes="20"
    draggable="false"
    view="agendaWeek"
    rightHeaderTemplate="agendaWeek,agendaDay"
    axisFormat="HH:mm"
    timeFormat="HH:mm">
    <p:ajax event="eventSelect" listener="#{bookingView.onEventSelect}" update="eventDetails" oncomplete="PF('eventDialog').show();" />
    <p:ajax event="eventMove" listener="#{bookingView.onEventMove}" update="messages" />
    <p:ajax event="eventResize" listener="#{bookingView.onEventResize}" update="messages" />
</p:schedule>

4.4 Payment

Payment is done via the PayEx api for merchants. It is handled via a sequence of requests, responses and redirects. The actual payment page will look as below after the user has been redirected:
The actual system flow for handling payments in the system with PayEx works as below:

**Figure 16. The Credit Card payment page via PayEx.**

Payment begins when a user clicks on the payment button on the booking web page. This will trigger the actionListener below (step 1):

```html
<p:commandButton value="boka och betala" actionListener="#{bookingView.bookTime()}" oncomplete="PF('eventDialog').hide();"/>
```

The system will then check if the user already has standing credits before continuing with payment. If the user does not have credits remaining the schedule will reserve a time slot and a purchase will be started:

```java
if (bc.checkCredits(userid, 1)) {
    bc.bookTime(userid, schemaid);
}
updateSchedule();
purchaseStageOne();
```
In the method \texttt{purchaseStageOne()} the system will call a \texttt{paymentPartOne()} method from the class \texttt{BookingController} where the PayExOrder will be initialized (step 2) and then returned:

\begin{verbatim}
    payexOrder = bc.paymentPartOne(1, userSession.getUserId());
\end{verbatim}

\begin{verbatim}
    PayexOrder a = new PayexOrder(ipAddress, "sv-SE", em.find(Product.class, product.getId()), "SEK");
    ...
    if (a.initTransaction()) {
        ...
        return a;
    }
\end{verbatim}

When the PayexOrder object returns it will contain the correct redirect URL that the user needs to be sent to in order to pay for the corresponding transaction that had already been initialized at PayEx in step 2 (step 3):

\begin{verbatim}
    FacesContext.getCurrentInstance().getExternalContext().redirect(payexOrder.getRedirectUrl());
\end{verbatim}

The user will be redirected to PayEx and the correct return URL will be attached to the order so PayEx can redirect back accordingly. In our case the return URL will be 'bookingVerification.xhtml' (step 4). On this web page the user (now a paying customer) will be presented with information about the purchase (the receipt). Before the entire page is rendered the system will do a final request to PayEx (step 5) by completing the order:

\begin{verbatim}
    Orderdata od = bc.complete(payexOrder, userSession.getUserId());
\end{verbatim}

Now the page will be fully rendered containing the information received from the response from step 5 and the user will be presented it (step 6).

\section*{4.5 Meeting}

The actual meetings with video and audio in the system are done via the front end with JavaScript, but before a client can start a call in their browser the back end needs to have generated the necessary sessions and tokens for the meeting as described in chapter 2.2.
The OpenTok session IDs are generated when a meeting is first booked. This happens in the BookingController.bookTime() function that is described in chapter 4.3. The session ID is generated as follows and then stored in the database for later when the meeting actually starts:

```java
Session session = createSession();
meet.setSessionID(session.getSessionId());
em.merge(meet);
```

Now that the session ID is stored it can later be used when the meeting actually starts. When a meeting is started and the starting time is eligible then the necessary tokens for the users are generated with publishing capabilities. The call is done in the view:

```java
token = vc.generateToken(sessionID);
```

The generateToken() method exists in the controller:

```java
public String generateToken(String sessionID) throws OpenTokException {
    OpenTok opentok = new OpenTok(API_KEY, API_SECRET);
    TokenOptions tokenOpts = new TokenOptions.Builder().role(Role.PUBLISHER).build();
    String token = opentok.generateToken(sessionID, tokenOpts)
    return token;
}
```

Once these values are available (session ID and token IDs) the meeting can be started in the client’s browser on the front end. The OpenTok client code for the meeting is done with JavaScript. To connect to an OpenTok session (a
meeting) browser must initialize a session object with the correct ID and then connect, subscribe and publish:

```javascript
var publisher = OT.initPublisher("myPublisherDiv");
session.connect(token, function (error) {
    session.publish(publisher, "myPublisherDiv", properties);
});
session.on('streamCreated', function (event) {
    session.subscribe(event.stream, "subscribersDiv", properties);
});
```

Once the session has been initialized, connected to, subscribed and published to the users will actually be able to see and interact with each other.

### 4.6 Database
The database is a SQL relational database running on MySQL Server 5.7.11 that is used to store permanent and necessary data such as information pertaining the users, orders and meetings.

![Figure 19. Diagram of the database model.](image)

### 4.7 Security
The system does not currently have full implementation of proper security. Instead the basic security problem areas have been given general remedies and more advanced security measures would have been implemented further down the development stage.
Regarding XSS and CSRF attacks mentioned in chapter 2.3.5 there are some simple countermeasures set in place in the system. These are built into the Java EE platform via the way of JSF as described in chapter 2.3.5. This will safeguard against most simpler attacks.

SSL is activated for a few select web pages that require encrypted traffic. These are the web pages that pass potentially sensitive information such as the booking page and the actual meeting page. This is done via entries in the web.xml file of the web application. The specific web pages that require SSL are found in the folder named 'secure' and they are then referenced in the web.xml and appointed as requiring 'CONFIDENTIAL' transportation:

```xml
<security-constraint>
  <display-name>UserConstraint</display-name>
  <web-resource-collection>
    <web-resource-name>User</web-resource-name>
    <url-pattern>/secure/*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <description>Users can access this resource</description>
    <role-name>User</role-name>
  </auth-constraint>
  <user-data-constraint>
    <description>Always use HTTPS</description>
    <transport-guarantee>CONFIDENTIAL</transport-guarantee>
  </user-data-constraint>
</security-constraint>
```

In the back end the database should be running on an internal socket and have access disabled via network resources. Further all the SQL queries are parametrized in order to avoid common SQL injections. This is as described in chapter 2.3.1. See below for an example of this:

```java
Integer userId = user.getId();
Query q = em.createNativeQuery("SELECT g.group FROM useringroup g WHERE g.user = ?1");
q.setParameter(1, userId);
```

The security implemented in the current system is meant to stop and discourage the easier and more common attacks. Any highly invested or experienced attacker would be able to find flaws in almost any system and implementing all necessary countermeasures would take a significant time investment. Since this is a prototype such an investment has not yet been made.
5 Results

This chapter will detail the results that have formed from the problem and the questions stated in chapter 1.2.

5.1 VideoPortal

The resulting system for this thesis has the working name VideoPortal. It is a distributed telemedicine system with video and audio calling capabilities for remote medical consultations. The current stage of the system is a simpler prototype that works as a proof of concept to reinforce the information given in this thesis and is far from a finished product.

Figure 20. The starting page.

A typical user flow begins with the front page. The front page presents the user with a prompt to login via BankID or mobile BankID. After the user has been authenticated with their BankID they will be directed to the booking page. Here the user can see all vacant time slots, the soonest available time and any meetings they have already booked. From here the user can start a booked meeting or book a new meeting and be directed to the payment page. Payment is done quickly through PayEx and then the user is sent a confirmation.
### 5.2 Features

Current general features of the VideoPortal system:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Implemented</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Login</td>
<td>Yes</td>
<td>Login via BankID redirect.</td>
</tr>
<tr>
<td>Doctor Login</td>
<td>No</td>
<td>Specific login for doctors that authenticates their profession.</td>
</tr>
<tr>
<td>Meeting</td>
<td>Yes</td>
<td>Booking and attending meetings with comments.</td>
</tr>
<tr>
<td>Payment</td>
<td>Yes</td>
<td>Payment via PayEx.</td>
</tr>
<tr>
<td>Video/Audio</td>
<td>Yes</td>
<td>Video and audio for meetings via OpenTok with comments.</td>
</tr>
<tr>
<td>Roles</td>
<td>Yes</td>
<td>Roles for users, doctors and admins.</td>
</tr>
<tr>
<td>Database</td>
<td>Yes</td>
<td>MySQL database that stores necessary information.</td>
</tr>
<tr>
<td>The Personal Data Act</td>
<td>(Yes)</td>
<td>The Personal Data Act aims to prevent the violation of personal integrity in the processing of personal data. [33]</td>
</tr>
<tr>
<td>The Patient Data Act</td>
<td>Partially</td>
<td>The Patient Data Act stipulates that medical records must be kept for all patient care. The information is classified, and the Patient Data Act has strict regulations on how the information may be used by the health care providers’ staff. [34]</td>
</tr>
</tbody>
</table>

*Table 1. General features of the VideoPortal system.*
6 Conclusions

This chapter will detail the conclusions gained from reflecting on the results of this thesis report and how it influences the information from the introductory chapter. Positives, drawbacks, evaluations of the degree project and even future work will be described.

6.1 Summary

This thesis describes the implementation of a distributed prototype telemedicine system with the working name videoPortal. The system is capable of enabling simple remote consultation between patients and doctors with the help of video and audio on the web. The web application framework, the video/audio platform, the database system and other necessary chosen components are explained and so is the theory behind using them.

It is far from being a complete system. Instead it is a proof of concept prototype that shows that the theory in this thesis is well-founded and the resulting implementation is passable.

6.2 Discussion

The question to be answered in this thesis was: 'how could a telemedicine system enable remote medical examinations and how would that technical distributed solution be implemented?'.

The first part of the question pertains to what kind of a solution that would be viable. Using a web application to stream video and audio over networks proved to be a viable choice. This was introduced to the authors of this thesis via the specification from the requesting employer and later reinforced by user tests. It will allow doctors and patients to quickly connect and be able to provide and receive advice on medical matters instead of having to take the extra time and effort required to book and visit an in-person consultation.

The second part of the question was about how such a system could actually be implemented. What frameworks would be used, what components would be needed and how they would work together. The chosen answer to this question was a distributed telemedicine system using web application technology. The main chosen components were Java EE for the web application platform, OpenTok for the audiovisual streaming and MySQL for the back end database.

Java EE was a decent choice for the web application since it has the ability and the potential to fulfill all the necessary features of the prototype. It however was at times hard to work with. Development of Java EE began circa 18 years ago and as a result it has gone through many revisions, with many features being added, removed or forgotten. It is highly capable but introduces a layer of bloat to new practitioner. Understanding the differences between JSF, JSP,
Servlets and alternatives like Apache Velocity at a first glance and how they differ can be confusing.

There were also issues with the portability of the Java EE application server Glassfish. Getting it setup and running the web application on different development hosts introduced many smaller errors and clashes sequentially that broke functionality. It is also being deprecated and will soon no longer be supported with alternatives emerging such as WildFly.

Java EE does have an advantage to many because it uses Java. Java is a programming language that is taught to many students in today’s education. This is also true of the authors of this thesis which enabled faster development of back end features that are constructed with Java.

The OpenTok streaming however was fast to implement and easy to get running, at least in comparison to the enormity that was learning and using Java EE. A notable drawback to using OpenTok is that while their front end software is open to developers, their underlying software that powers their service is not. This means that certain modifications to how the software runs or works on a less basic level is not as easy and if such changes needed to be implemented an alternative would have to be found.

There were no direct problems with using and implementing MySQL as the database in the back end. It is widely used and has been in development with good standing for a long time which is due to its performance and usability. The actual choice of relational database manage software was not as picky as with the other components since they mostly work the same and are to a degree interchangeable. Every database however is not relational. There are valid opinions on using relational or non-relational database systems but for the prototype described in the thesis the specifications asked for a relational database system and the authors of this thesis already had experience with them.

Most of the system’s base functionality is present and implemented. Users can login and authenticate via BankID which will be a familiar process. They can then navigate a calendar view to see and book appointments and then start them. Paying for the appointments is done through PayEx which offers several payment methods. During these processes the top of the page displays a step-by-step banner showing an overview of where they are in the process. The system in its current state has shown that there is potential if worked upon further and that the idea behind the system is functionally sound.

Looking past the idea behind the system the current version of the prototype is very rough. The time spent working on it did not match the scale of the desired end product and it shows. This is something that several dedicated workers would need a long period of time to build from scratch and polish. Testing every possible component and use case, making sure that it complies to every necessary regulation and that there are no glaring issues. Even though
most of the functionality is there it does not have a proper integration for the authentication of doctors. They are not likely to use BankID as their primary login form but an alternative method that proves their legitimacy as certified medical personnel.

Security wise there are already some standards set in place. Most of it is through default or optional behavior in Java EE and MySQL. As mentioned in chapter 2.3.5 JSF already has built-in functionality to combat XSS and CSRF and SSL is forced on for the web pages that would require the traffic to be encrypted. General SQL injections are handled through the means mentioned in chapter 2.3.1 and other preventions made on the database include securing the server by blocking network access and managing correct user permissions. Further time will have to spent on implementing further and more robust countermeasures but the prototype didn't reach the stage where such a specific focus was legitimized.

The methods selected for this thesis proved sufficient and capable enough during the required work for the result of this thesis. This includes the preliminary methods of research and handling data as well as the development methods for the implementation.

Looking back at the work retrospectively the end result can be viewed as a slightly underwhelming success. It is far from a finished and production ready product, instead it is a prototype that will serve as a proof of concept and a base for continued work.

6.3 Future work

Future work on the prototype would be continuing the development of the system with the goal of reaching a finished product. Not only would additional time have to be spent on doing such work but the development would not stop after its launch. The end product would find its place in a new and competitive market with other vying companies trying to exploit its monetary potential as discussed in chapter 1.4.1. What this means is that there would be no monopoly and instead a chaotic market where constant improvement of the product would be necessary to stay on top of the market and be able to sustain a healthy and paying user base. All of this would require a significant future investment.

The project developed in this thesis was not meant to be immediately deployed. The goal was to continue on the earlier code base in order to produce a prototype version. As a result of this there will be no deployment stage to the public. Instead the project will be handed over to the original developer for either continued work or to be put on ice again.
References


