UiT THE ARCTIC UNIVERSITY OF NORWAY

Faculty of Science and Technology
Department of Computer Science

EDMON

A backend server for an infection detection system monitoring individuals with type 1 diabetes

—

Sverre Coucheron

INF-3981 Master's Thesis in Computer Science

To my family, friends and girlfriend;

thank you for all the support throughout the years.

“Writing a master’s thesis is hard.”
–Marius Wiik (30)

Preface

With infectious diseases being such a massive threat to the population, there is a great need for earlier disease detection. Together with this, individuals with type 1 diabetes also benefit a lot from self-monitoring and self-recording. Because of this, the EDMON project seemed like an excellent fit for me. To be able to help the project achieve its goals and build a platform to help both individuals and society, was to me, an honor.

The summer of 2018, I worked at DIPS, where I got an introduction to medical informatics and software development within the health sector. This led to a course at the University of Tromsø held by Gunnar Hartvigsen. The course had a strong focus on mobile health applications and the challenges within medical informatics. Both of these sparked an interest in the field and to create software to help, together with assisting users. After talking with Gunnar, it came clear that he was the founder of Tromsøstudentenes Dataforening, the student organization where I was currently the head of. We talked a lot about this, and to me, having him as my supervisor felt like a great match. We landed upon the EDMON project, where I wanted a back-end project.

First of all, I want to thank Gunnar Hartvigsen for all the support, guidance, and supervision throughout the project. He has provided me with knowledge not only in the project but helped me become a better computer scientist. I want to thank all of my co-supervisors too. Thank you to Ashenafi Zebene Woldaregay for providing me with continuous support and feedback and Miguel Tejedor for being there in all of my supervisions and all of his knowledge and feedback. A huge thank you to Eirik Årsand and Taxiarchis Botsis for all of their help and insight throughout the project.

A special thanks to my family for being there for me at all times - through thick and thin. They have given me continuous support throughout the years. Thank you for the opportunity you have provided me to chase a higher education, and for believing in me — and a special thank to my sister, Tina. From the time when she started school, she has provided me with help and tutoring. By forcing me to learn all that she had at school, it helped lead me to enjoy gaining more knowledge. She has also given me the support and encouragement in

all of these years, and a special thank you for helping me with my homework since the first grade.

Next, I would like to thank Guro Møller Ødegård, my girlfriend, for providing me with endless support and love. Thank you for sticking with me through the late nights and weekends I had to spend at the university. The endless help and support from you, helped me achieve the goal I had been chasing for the last 18 years - getting a masters degree.

I want to send a special thanks to the great company at the office. Thank you for a great semester with laughs, jokes, ribbing, and taunting. I will always cherish and look back at this time with love. Thank you to Marius Wiik for making me feel young and shaming me if I was 10 minutes late at 7 AM. To Valter Berg, whom I worked with last summer and had the pleasure of sharing an office with this semester. Thank you to Vebjørn Haugland for being a goof and his love for fossil cars. Tobias Borgen Olsen, thank you for talking so much about your glasses and how much they weigh, together with always being cheerful. Lastly, thank you to Andreas Isnes Nilsen for all the beer and wine, it has been a pleasure.

In truth, I could not have achieved my goal and finished my master thesis had it not been for the support from all of the people above. From the bottom of my heart, thank you all!

Tromsø, the 20th of May 2019
Sverre Coucheron

Abstract

There are a growing number of adults with diabetes worldwide. Within 2045 it is expected to become over 600 million individuals. Since there are no known cures for diabetes, self-monitoring and self-recording are often used to manage the condition. Having tools such as mobile applications allow individuals to do this. The world and society face a significant health threat from communicable diseases, which has resulted in a growth in the detection algorithms of infectious diseases.

This thesis proposes a back-end server with the functionality to implement disease surveillance algorithms on data from monitoring individuals with type 1 diabetes. It has a focus on standardization, security, and privacy. It also offers the opportunity for users to record themselves in a video with each medical recording. The design is devised with a modular approach to provide future scientists and researchers the possibility to extend the functionality.

Experiments and tests are conducted to see that the system is satisfactory and handles enough traffic for the task at hand. The solution handles 100 concurrent clients sending 10 000 requests, with around 800 requests per second. All testing is done on real user data, with calculations to simulate the EDMON infection detection systems performance. The server spends around 11 minutes running the algorithm on almost 3 million medical records, which is sufficient since this algorithm is meant to run once per hour.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Background and Motivations

There are around 425 million adults with diabetes worldwide, and this is expected to become 629 million by 2045. [1] Diabetes mellitus, often referred to as diabetes, is a collection of three types of diabetes; type 1, type 2 and gestational diabetes. The focus in this thesis is on diabetes type 1, which is where the body can not produce enough, or any at all, of the hormone insulin [1] [2].There are no known cures for diabetes, but with active surveillance, self-monitoring, together with treatment, the condition is manageable. The Internet of Things (IoT) has helped a lot in this aspect, paired with mobile health (mHealth). When one combines these to create a body area network (BAN) with sensors and wearables, it may be of great help for the individuals. It may create a solution for self-monitoring that could potentially save lives by helping the users to monitor their vitals. According to the Global Health Observatory [3] by the World Health Organization, [4] diabetes is the seventh highest cause of death yearly, as can be seen in Figure 1.1. Around 1.6 million deaths are directly attributed to diabetes each year [5]. To control diabetes type 1, one has to test blood values regularly [6] to inject with the right amount of insulin at the correct time. Often the individual has to follow a regular scheme of when to check blood values and take their long term insulin.

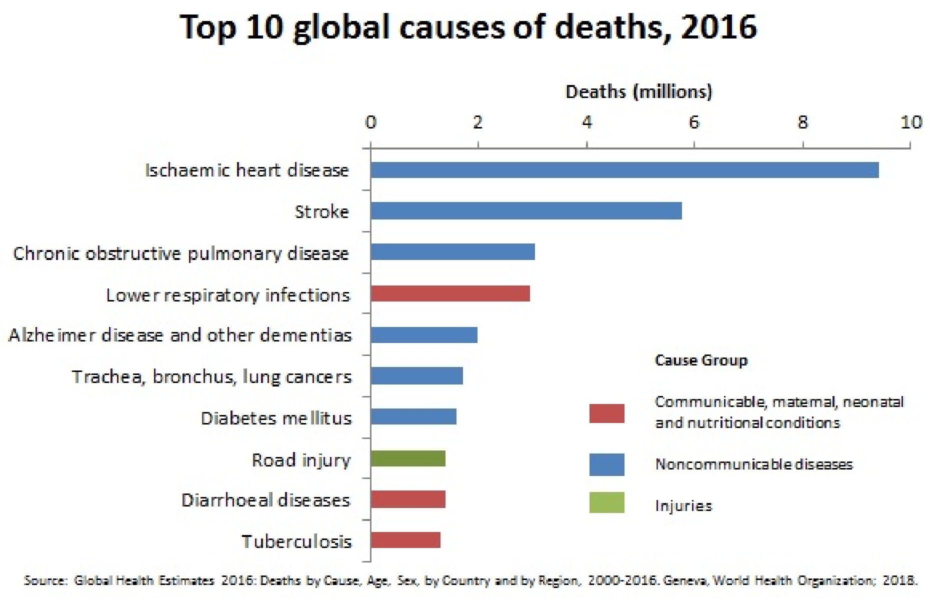


Figure 1.1: Top 10 causes of deaths in 2016

One can, therefore, assume that there is a great need for the individuals with diabetes to have a solution that can help them monitor their condition with ease, and give them a warning or a heads up that something may be wrong. The focus of this project is to create a cloud back-end solution which supports the further development of mobile applications for individuals with diabetes type 1.

This thesis is a part of EDMON - the Electronic Disease Monitoring Network [7] at the University of Tromsø. EDMON proposes a real-time early disease outbreak detection system by the use of self-monitoring and gathering data from individuals with diabetes type 1. The data undergoes micro event analyses, which detects infection induced elevated blood glucose pattern on an individual level, to reach on a conclusion for uncovering macro events in the world and on the general population [7]. As stated in the EDMON paper, the goal of the system is to create a mobile computer tier - a standalone mobile app that integrates the reading of the individual's essential diabetes and physiological parameters.

As communicable diseases are still a significant health threat for the population, there has been a growth in detection algorithms to try and detect this at an earlier stage [8]. Today there are a lot of data that can potentially give clues to when an outbreak of infectious disease has happened, and therefore it may be stopped at an earlier stage. There have been ways to do this in the past, but most of the solutions have had the issue of lag between the recording of the data, and when a general practitioner can make an assessment. For some time, solutions are developed, such as one by the New York City department for health and Mental Hygiene. They developed a syndromic surveillance system that uses data from different emergency departments and analyzed them electronically to try and detect disease outbreaks early. [9] In France, the same type of system is developed, with the name SurSaUD [10].

In the last years big data has been used in several aspects. Big data is used to describe the vast amount of data that is stored and how rapid the growth of the data is. As a result of the magnitude of data that is collected, researches started to use the data to try to gain knowledge [11]. It leads to standardization issues [12] [13] where the more complex infrastructure leads to several problems if there is not a standard in place to handle the data. There are organizations which are trying to help with this, such as Health Level 7 [14], which has been adopted to be the standard by many within healthcare. Another organization, and one of the most prevalent parts in disease surveillance, is the International Society for Disease Surveillance [15]. Their vision is to "work toward a timely, effective, and coordinated disease prevention and response among a skilled public health workforce through programs..." [16]. Standards aid and create opportunities for organizations such as ISID to work more efficiently towards their goals.

Within healthcare and storing sensitive data, problems arise when talking about privacy and security [17]. Healthcare in the US is a billion dollar industry, with millions of users [18], which leads to privacy and security of their confidentially health-related data to become a pressing issue. In Europe, the European Union tries to regulate laws regarding security and privacy for users, which helps to identify and fix issues. The General Data Protection Regulation recognizes the concern and defines clear rules around the issue [19]. In 2016, Norway defined how a health related system should be and which laws one has to obey to create an open system with data from the health sector. This reflects the ruling of Datatilsynet [20] and a national service which should make healthcare data available for responsible usage by third-parties with legitimate purposes [21].

1.2 Scope and Research Questions

The project started as a way to help solve the challenges and problems of individuals with type 1 diabetes. Both self-monitoring and identifying if their blood glucose levels are abnormal. By gathering their data and gaining a deeper understanding, we want to develop a disease outbreak system, which is the second challenge of this project.

The project is aimed to develop a general server-side solution which allows the developers of the project to use the back-end server for their applications. Further development should also help assist in creating microservices[22] and farther develop the functionality needed to aid individuals with type 1 diabetes. One of the goals for the project is to provide a permanent storage and interface for diabetes type 1 users to gather self-recorded data by self-monitoring.

The main research question of the thesis is:

Research question

"How can an electronic health system server to aid individuals with diabetes type 1, while creating a warning system of disease outbreak with security and privacy in mind, be designed?"

With this question, we try to identify sub-question which deal with the security and privacy aspects, together with having the expansion of the services components in mind.

Subquestion one

How can a system for disease surveillance be designed, and which drawbacks or advantages is there of the state of the art systems?

When designing a system for storing data and creating a warning system for disease surveillance, one has to look into the state of the art of the technology, together with how the current solutions chose to design their systems. It is, therefore, an important aspect of this thesis.

The design of the application meets the application developers, and users need and expectations. Aspects such as usability, security, availability, and privacy prevail as some of the essential choices when designing a project such as this. Therefore the second question is articulated as follows.

Subquestion two

How can secure data storage for mobile applications for individuals with diabetes be created?

Since this solution is meant to store the data for mobile applications, within healthcare, one has to ensure that the data transmitted is secure. Further research and a dive into the current solutions are also needed.

Subquestion three

How can user's authentication be ensured with the least amount of user-interaction in the front-end application?

As mentioned, the system is meant to be used, paired with mobile applications. Because of this, one has to take the users experience into consideration. It can not be too much of a hassle to log in to the system. The server solution should, consequently, ensure the authentication of a user to ensure privacy, together with making it as convenient as possible. An important aspect of login into applications is that the barrier is not too high. How this is done, when thinking about the design choices of the back-end server is one of the core questions.

Subquestion four

How can an electronic health system that is to be expanded by other researchers and students be designed?

This thesis is a part of the EDMON project thus it has to provide the researchers and students which will develop further on the solutions an easy way of expanding and understanding the system.

1.2.1 Assumptions and Limitations

The project has a focus on storing the data safely for the user while ensuring privacy. Since this is a building block for a larger project, there is also a focus on documentation and ease of further development. It has a focus on helping a limited target group: individuals with type 1 diabetes. According to Folkehelseintitutet, there are around 28.000 individuals in Norway in 2017 [23]. The disease detection algorithm is under development of someone else in the project.

Because of this, there are some limitations for the project, which leads to some assumptions before the project:

1. The detection algorithm may not be integrated into the server solution.

2. Since the project has a focus on the back-end solution for further development, there is no usability testing with a target audience.

3. Persuasive technology is not a focus of the thesis, but rather of the front-end application later in the project.

1.3 Methods

The thesis and the development work is done in the following order:

1. Review of relevant health systems with a focus on diabetes
2. Create requirements and an outline of the project
3. Developed a prototype using Golang's HTTP libraries
4. Designed and implemented database
5. Developed a new version of the system using GIN libraries
6. Created request using Postman [1]
7. Added support for JSON web tokens
8. Further development for https support
9. Performance testing and analysis of the results

As a starting point for the project, a state of the art review was conducted in the field to understand the current solutions. Together with this, the author had to outline the needs of the thesis for the EDMON project. Because of this, a lot of the design and development has had a focus on supporting further development and usage of the system.

Denning [24] suggest an engineering approach, and this was used to create requirements and as a tool to create prototypes that can solve the problems.

Throughout the project meetings with supervisors and experts within the field has helped shape the project and provided feedback for which direction the problem should go. There was also contact with the experts in the EDMON

1. Postman `https://www.getpostman.com/` (Visited 2019-05-16)

project to ensure that the solution would satisfy their needs.

1.4 Significance and Contribution

The projects' main contribution is to create a back-end server for mobile applications monitoring individuals with type 1 diabetes and using this data to create a real-time disease surveillance system. As an addition, the project has a focus on security and privacy, while giving the users a diary that is usable in daily life. This diary consists of short videos where each user can talk about their current feelings and well-being. The features in the project are defined by the state-of-the-art of the field, together with the requirements created and the EDMON requirements.

Another contribution of the system is that it should be a general enough solution that the project as a whole can use it in the coming years. Since several researchers and students are creating interesting thesis' where this project works as the foundation. The project will be used to collect data from real users, and the data can be analyzed further to help gain an understanding of how blood glucose levels, for example, can be used to identify if an individual has caught a disease. The system also supports extensions of the current solutions by creating a middleware that can add microservices to create more room for analysis, or satisfy future ideas.

1.5 The organization of the report

The organization of the thesis is as follows:

Chapter 2 - Theoretical framework
This chapter gives an overview of existing healthcare back-end solutions, cloud solutions, and similar technologies. It also explains more about diabetes and why this project can give a helping hand to its users, together with the data used as a tool to analyze and alert if there are infectious diseases. It also looks into existing technology in disease surveillance.

Chapter 3 - Methods and Materials
The chapter gives a brief understanding of which methods used under development, together with the materials used.

Chapter 4 - Requirements and Specification
The chapter looks into the project specifications and the functional and non-functional requirements, together with their sources.

Chapter 5 - Design
The design process as a whole is explained together with the design choices throughout the development of the project. There is also a history of how the server choices were made.

Chapter 6 - Implementation
The chapter looks into the chosen tools for developing a back-end server for a project within health care and reasons for choosing them. There are also code examples and an overview of the code-structure which allows for add-ons.

Chapter 7 - Test and Result
This chapter presents the testing and the results from the project in the form of latency and throughput.

Chapter 8 - Discussion
The chapter discusses the results of the test, together with a comparison of the project to existing technologies. There is also described some identified improvements to the project. Also contains the future work of the thesis.

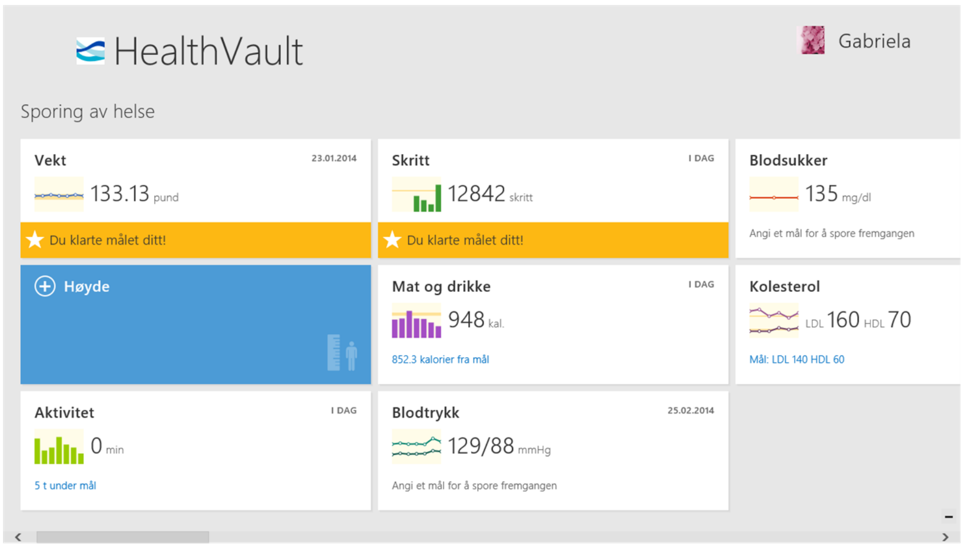
Chapter 9 - Concluding Remarks
Concluding remarks for the research and the contribution of the project.

2 Theoretical Framework

2.1 Health platforms

As the world has moved from health records to electronic health records (EHR) rapidly, several different approaches occur. From 2000 to 2011, every major industry invested heavily in computerization [25], and this does not seem to stop. DIPS, which has around 85% of market share in hospitals in Norway [26] is developing with OpenEHR as a focus. Because of this, their products "consists of open specifications, clinical models and software that can be used to create standards, and build information and interoperability solutions for healthcare" [27].

From a consumer perspective, there would be significant benefits to be able to gather relevant health data by self-monitoring and being able to utilize the open systems such as DIPS. Several vendors already create server-side solutions which users can collect data from a considerable amount of different electronic devices, such as smart-watches. By collecting data and integrating it to your general practitioner's system, one could get a diagnosis based upon the data you have continuously gathered.


HealthVault

Gabriela

Sporing av helse

Vekt 23.01.2014 133.13 pund

Du klarte målet ditt!

Skritt I DAG 12842 skritt

Du klarte målet ditt!

Blodsukker 135 mg/dl

Angi et mål for å spore fremgangen

Høyde

Mat og drikke I DAG 948 kal.

852.3 kalorier fra mål

Kolesterol LDL 160 HDL 70

Mål: LDL 140 HDL 60

Aktivitet I DAG 0 min

5 t under mål

Blodtrykk 25.02.2014 129/88 mmHg

Angi et mål for å spore fremgangen

Figure 2.1: Healthvault screenshots

HealthVault

HealthVault [28] is one of the suppliers of a system for self-monitoring and self-recording of data. It is a service run by Microsoft which integrates 24 applications and 188 different monitor devices such as smart-watches, according to their website [28]. It is also showcased that you can get more out of visiting your general practitioner since you can have an updated list of medicines you are on and allergy lists, combined with the self-recorded data. It offers applications for iPhone as well as Windows. In Figure 2.1 the Windows application is shown. The image fetched from the Microsoft store [1]

It also allows developers to create third-party applications which are compatible with their platform. This may be the reason why they can provide such a large number of devices. One of the reasons to use HealthVault as a developer is that they provide solutions for storage and authentication. It may lower the cost for the developer, together with creating new business opportunities [29].

Diabetes applications

While there are corporations like HealthVault that provide a way to store your data, there are also open-source projects such as Nightscout [30]. The thought of Nightscout is that everyone should be allowed to access their blood

1. Microsoft Store HealthVault `https://www.microsoft.com/nb-no/p/healthvault/9wzdncrfj3mc`

glucose levels via a personal website and that it uses an open source, do-it-yourself approach. This system is tailored to diabetes users for recording their data.

Nightscout is created by the parents of a 4-year-old boy that had to use a glucose monitoring system after being diagnosed with type 1 diabetes. Since there was no commercial option available, they created one for them. Their initiative has created a community where the focus is on creating a way to monitor their chronic condition by integrating hardware with their software. You can also use their code to upload data to the cloud by setting up a MongoDB database and using Azure as a web hosting service.

As an alternative to NightScout, one also has applications such as mySugr [31]. The application is available for both iOS and on Google Play and allows the user to record their data to the cloud. They claim to be loved by more than 1 million people with diabetes and offer blood glucose recording, estimated blood glucose, continuous glucose meter data, insulin calculations, and a personal diabetes coaching functionality. The functionality and how the application looks on Android as seen in Figure 2.2. The screenshots are from Google Play. 2

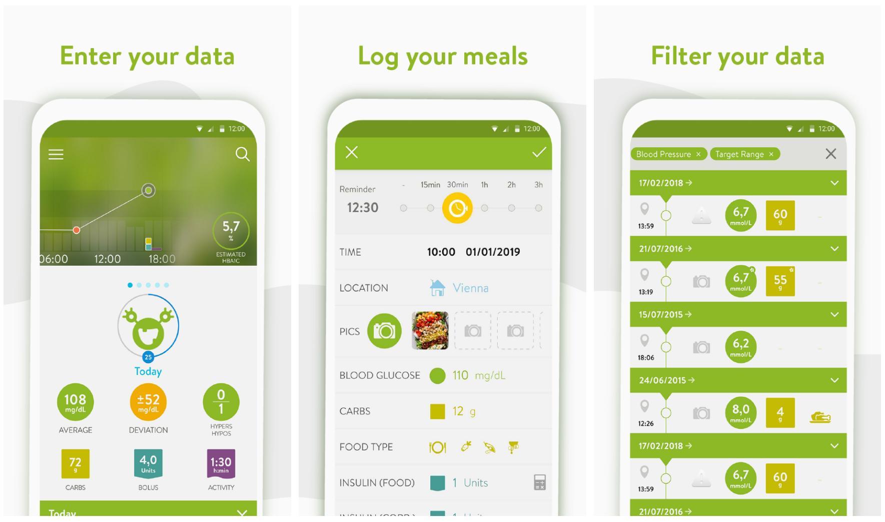
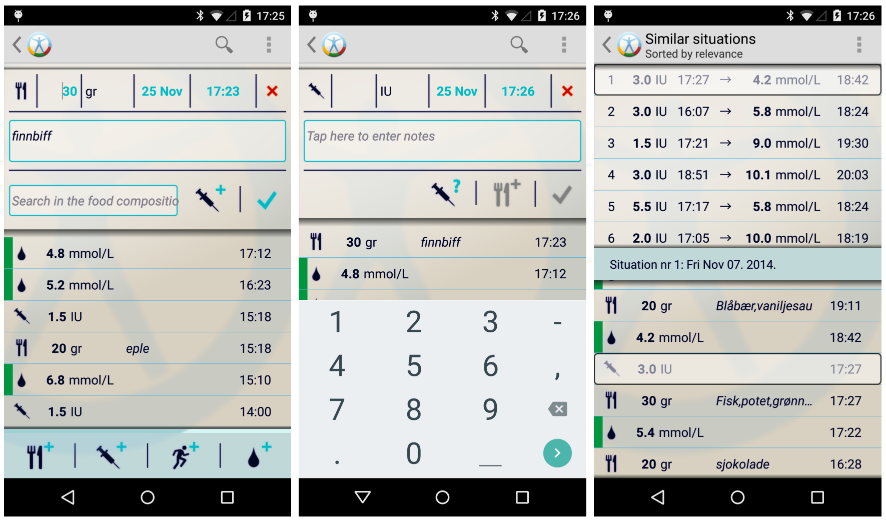


Figure 2.2: mySugr screenshots

2. Images from Google Play `https://play.google.com/store/apps/details?id=com.mysugr.android.companion&hl=english`

Diabetesdiary

Another approach is the Diabetesdiary [32] where data is gathered and used in self-monitoring. The user provides the application with data such as what the meal contained, or about physical exercising, and stores the data so that the user have full control over what the value was in, for example, insulin at a given time. As can be seen in Figure 2.3, the application has a button for registering insulin, and see the records made. Together with this it provides tools for monitoring food intake. The images are from the Diabetesdiary webpage [3].


17:25

30 gr | 25 Nov | 17:23

finnbiff

Search in the food compositio

4.8 mmol/L		17:12
5.2 mmol/L		16:23
1.5 IU		15:18
20 gr	eple	15:18
6.8 mmol/L		15:10
1.5 IU		14:00

17:26

IU | 25 Nov | 17:26

Tap here to enter notes

30 gr	finnbiff	17:23
4.8 mmol/L		17:12

17:26

Similar situations
Sorted by relevance

1	3.0 IU	17:27	→	4.2 mmol/L	18:42
2	3.0 IU	16:07	→	5.8 mmol/L	18:24
3	1.5 IU	17:21	→	9.0 mmol/L	19:30
4	3.0 IU	18:51	→	10.1 mmol/L	20:03
5	5.5 IU	17:17	→	5.8 mmol/L	18:24
6	2.0 IU	17:05	→	10.0 mmol/L	18:19

Situation nr 1: Fri Nov 07. 2014.

20 gr	Blåbær,vaniljesau	19:11
4.2 mmol/L		18:42
3.0 IU		17:27
30 gr	Fisk,potet,grønn...	17:27
5.4 mmol/L		17:22
20 gr	sjokolade	16:28

Figure 2.3: Diabetesdiary application

General

There are several options to these, and most big technological companies offer some way of recording health data. Apple HealthKit[33], Google Fit[34] and Samsung Health[35] to mention a few. These often focus on exercising and following a goal in for example number of steps each day, but the last years they have integrated support for personal health monitoring. Other services such as CareZone [36] and NoMoreClipboard [37] have the same focus as HealthVault, that is, to collect data and organize health information. For example, NoMoreClipboard has a focus on an own and personal health record where one can access the data anywhere and any time. They also state that it does not matter what system one uses.

The services mentioned above are discussed and explained in detail in chapter

3. Images for Diabetesdiary `http://www.diabetesdagboka.no/en/howtos/`

Chapter 8, but as one can see, many solutions offer ways to collect your health data.

2.2 Disease surveillance

Through the years, surveillance systems have served greatly in detecting diseases in society and responding to the infectious disease outbreaks. [38]. Since the world has changed extensively the past decades, we sit with more information on each individual than ever, and this has the potential to help us develop solutions that can alert the general public earlier in situations where there are infectious disease outbreaks. Traditional systems often have a lag between their readings and the time one can say for certain that there is an outbreak [38].

As the internet and computers grow, more and more data is generated [11]. This data can, for example, be about an individuals health, where one uses a Body Area Network (BAN) to record heart rate or sleep quality of a user. Another example is to record individuals with type 1 diabetes blood glucose to manage their chronic condition [39]. The Internet of Things (IoT) [40] has created an opportunity where one can easily record data and push it to stable storage for analyzing. These factors lead to an opening in creating systems that can analyze huge amounts of data and make conclusions on a larger scale.

Modern technology and solutions allow for faster response and reaction to global threats such as infectious diseases, and there are several solutions which have sped up the detection of these. There are systems in place already, such as using Googles search trend to predict outbreaks [41]. Others go another route and develop apps for disease surveillance in Africa [42]. This project included both livestock and human in a trial, where within the first 5 months of the deployment, a total of 1915 clinical cases were reported.

The Medical Informatics and Telemedicine group at the University of Tromsø has looked into using blood glucose levels to create an automatic infection system for a while, and already in 2007, they investigated the possibility of using it to create an automatic infection detection system [43]. Here it was concluded that the blood glucose data had the potential to be used as an indicator. While the MI&T group used blood glucose data, others analyze tweets on Twitter at the time of outbreaks [44] [45].

In 2006 a team of researches, epidemiologists and software developers founded HealthMap [46] [47]. It offers disease outbreak monitoring in real-time and bases itself on several sources such as Google news, WHO [4], news sources, and Baidu [4]. They have created a map where alerts of outbreaks are added as can be seen on Figure 2.4, when one press on the dots, information about what has happened and where the source originated. For example, on the 6th of May there was a young woman who died from a rabies infection in Norway [5]. On the map, it is marked as an incident. There is also a summary of the story where you can find the sources and more information about what happened.

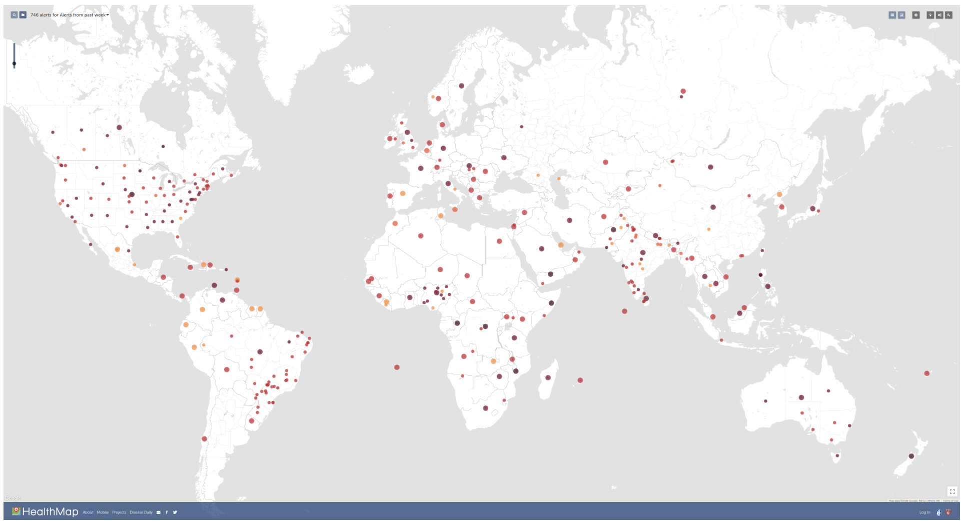


Figure 2.4: Screenshot from Healthmap [46]

2.3 Diabetes

Diabetes mellitus, often only referred to as diabetes is a chronic condition where the blood glucose is too high [48]. Blood glucose (also referred to as blood sugar) is consumed when eating, where the food is broken down in the digestion. The body uses the pancreas to produce insulin, which then helps the glucose in the blood to go into cells and create energy [48]. Therefore the result of not producing insulin is that the glucose would stay in the blood forever.

There are several types of diabetes; type 1, type 2, gestational, and others such as monogenic diabetes, even though the latter is less common [49]. As mentioned, this project focuses on individuals with type 1 diabetes. Diabetes milletus type 1 is when the body produces small amounts of insulin or no insulin

4. "Baidu" `http://www.baidu.com/` Visited (2019-05-04)
5. "Birgitte Kallestad (24) fra Hordaland døde av rabies" `https://www.aftenposten.no/norge/i/70B5mo/Birgitte-Kallestad-24-fra-Hordaland-dode-av-rabies` (Visited 2019-05-10)

at all [48]. Normally this is a result of the body's immune system attacking and destroying the betacells in the pancreas [48]. Type 1 diabetes is also referred to as juvenile diabetes since in most cases it affects children or young adults [50].

Monitoring diabetes

Since the body can not produce insulin, the individuals have to inject insulin into their bodies daily. This results in the need to self-monitor and using applications for recording data, as mentioned in Section 2.1. Cindy Marglin and Razcan Bunescu promoted and facilitated research in blood glucose level prediction where they gathered eight weeks worth of data with continuous glucose monitoring, insulin, physiological sensors, and self-reported live-event data for six people with diabetes type 1. This can be seen in Table 2.1, which is a table from the Ohio paper. They provide this data to researchers and scientists which want to gather more knowledge about which data is relevant to obtain and to try and understand how the data can be used in research.

Table 2.1: Datatypes: The OhioT1DM Dataset for Blood Glucose Level prediction, p. 2 [51]

Value	***Explanation***
patient	The patient ID number and insulin type. Weight is set to 99 as a placeholder, as actual patient weights are unavailable.
glucose_level	Continuous glucose monitoring (CGM) data, recorded every 5 minutes
finger_stick	Blood glucose values obtained through self-monitoring by the patient.
basal	The rate at which basal insulin is continuously infused. The basal rate begins at the specified timestamp ts, and it continues until another basal rate is set.
temp_basal	A temporary basal insulin rate that supersedes the patient's normal basal rate. When the values is 0, this indicated that the basal insulin flow has been suspended. At the end of a temp_basal, the basal rate goes back to the normal basal rate.
bolus	Insulin delivered to the patient, typically before a meal or when the patient is hyperglycemic. The most common type of bolus, normal, delivers all insulin at once. Other bolus types can stretch out the insulin dose over the period between ts_begin and ts_end
meal	The self-reported time and type of a meal, plus the patient's carbohydrate estimate for the meal.
sleep	The times of self-reported sleep, plus the patient's subjective assessment of sleep quality: 1 for Poor; 2 for Fair; 3 for good.
work	Self-reported times of going to and from work. Intensity is the patient's subjective assessment of physical exertion, on a scale of 1 to 10, with 1 being most physically active.
stressors	Time of self-reported stress.
hypo_event	Time of self-reported hypoglycemic episode. Symptoms are not available, although there is a slot for them in the XML file.
illness	Time of self-reported illness.
exercise	Time and duration, in minutes, of self-reported exercise. Intensity is the patient's subjective assessment of physical exertion, on a scale of 1 to 10, with 10 being most physically active.
basis_heart_rate	Hear rate, aggregated every 5 minutes.
basis_gsr	Galvanic skin response, also known as skin conductance, aggregated every 5 minutes.
basis_skin_temperature	Skin temperature, in degrees Fahrenheit, aggregated every 5 minutes.
basis_air_temperature	Air temperature, in degrees Fahrenheit, aggregated every 5 minutes.
basis_step	Step count, aggregated every 5 minutes.
basis_sleep	Times when the Basis band reported that the subject was asleep, along with its estimate of sleep quality.

Dangers of diabetes

Individuals with diabetes have a chance to get hyperglycemia, which is high blood glucose. It is a result of the body not getting enough insulin, and can, for example, occur if the individual has forgotten to inject insulin. [52] The first symptoms of hyperglycemia may include: [53]

1. Thirst or a parched mouth
2. Frequent urination
3. High levels of ketones in the urine

When an individual can not break down the sugars, and the cells do not get the glucose they need for energy, the body begins to burn fat for energy, which in return produces ketones. High levels of ketones can poison the body. [52].

On the other side, there is a danger of having too low blood glucose. This is called hypoglycemia, and the most severe symptoms are: [54]

1. Unable to eat or drink
2. Seizure or convulsions
3. Unconsciousness

Hypoglycemia often occurs in people with type 1 diabetes [54] In the worst case, both of these can lead to diabetic coma or even death.

2.4 Security and Privacy

When working with health-related personal data, many aspects have to be taken into account, such as authentication, encrypting data, security, and ensured privacy for the user. For this project, a natural aspect to look into is the General Data Protection Regulation (GDPR), encryption of data, how to authenticate a user and using de-identification to ensure the privacy of each user. The concern for privacy and security of a user and the users' information has been around for a while, but when the GDPR went into effect on the 28th of May in 2018, there was a shift in the industry.

2.4.1 General Data Protection Regulation

Datailsynet[20] is an independent inspection body in Norway which is responsible for ensuring that privacy and security of the people are safeguarded in the processing of digital data registers [55]. Datatilsynet has a checklist of sixteen points about the GDPR, where each point on the list describes an important aspect of handling personal data, such as asking for permission, keeping protocols over how to store the data, and it also goes into detail about which type of data that is stored.

For the thesis, the most crucial aspect of GDPR is that the GDPR enforces the service to allow the user to see all the data about themselves, and delete everything if wanted. One also has to ask consent from the user before storing or collecting any personal data, and when it is stored there should be apparent to the user what is stored and what the data is used for.

2.4.2 HTTPS

Since there are rules in place for protecting a users' privacy, there has to be a technical solution on how to solve these. In the early days of the internet, HTTP was invented at CERN by Tim Berners-Lee [56]. Later it is adopted by most of the web. There was a need for creating a secure version of HTTP, where a third party could not read the data. Therefor HTTPS came along, which run HTTP on top of SSL/TSL [57].

HTTPS protects the users' privacy when the connection to a web service by bidirectional encryption between the service and the user. Therefore it protects against third-parties eavesdropping or editing the communication in any form. There is a cost of using HTTPS, since encrypting and securing data, inevitably leads to an increase in infrastructure costs [58]. It can also lead to direct and noticeable protocol-related performance costs[58].

2.4.3 De-identification

The Health Insurance Portability and Accountability Act (HIIPA) provides guidance regarding methods for de-identification [59]. De-identifications ensures the removal of all personally identifying private health information (PHI) from the records [60]. When Uzner evaluated the State-of-the-Art in Automatic De-identification [60], he says the following:

"According to HIIPA, for the data to be treated as de-identified it must clear on of two hurdles:

1. An expert must determine and document “that the risk is very small that the information could be used, alone or in combination with other reasonably available information, by an anticipated recipient to identify an individual who is a subject of the information.”

2. Or, the data must be purged of a specified list of seventeen categories of possible identifiers relating to the patient or relatives, household members and employers, and any other information that may make it possible to identify the individual.10 Many institutions consider the clinicians caring for a patient and the names of hospitals, clinics, and wards to fall into this final category because of the heightened risk of identifying patients from such information" ([60] vol. 14, no. 5, pp. 550)

This highlights the requirement for de-identifying a user. No information, alone or in combination with others, should be enough to identify the individual. Since the project handles sensitive personal data, such as the blood glucose level, it has to take the threat of the user being identified seriously. This is addressed in chapter Chapter 5.

2.4.4 JSON Web Tokens

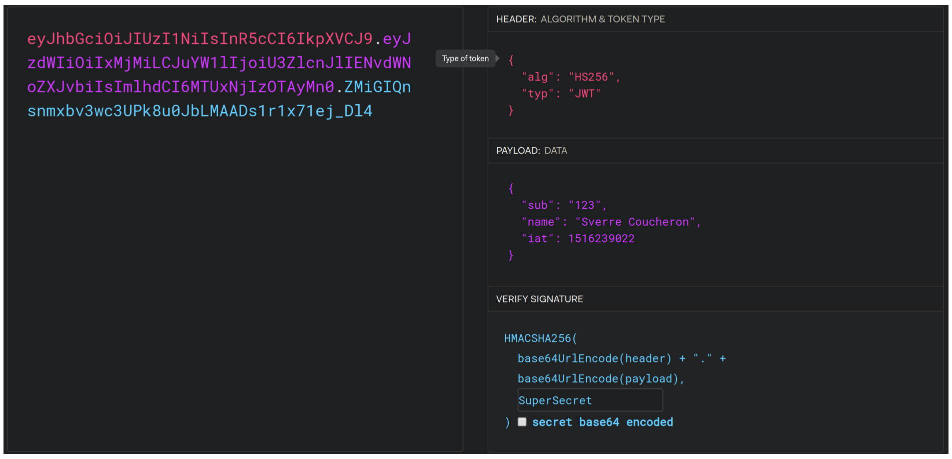
According to the RFC standard;"*JSON Web Token (JWT) is a compact, URL-safe means of representing claims to be transferred between two parties*"[61]. A JWT is a way for the user to authenticate who they are once, where the server sends back a token that can be used to authentication at a later point. They are often used in mobile applications since it provides a seamless log in feature where the users can log in once and then use the application daily without logging in again if the expiration date of the token is long enough.

JSON web tokens is a standard in the RFC 7519[61], and its basis is that it should fit in the header of a basic RESTFull HTTP call. A JWT consist of three parts, the header, payload, and the signature that verify that the message is not altered along the way. An example of how a JWT may look can be seen in Figure 2.5. All parts are written in the JSON format, and after that Base64 encoded. [6]

The header of a JWT usually consists of two parts; the type of token and which signing algorithm. The type of token is JWT, while the type of signing algorithm varies on the implementation, but it is often RSA and HS256 [62]. The payload

6. "Base 64 encoding and decoding" Link: `https://developer.mozilla.org/en-US/docs/Web/API/WindowBase64/Base64_encoding_and_decoding` (Visited 2019-05-10)

of a JWT contains claims that the user sends, together with some additional data. These claims are statements about the user, such as who the issuer is, the expiration time for the token and private claims that are created to share information between the user and the service. The last part of the token is the signature. The signature is used to verify that the message as not changed in transmission. How this is created will vary between implementations. If one creates both the issuer of tokens and the application that is to use the service and has a way to distribute a shared secret, HS256 could be used.



```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ
zdWIiOiIxMjMiLCJuYW1lIjoiU3ZlcnJlIENvdWN
oZXJvbiIsImlhdCI6MTUxNjIzOTAyMn0.ZMiGIQn
snmxbv3wc3UPk8u0JbLMAADs1r1x71ej_Dl4
```

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

```
{
  "sub": "123",
  "name": "Sverre Coucheron",
  "iat": 1516239022
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  SuperSecret
) ☐ secret base64 encoded
```

Figure 2.5: Example of a JWT token

As can be seen in Figure 2.5, the signature is created by adding the base64 encoded header and payload together by separating them with a dot. The signature is hashed, and since the secret is shared between the user and service, it can be read by the service, and therefore it confirms the authentication of the user.

While JWT's work in authenticating the user, it is worth mentioning that it does not encrypt the payload. As a result of this, one should be careful when sending sensitive data unless there is encryption in place to ensure that no one can read the payload.

2.5 Standardization

In the health sector, interoperability and efficiency in the sharing of data is an important aspect. The data from a patient may be needed in another sector, or by a general practitioner that is examining a patient. Earlier there has been a spread in how to define data and what is good medical practice, whereas in the later years the need for a standard has become apparent as we move towards electronic health records [63].

2.5.1 Health Level 7

Health Level 7 International (HL7) is a not-for-profit, ANSI-accredited standard developing organization [14]. Often it is seen as the standard for sending health data and labeling health data between software used by healthcare providers. One of the primary missions of HL7 is to provide interoperability between the system so that they can exchange data.

Health Level 7 offers guides and introduction courses in how to work with their framework and exchange data between electronic health records. There are different type of standards, such as the primary standards which are considered the "most popular standards integral for system integrations, interoperability and compliance" [64]. They also offer implementation guides to aid developers when creating projects. FHIR [7] is HL7's standard for health care data exchange. it offers the following guides:

1. Basic framework on which the specification is built
2. Supporting implementation and binding to external specifications
3. Linking to real world concepts in the healthcare system
4. Record-keeping and data exchange for the healthcare process

All of these has a foundation in how to create a service with the support for FHIR.

2.5.2 LOINC

Logical Observation Identifiers Names and Codes (LOINC) is, as the name suggests, a way of giving identifiers codes and names to provide a global universal standardization [65]. It is often used for identifying health measurements, observations and documents [65] or in combination with HL7 version 2 messaging standard, as can be seen in Figure 2.6. [8]

7. FHIR `http://www.hl7.org/FHIR/` Visited (2019-05-11)
8. Loinc example `https://dev.loinc.org/wp-content/uploads/2016/09/get_started_1_coded_results.png` (Visited 2019-04-08)

LOINC provides a database where one can find all their codes with a simple search if one creates a user. Because of this, it is often used as a standard for labeling different types of health-related data, so that different health services can communicate with each other and expect to know what each code represents.

An example is a code for heart rate post exercise, which has the code 40442-6. By using an identifier as this, one can easily log the heart rate at a given time and therefore use the data at a later time. Because it is a standard, it would also suggest that the data could be imported to, e.g., a hospital and be used by the general practitioner to aid the patient when in need.

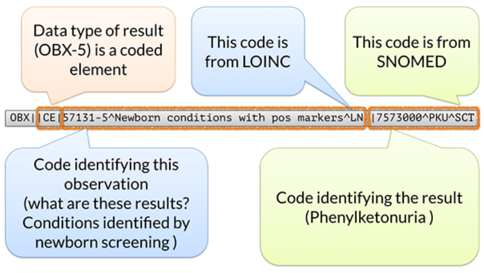

Data type of result (OBX-5) is a coded element
This code is from LOINC
This code is from SNOMED
OBX||CE|57131-5^Newborn conditions with pos markers^LN|7573000^PKU^SCT
Code identifying this observation (what are these results? Conditions identified by newborn screening)
Code identifying the result (Phenylketonuria)

Figure 2.6: LOINC example with HL7 integration

2.5.3 FullFlow

The Norwegian Center for E-health research has a project called Full Flow which aims to integrate patient-gathered data to the Norwegian electronic health records. Per now this has not been possible, but they believe that by shining a light on the problem and researching on how to integrate self-recorded and self-managed systems for real patients into the support systems designed for health care personnel, it can unveil a vast amount of new information and insight. [66]

2.6 EDMON

Electronic Disease Surveillance Monitoring Network (EDMON) is a proposal by the scientist and researches from the Department of Computer Science at the University of Tromsø in cooperation with the Norwegian Centre for E-health Research at the University Hospital of North Norway. It proposes a solution that supports the detection of infections before the onset of the first symptoms [7] by the monitoring of individuals with type 1 diabetes. These individuals monitor their blood glucose levels to manage their condition, and with ubiquitous technology such as mobile phones.

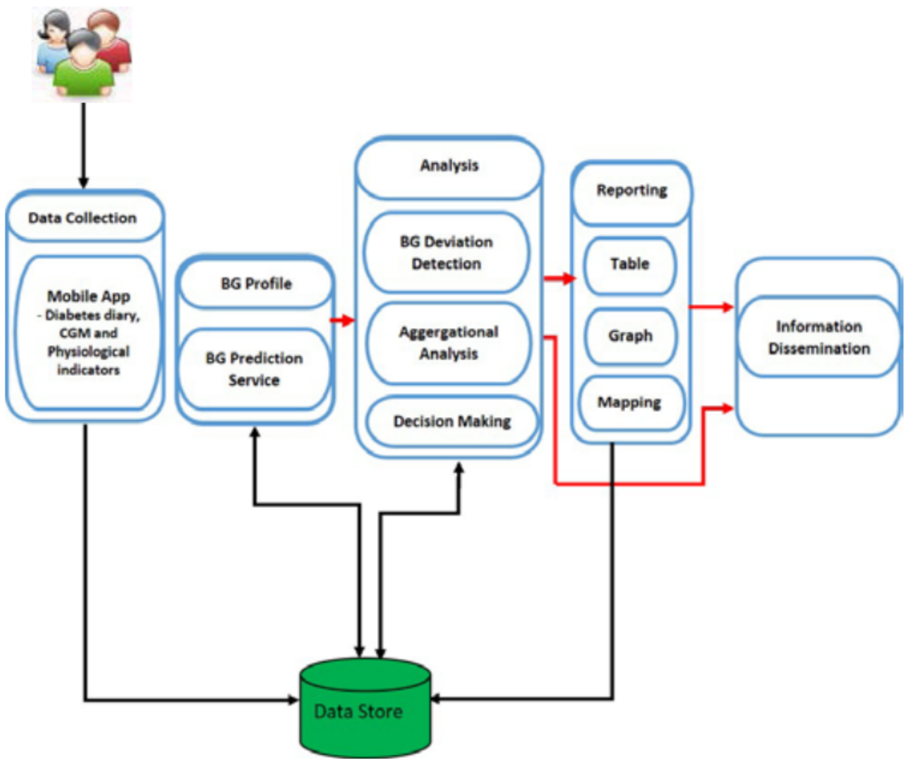


Figure 2.7: Proposed EDMON System Architecture [67]

Since the use of blood glucose levels to detect early outbreaks of diseases [68] [69] [70], EDMON proposes a system that consists of five different components [67], together with a data store as can be seen in Figure 2.7. Firstly a way of collecting the data is needed. This is suggested to be in the form of a mobile application which will collect the individuals diabetes data together with other physiological indicators [67]. The data will be analyzed and mapped so that the information can be used to give an overview of disease in the society as a whole.

For this thesis, the author thinks that the most critical aspects of the EDMON design is the collection and storing of the data together with the possibility of extending the functionality of the back end solution. The reason for this is that the author is limited in time. The thesis, therefore, discusses the possibilities and opportunities that the designed system contributes to EDMON, together with the path forward.

2.7 State-of-the-Art

This section showcases the systematic review conducted of academic literature. The literature is about disease surveillance systems, self-monitoring diabetes systems, and standardization in health care systems.

The purpose of the state-of-the-art review is to gather resources and knowledge about the current solutions while finding the information available on how to creating an electronic health record system which aims to provide individuals with diabetes type 1 a solution to self-monitor and self-report. Using the gathered data as the main attributes in creating an infectious disease warning system, is in itself a difficult task. Therefore we try to gather as much information of the field as possible as a way to contribute to the solution of the requirements and to investigate the state-of-the-art for the thesis topic.

2.7.1 Data Sources and Search Criteria

Several electronic databases contain relevant literature. These were used in the state-of-the-art literature review and are as following:

- Scopus (`https://www.scopus.com/`)
- Google Scholar (`https://scholar.google.no/`)
- Journal of Diabetes Science and Technology (`https://journals.sagepub.com/home/dst`)
- IEEE (`https://www.ieee.org/`)
- ScienceDirect (`https://www.sciencedirect.com/`)
- ACM Digital library (`https://dl.acm.org/`)
- PubMed (`https://www.ncbi.nlm.nih.gov/pubmed/`)

When choosing which databases to query, it was essential to try and use relevant ones. Databases such as Scopus, google scholar, IEEE are massive databases which contain thousands of papers and relevant literature. The Journal of Diabetes Science and Technology, on the other hand, is a journal, and not a database containing papers from several hundred journals. Because of the relevance, it is included in the systematic review. To exclude the papers deemed as not relevant several exclusion criteria are applied. How relevant a paper is are based on the abstract and sometimes the whole text if needed. The exclusion criteria are the following;

- Papers without a full-text or PDF
- Papers that is not written in English
- Papers only focusing on the physiological aspects of diabetes management
- Papers that only focus on machine learning approaches of disease surveillance

When searching for relevant literature, a specific combination of keywords is made. Since there are several aspects to include and therefore we want to be as specific as possible to filter out papers that are not relevant. Together with searching for relevant papers a search over Google Play, Apples App Store, and a search on google to see if there are related applications on the market were conducted.

The main focus of the literature search is to find papers which focus on creating a back-end server which records confidential health-related data and possibly has a way of using this data to create a system for disease surveillance. Together with this self-monitoring of type 1 diabetes works as an essential factor, so it is also included. The following is the resulting query that was used to search the databases;

Systematic review query

((((((diabetes) AND type 1 diabetes AND mellitus) AND server) AND self-care) AND self-management) OR disease AND surveillance AND server) AND (LIMIT-TO (SUBJAREA , "COMP")) AND (LIMIT-TO (LANGUAGE , "English"))

As can be seen, the search is limited to the subject of computer science in the databases that this is a possibility. Because of this, one could say that it is an added exclusion criterion. Most of the searches are only done with the query up until this criterion.

Table 2.3: Results from systematic review

Source	***Found***	***Hits***
SCOPUS	297	3
Google Scholar	1010	9
Journal of Diabetes Science and technology	168	2
IEEE	22	1
ScienceDirect	87	2
ACM	1	0
Total	***1585***	***11***

In Table 2.6 there is a brief overview of the papers that are deemed most relevant and therefore included as the primary source of information in this literature review. Since the thesis is under the EDMON project, all papers from the EDMON group are relevant and included in the search.

In Table 2.3 one can see the number of articles found together with the hits. These hits are based upon the excluding criteria together with using the Prisma flow diagram [71]. The process can be seen in Figure 2.8. Because of the time limitation in this thesis, the author chose to be strict in choosing relevant papers in the literature review.

Some of the leftover papers are included in the thesis but is not a part of the systematic review for this reason. In Table 2.3 we can see that the Journal of Diabetes Science and technology yielded 168 papers even though it is a journal. This is more than ACM, ScienceDirect, and IEEE yielded together, which showcases how relevant the journal proved to be.

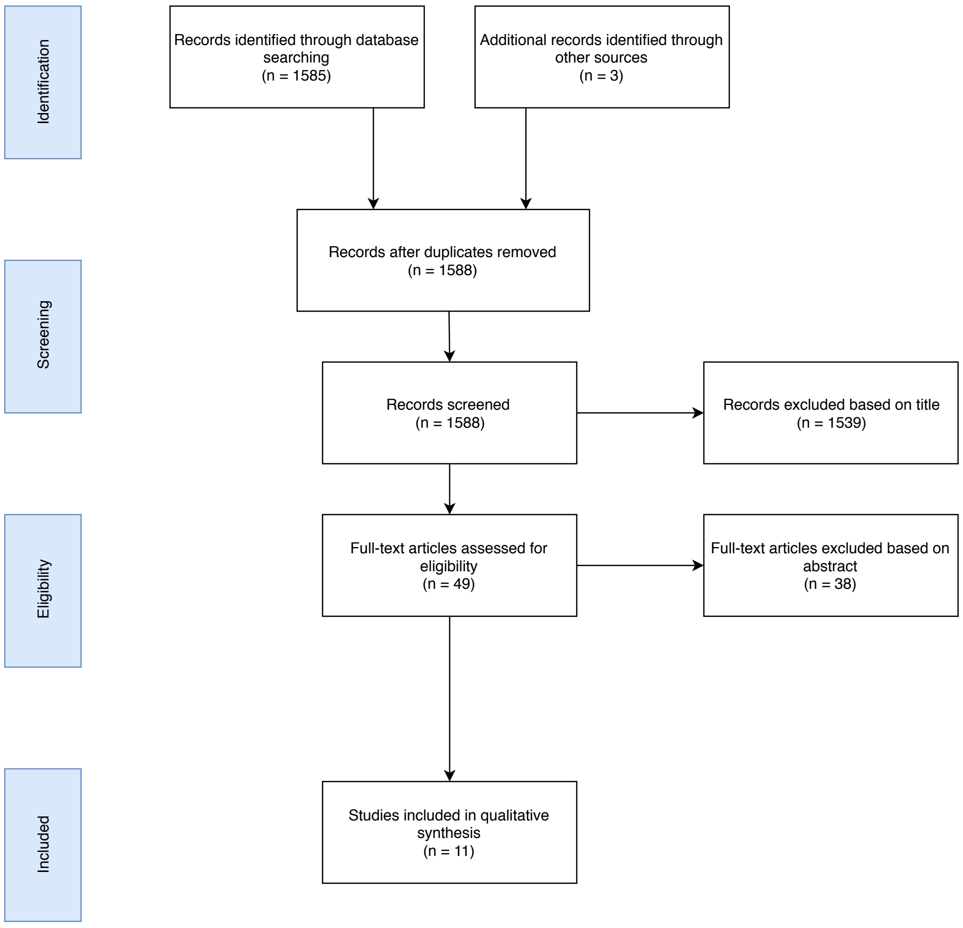

Identification
Screening
Eligibility
Included
Records identified through database searching (n = 1585)
Additional records identified through other sources (n = 3)
Records after duplicates removed (n = 1588)
Records screened (n = 1588)
Records excluded based on title (n = 1539)
Full-text articles assessed for eligibility (n = 49)
Full-text articles excluded based on abstract (n = 38)
Studies included in qualitative synthesis (n = 11)

Figure 2.8: PRISMA Flow Diagram

The 11 papers we are left with are then grouped into three different groupings:

- Group 1: Disease Surveillance Systems
- Group 2: Diabetes Monitoring
- Group 3: EDMON Papers

The first group is disease surveillance. In this group, the papers that have a foundation of what is needed to build a disease surveillance system such as; security, disease surveillance algorithms or definitions and showcasing of electronic health systems. Group 2 are papers with an emphasis on diabetes monitoring, which includes data sets or papers that contain information about how one should build an application for individuals with type 1 diabetes. The last group is the papers that this thesis is a part of, EDMON. These are the most important ones since they talk specifically about what the project as a whole is going to look like and showcases an example of the design.

Table 2.4: Detailed list of relevant mobile and web-based applications

App	***Group***	***Significant features***
HealthVault [28]	1	Electronic health system with the possibility to extract data from several devices such as smartwatches.
NightScout[30]	3	Open source project which integrates with CGM data to have real-time access and self-monitoring for diabetes. Has a personal website and mobile applications for users
MySugr [31]	3	Diabetes manager. Self-monitoring.
Apple HealthKit [33]	2	Electronic health record which integrates data seamlessly from other mobile applications together with Apple Products
Google Fit [34]	2	Electronic health record. Integrates with other app. and has open API that one can post and fetch data to.
Samsung Health [35]	2	Electronic health record. Keeps track of sleep, food intake, stress levels and heartrate. Focus on fitness.
CareZone [36]	1	Electronic health record. Organizes health information.
NoMoreClipBoard [37]	1	As the same suggests it is a online storage point for health data. (Electronic health record)
HealthMap [46]	4	Map that showcases flu, Ebola or other disease outbreaks by parsing news in different countries

Table 2.6: Detailed list of reviewed literature

Author	*Group*	*Significant features*
E. Arsand et al [32]	1	Creates and test a diabetes eDiary. prototype developed and tested on real users. Tries to improve the solutions at the time.
C. Marling [51]	2	Provides data set and an example of a data format for diabetes data. Real test data from users.
I. Rodríguez-Rodríguez [39]	2	How the use of ICT and body are networks can help improve the lives of individuals with type 1 diabetes. Looks into challenges and where we have to move to get an ICT structured platform.
N. Menachemi et al. [25]	1	After HITECH Act of 2009 this discusses the benefits and drawbacks that occurred in creating an electronic health system. Describes from drawbacks such as costs and how it disrupts the workflow to learn a new system, together with privacy concerns.
A. Appari [18]	1	Looks at the state-of-the-art in security within healthcare. Discusses several aspects over several domains while including privacy concerns from both perspectives (user and provider)
O. Granberg [43]	1, 2	Investigates and proves that blood glucose levels can be used to detect contagious diseases earlier. Creates a prototype named AID. Detects earlier than other systems that were used at the time.
M. Meingast et al. [17]	1	Discusses the security and privacy aspects now as EHR is becoming more prominent. Describes methods used and suggests solutions that exist together with what can be improved on
R. Heffernan et al. [9]	1	Implemented system that collect information from emergency departments to create an early detection system
P. Kostova [8]	1	Lays out a road-map to integrate digital public health and surveillance systems. Highlights the needs a system that scan services such as Facebook, Twitter, and other big platforms need to work together and analyze the data.
A. Z. Woldaregay et al. [67]	3	Explains the proposed EDMON system architecture. Showcases how BG values can be used to predict infectious diseases
A. Z. Woldaregay et al. [7]	3	Explains the outline for this thesis' groundwork. Looks into how one can use diabetes data to predict an infection incident. Explains architecture of EDMON.

Together with finding relevant literature, the author searched for mobile applications and server solutions that are state-of-the-art within the research field. These can be seen in Table 2.4. The applications are divided into groups; group 1 is an electronic health record, group 2 are health services which offer to store of electronic health data together with analysis on the data to improve the users' daily routines and self-monitor. Group 3 are applications created for individuals with diabetes to self-monitor, while group 4 are disease surveillance platforms that showcase current technology. To read more about each application, see earlier in this Chapter.

/3

Methods and Materials

The chapter contains information on which methods and materials are used throughout the project and thesis. Firstly, about the research paradigms and tools. Next, the materials and programming language, together with the rationale for choosing the given language, is explained. After that, a walk-through of how the testing is conducted is given. Lastly, a summarizing of the critique of the methods and materials. It is worth noting that the project is experimental.

3.1 Research Paradigm and Tools

In 1989 Denning et al. created a task force to define the paradigms for computer science as a discipline [24]. They defined three paradigms that divide the discipline of computing; the first is theory, second is abstraction while the third is design. They state that the theory paradigm is rooted in mathematics and has four steps;

"*(1) characterize objects of the study (definition);*
(2) hypothesize possible relationships among them (theorem);
(3) determine whether the relationshiups are true (proof);
(4) interpret results." (Denning et al., p. 10)

The second paradigm, abstraction (modeling), is rooted in experimental scientific methods, which has the following four steps;

"*(1) form a hypothesis;*
(2) construct a model and make a prediction;
(3) design an experiment and collect data;
(4) analyze results." (Denning et al., p. 10)

The third paradigm, design, is rooted in engineering and has the following four steps;

"*(1) state requirements;*
(2) state specifications;
(3) design and implement the system;
(4) test the system." (Denning et al., p. 10)

For all of the paradigms, the steps are iterative. One has to go back and look over the earlier steps and reiterate to ensure correct results. Denning et al. argue that when examining closer these are intertwined, so one can not use one of the paradigms without touching the others.

This thesis uses a bit from all of the paradigms but has a focus on the design paradigm. Firstly, the requirements are created together with the specifications of the implementation. After that, the system is designed and implemented, and lastly tested. There have been iterations over the four points to ensure that the system was correct and had a satisfying design and implementation. This will be mentioned in Chapter 5 and Chapter 6.

3.2 Materials

Under the development of the service, several tools are used. Vertelo [1] served as a visualizer for the database, meanwhile, Draw. io [2] served as a design and illustration tool for figures. For further implementation of the project prototype, the following libraries and software is used.

- Golang[3], a programming language designed by Google
- Gin libarary [4], a framework used for HTTP requests and debugging

1. Vertelo, design of the database. `https://www.vertabelo.com/` (visited 2019-05-11)
2. Draw.io `https://www.draw.io/` (Visited 2019-05-11)
3. Golang official site `https://golang.org/` (Visited 2019-05-11)
4. Gin HTTP library `https://github.com/gin-gonic/gin` (Visited 2019-05-11)

- Postgres [5], an open source database used as the systems database
- Gin-jwt [6], used as a middleware to handle JSON web tokens
- Openssl [7], used to create a certificate and public key for HTTPS requests

For testing, Python was chosen as the programming language since the author lacks graphical programming experience in golang. It was used to test and create graphs under the testing of the project.

3.3 Golang

The implementation of the project is done in Golang [8]. Golang is a programming language developed by Google, and it is statically typed. It is designed to have a syntax similar to C, but with features such as garbage collection and structural typing. It was designed to address the criticism of other languages that were used at google, but to keep some of the characteristics such as readability and usability, high-performance networking and multiprocessing, together with runtime efficiency [9]. Several applications are developed using golang, such as Docker which is a tool for deploying Linux containers. The cryptocurrency Etherum is also an example, while other services use golang to an extent, such as Netflix, MongoDB, and Uber [72].

Goroutine

An interesting feature of Golang are goroutines. The Golang tour [10] defines a goroutine as "... a lightweight thread managed by the Go runtime". This provides an application with the possibility to easily spawn new thread to handle a part of the workload separately from the main application.

5. Postgres open source database `https://www.postgresql.org/` (Visited 2019-05-11)
6. Gin-JWT library `https://github.com/appleboy/gin-jwt` (Visited 2019-05-11)
7. OpenSSL `https://www.openssl.org/` (Visited 2019-05-11)
8. Golang `https://golang.org/` (Visited 2019-05-13)
9. Robert Criesemer about Golang `https://talks.golang.org/2015/gophercon-goevolution.slide` (Visited 2019-05-13)
10. A tour of go: Goroutines `https://tour.golang.org/concurrency/1` (Visited 2019-05-13)

Channels

Another feature Golang offers are channels. A channel is a way to create a flow of data between to components. One can create a shared channel, where one can send an receive values [11]. By default, a channel block until it receives data, which in turn creates the possibility for goroutines to synchronize without locks or condition variables.

3.3.1 Rational for choosing Golang

Golang is chosen as a solution for developing the system since the author has experience in programming in C while wanting to explore the strength and weaknesses of a programming language at Google. The usage of goroutines and channels fit the type of system that is created, where there are several parts of analysis and algorithms that have to be done. These can be outsourced to goroutines at all time, which may take off some load of the main application.

The language also offers plenty of libraries for most usages, and enforce strict typing, which in turn forces the programmer to write readable and well-documented code. This leads to the code being easier to maintain later in the EDMON project.

To ensure that all resources available are used when needed, there is also the possibility to create a pool of workers (goroutines) which fetches data from a channel if there is any work to be done. By having the choice to put parts of the workload over on several small workers, it may ease the workload when adding several microservices.

3.4 Literature review

When gathering functional and non-functional requirements for the project, a literature review is conducted. This can be seen in chapter 2, under the subsection "State Of the Art". The main outline for the project is formed during this review and in meetings with the supervisor and co-supervisors. The design of the project can be summed up as below.

An individual with diabetes type 1 should record information about their

11. A tour of go: Channels `https://tour.golang.org/concurrency/2` (Visited 2019-05-13)

condition, such as blood glucose, and this information is permanently stored on the system. This system analyzes the data, and give the user feedback if the values are within the normal range. The individual should have the right not to share their data, and use the application for their gain, but this limits the feedback of the disease surveillance aspect for their use. The system should be able to store images or short videos for the individuals to look back on. This can help them remember how they felt or their state of mind at a given point in time. The system should save the location of both the individuals and the recording they do, such that it can analyze the result and based upon the number of sick individuals within an area, and respond accordingly.

The literature review also helped gain knowledge about what the current solutions within health applications offer, together with how disease surveillance systems are designed.

3.5 Testing

Several aspects of the system are benchmarked and tested. The metrics used are throughput in the form of the number of requests, memory and CPU activity, and some API calls are profiled to see which functionalities demand more resources.

3.5.1 Experimental design

All experiments are conducted on a single computer, where all aspects run locally. Both the scripts that are used to test, together with the system itself are run on this computer. The hardware specifications used for testing the system can be seen in Table 3.1.

Table 3.1: Hardware specifications

Operating system	Ubuntu 18.04.2 LTS x86_64
Kernel	4.15.0-50-generic
GNOME	3.28.3
CPU	Intel i7-7700 (8) @ 4.200GHz
GPU	Intel HD Graphics 630
Memory	32056MiB

There are several experiments which provided insight and useful information about the system. These are explained below.

OhioT1DM dataset

All tests conducted on the system uses the OhioT1DM [51] dataset. This is provided to the author and contributes to test the system under real-life conditions with actual user data. This dataset contains six weeks of real user data from seven different users. In total, the training set used contains almost 29.000 records. For testing purposes, the data fetched is a subset of the needed values. The dataset is seeded to the database and system with the following values:

- User information
- Glucose level
- Meal information
- Basal reading
- Bolus reading

The user information is an ID, together with the weight of a user. The weight is by default set to 99. All the readings have a timestamp and a value. The meal information is the number of carbohydrates that are consumed, while the basal and bolus values are the insulin injected by the user. It is worth mentioning that the bolus readings come with a carbohydrate reading too, but this is translated into a meal in the database when seeding.

3.5.2 EDMON algorithm

The system is created to execute the EDMON algorithm. To ensure that the server can handle executing larger data-sets tests are conducted to showcase the performance when running a simple algorithm that mimics the EDMON intended algorithm. During testing the database is filled with different amounts of data from the OhioT1DM dataset. Since the dataset is limited in size, it is reused and used as a way to seed the database. The EDMON algorithm is then run on the given number of records in the database to see the execution time when fetching and calculating on the records.

To record the time spent executing Golang's time library is used [12]. To ensure accurate readings, each measurement is executed 100 times, and calculations of the mean together with the standard deviation. The highest and lowest value is also removed to ensure that huge outliers have less impact on the result.

The EDMON algorithm is run on a different number of records, as can be seen in Table 3.3. The algorithm is meant to run once each hour and use the last hour result to analyze if there are any deviations in the users' data, which may indicate illness. Because of this, the values chosen are to illustrate the number of medical records that the system has to fetch, together with the number of users this represents. The number of medical records seeded to the database will, therefore, represent the number of medical records in the last hour.

Table 3.3: Database information when executing EDMON algorithm

Times dataset is used	# of users	# of records
1	7	29 000
2	14	58 000
3	21	87 000
4	28	116 000
5	35	145 000
10	70	290 000
100	700	2 900 000

3.5.3 Performance

In this subsection, the average time spent executing in different aspects of the system is shown. It is worth noting that the tests explained below had some logging on the server side which has an impact on the system's performance, but it is in place to simulate how a real-life server would log important events.

All of the experiments below, which creates a medical record, has the same structure. This can be seen in Listing 3.1. The record contains a value, an arbitrary standardization code, a timestamp of the recording, a location, and a small image to illustrate the storage of a short video diary. We assume that the location is already created for the given medical record.

12. Golang Time Package `https://golang.org/pkg/time/` (Visited 2019-05-22)

Listing 3.1: HTTP Body of the medical record

```
{
    "Value": 781,
    "HL7": "LOINC-123",
    "Timestamp": "2019-05-05 12:25:01",
    "Location_id": 1,
    "recording": "data:image/png;base64,iVBORw0
       KGgoAAAANSUhEUgAAAAEAAAABCAQAAAC1HAwCAAAAC0
       lEQVR42mNk+A8AAQUBAScY42YAAAAASUVORK5CYII="
}
```

The user's values that are used to test the system's performance can be seen in Listing 3.2.

Listing 3.2: HTTP Body of the user

```
{
    "username": "Sverre",
    "email":"Sverre@Sverre.com",
    "password": "Sverre123"
}
```

3.5.4 Average time of execution for a user

The first time of execution test is to measure the average time to do a given set up and tear down script while the server is under load. To test this, the author wrote a python script which is used for several parts of the testing. This script does the following;

- Creates a user
- Log in as the user (with password and username)
- Stores the authorization token in memory
- Creates X number of medical records
- Creates X number of locations
- Creates X number of medical records with the given locations
- Fetches all the medical records from the system

- Deletes all of the medical records that were created

This script is referred to as the setup and tear down script later in the thesis. As mentioned, it is used in other parts of the testing, such as in the profiling. To test the average time for executing the script, it is run 20 times. The time spent executing was recorded using pythons time library [13]. The results are then stored and sorted, where the shortest and longest times are removed to get rid of possible outliers. The mean time and standard deviation are calculated to showcase how long it may take to execute. This measurement is done to provide insight to the user uploading more than one element, and how the system does when creating a user, medical records, and deleting them again. The test is executed with 1, 10, 50, 100, 150, 200, 250, 300 number of records of medical records, locations, and medical records with a location.

3.5.5 Average time of concurrent requests to the system

The next experiment done is to test the time of a given type of request by sending concurrent requests at the system. This is done to stress-test the server and see how well it handles several concurrent requests, together with how it may affect the time of each request. The python script used does the following;

- Creates a user
- Authenticate as the user (username and password)
- Create a medical record

This test runs with a given number of concurrent clients to test and see if it provides an insight into the performance of the system. It is worth noting that for creating a user, the script only creates a user, but when authenticating one create a user first. The same goes for creating a medical record, where the creation and authentication of a user is also a request to the system. The time measured is the time for the given type of request, excluding the other factors. Because of this, there is a lot more throughput and load on the server when creating a medical record, than when testing the creation of a user. The number of requests used is 10,100, 1000, and 5000. As with the previous experiments and tests, the results are saved and sorted, and the highest and lowest values are removed in case of a potential outlier. The mean and standard deviation is then calculated to showcase how long each type of request on average takes,

13. Python Time Library `https://docs.python.org/3/library/time.html` (Visited 2019-05-23)

and how it potentially can vary depending on the type of request.

3.5.6 Profiling

To spot potentials bottlenecks and drawbacks, together with measuring how effective the system as a whole is, extensive profiling is conducted. This is done to see which parts of the system spends the most time executing and can showcase possible slow parts of the system and possible improvements. It is worth noting that when testing this all logging of the system is turned of since I/O operations are slow.

The first experiment and profiling records the CPU and memory usage of the system with psrecord [14]. It is a tool which records the memory and CPU usage of a given process and creates a graph over a given amount of time spent recording. How to use it can be seen below:

$$psrecord < PID > - - interval1 - -plotplot.png$$

To test several times, psrecord is running while the setup and tear down script with a different number of medical records created is executed. With 20 clients, the script executes with 10, 50, 100, and 200 medical records, where each client is set up by creating a new user and creating medical records for the given user.

Snapshot of memory and CPU

Secondly, to see the utilization of the computer's hardware during the executing of setup and tear down script, a snapshot of the Linux' system monitor is taken. It showcases the usage of CPU and memory when the server handles concurrent users. To run the script, a bash script is set up to create and concurrently run 500 users with 20 medical records each. This results in a continuous stream of requests to the server.

Profiling in golang

Golang provides tools to profile a given program. In this experiment the HTTP Golang package pprof is used [15]. The system sets up a monitoring page at port 6060 when it is set to debug mode, and it serves as a path to the profiling tool

14. psrecord `https://github.com/astrofrog/psrecord` (Visited 2019-05-25)
15. Pprof `https://golang.org/pkg/net/http/pprof/` (Visited 2019-05-25)

under `http://localhost:6060/debug/pprof`. The profiling tool is used for 30 seconds when the server is under heavy load from apache benchmark [16].

Profiling for three different API calls are done, as listed below:

- Creating a user
- authentication
- Creating a medical record

Apache benchmark is used by creating 100 concurrent clients and sending 10 000 requests, how to do this is showcased below;

ab -T 'application/json' -k -c 100 -n 10000 -p user.json http://localhost:8000/user/CreateUser

ab -T 'application/json' -k -H 'Authorization: Bearer <TOKEN>' -c <CONCURRENCY> -n <requests> -p user.json <IP:PORT/PATH>

The profiling creates a PDF which provides information about the call stack and which functionalities and functions in the implementation that spends the most time executing. Since this provides many graphs, a screenshot of the functions that execute for the longest is provided. Since apache benchmark is used, it also gives an indication of how many requests the server manages to execute in 30 seconds.

3.5.7 Throughput

The last experiment executed is to test the throughput of the server. Apache benchmark is used to spam the server with requests. This indicates how the server performs under heavy load, together with showcasing how many requests per second the server can handle and how long each request takes on average. It is also done to test the maximal number of concurrent clients that can spam the server. The throughput tests are done by executing two different apache benchmarks, first with 100 concurrent clients and then with 150 concurrent clients. The tests are executed by creating new users. The number of requests with 100 concurrent clients is 10000, while with 150 concurrent clients it is 15000.

16. Apache benchmark `https://httpd.apache.org/docs/2.4/programs/ab.html` (Visited 2019-05-25)

3.6 Evaluation methods

Feedback regarding the requirements and design of the application was mainly gathered in meetings with the supervisor and the co-supervisors. Here we discussed wishes and improvements, how the application can be expanded, and which security measures that had to be in place.

Since there was no mobile application created, the evaluation that has been done of the complete system is mostly about how well it performs, in the form of requests per second and hardware load.

3.7 Critique of the Methods Used

The main critique of how the project's methods was conducted, is that there should have been an application which allowed for user-testing. This could have led to a deeper user-oriented implementation and helped discover potential bugs in the solution. When testing the project, there should also have been tests where the system was in deployment. This would have lead to more accurate results in a real-world scenario.

Even though there have been meetings with the supervisor and experts within the field, there could have been more contact with doctors within the field to gain a deeper understanding and make further improvements.

Because of the reasons above, there has not been a field test to see if the system works in a real-world context, which means that there can be limitations. On the other hand, there has been thorough testing of the different throughput of the application, but in the author's opinion, it would have been helpful to have user tests to see that the system worked.

Possibly if the project was longer, there could have been developed a mobile application to go with the server-side solution, but this is mentioned more in chapter 8.

4 Requirements Specification

In this chapter, the requirements are defined by functional and non-functional requirements. To create these, the Volere Requirements Specification Templates was used to apply function requirements, together with user stories from SCRUM [1].

Some assumptions were also made in regards to the system;

- The mobile application is to be created by someone within the MI&T [2]
- We assume that the server is to be expanded with other master and Ph.D. students, and therefore need a general design for the solution

4.1 Source of the Requirements

Requirements are created to set a need for the system and defines how the system as a whole should be designed and work. According to the Volere template [73] there are different types of requirements as seen below.

1. Scrum `https://www.scrum.org/` (Visited 2019-05-10)
2. MI&T group at UiT `https://en.uit.no/forskning/forskningsgrupper/gruppe?p_document_id=343402` (Visited 2019-05-11)

- Functional requirements
- Non-functional requirements
- Project constraints
- Design constraints
- Project drivers
- Project issues

In this project, the main focus is on the functional and non-functional requirements. A functional requirement is defined as; "the fundamental or essential subject matter of the product. They describe what the product has to do or what processing actions it is to take." [73].

A non-functional requirement is defined by the Volere template as; "the properties that the functions must have, such as performance and usability. Do not be deterred by the unfortunate type name (we use it because it is the most common way of referring to these types of requirements)—these requirements are as important as the functional requirements for the product's success."

So in short, a functional requirement describes **what** the system should do, while non-functional requirements are constraints on **how** the system should do it.

4.2 Requirements

As a prerequisite for understanding and setting all functional requirements for the thesis, three different scenarios are explained to present the possible problems of individuals that may require a system as in this project.

4.2.1 Scenario one

Ola Nordmann is a man of 37 years with type one diabetes. He got the diagnosis several years ago and has lived with the condition for some time. To measure his blood glucose, and when to inject insulin, he monitors himself daily with a continuous glucose meter. He has to look at the meter and see where his levels are at throughout the day when he eats or wakes up. If Ola eats, he often looks at the number of calories and his blood glucose to know how much insulin to inject.

From this scenario, we can see that Ola needs a better way of monitoring his disease. By having an application in which he can record his blood glucose, food intake, and insulin dosage. Because of this, he can be confident that he injects himself with the correct amount of insulin at a given time.

4.2.2 Scenario two

Kari Nordmann has cancer. She is 26 and goes to immune therapy, and therefore has a weakened immune system. Because of this, she is easier susceptible to infectious diseases, and they may have a greater impact on her than the average individual. She has no way of knowing if there is an infectious disease that is currently spreading, and nowhere to look for cases where, for example, pneumonia is spreading.

This scenario showcases another usage of a system like EDMON. Even though she is not infected, she needs to know if there is a higher risk of getting infected.

4.2.3 Scenario three

A man of 42 years named John Doe has diabetes type 1. He has been infected with a new version of the swine influenza. For work, he drives a bus and each day he infects several people. Because John does not call in sick, he infects several people, which in turn infects even more. This gets to spread on a large scale before the current systems can catch it and in the end, it is too late. The influenza has spread throughout the country.

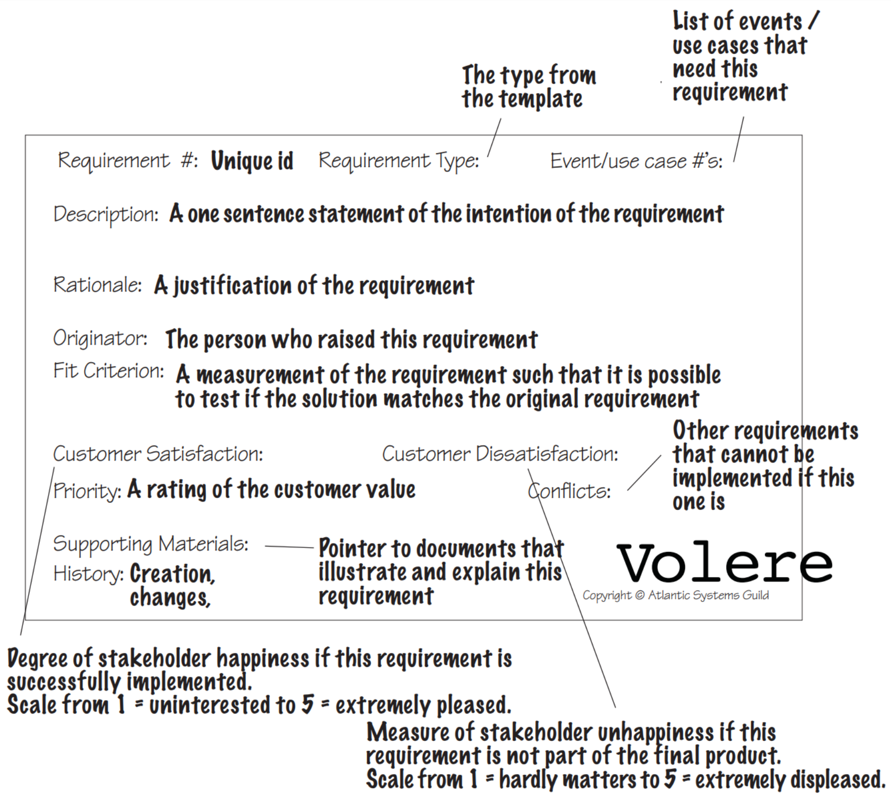
In this scenario, we highlight the warning mechanisms that are needed. It is also important to note that even though there are systems in place, we are still in need of lower latency between the first infection until an alarm tells us that there is an outbreak. By letting John Doe monitor himself and his vitals while sharing his data, there is a possibility to create a real-time monitoring system. This could let the government know that measurements have to be taken and that there is a risk of new infectious diseases spreading.

From the scenarios above, one can see that the use cases are both on a micro and macro level. Where the micro level is the individuals help to monitor diabetes type 1, together with monitoring the public health situation for vulnerable individuals. The macro level is to create an earlier warning system that helps to try and stop the spread of infectious diseases. Therefore, we need a tool that:

- Helps users self-monitor and self-record to improve their quality of life
- Let users see if there are immediate threats in the form of infectious diseases if they have a weakened immune system for example
- Has the functionality to look at macro events and use clustering algorithms to warn the right authorities if there is an outbreak by using real-time analytics

4.2.4 Functional requirements

By using the scenarios above as a foundation, the main features for the system were defined using the Volere template shown in Figure 4.1. All of the functional requirements are listed in Table 4.1. In the table, the source is assigned a number. One is equal to the author, while two is colleagues.



List of events / use cases that need this requirement
The type from the template
Requirement #: Unique id
Requirement Type:
Event/use case #'s:
Description: A one sentence statement of the intention of the requirement
Rationale: A justification of the requirement
Originator: The person who raised this requirement
Fit Criterion: A measurement of the requirement such that it is possible to test if the solution matches the original requirement
Customer Satisfaction:
Customer Dissatisfaction:
Other requirements that cannot be implemented if this one is
Priority: A rating of the customer value
Conflicts:
Supporting Materials:
Pointer to documents that illustrate and explain this requirement
History: Creation, changes,
Volere
Copyright © Atlantic Systems Guild
Degree of stakeholder happiness if this requirement is successfully implemented. Scale from 1 = uninterested to 5 = extremely pleased.
Measure of stakeholder unhappiness if this requirement is not part of the final product. Scale from 1 = hardly matters to 5 = extremely displeased.

Figure 4.1: Requirement shell from Volere template [73]

Table 4.1: Functional requirement specification

#	Description	Purpose	Source	Fit criteria	Priority
1	The system should provide an open API for application developers	Further development of the application	1	An API call can be done to the system	1
2	The system should be able to create new user accounts	Provide personal data for a user	1	The user is stored in the database	1
3	The system should be able to create medical records	Self-monitoring and self-recording for the user	1	The medical record is stored in the database.	1
4	The system should be able to create locations	Cluster detection in later projects	1, 2	The location is stored in the database.	1
5	The system should be able to permanently store data for the user	Permanently store the data for better calculations in the machine learning of EDMON	1, 2	The data is stored in the database.	1
6	The system should iterate through the data and run the mathematical operations on the data	Test the load on the system for later integration	1	Users sickness-status is changed every hour	2
7	The system should be able to do calculations on the user's blood glucose	Only use a given type of medical record and do calculations on it	1	The system is able to use algorithms on the BG data	2
8	The system should be able to fetch relevant data from a stable storage	The user can see its own data at all time to self-monitor	1	The API returns the wanted data	2
9	The system should provide authentication and access control for user's data	Ensure that one can only view, edit, create and delete data which belongs to you	1	Can fetch your own data, but others data will give an error	1
10	The system should encrypt all data sent between the user and the system	Confidential data and should not be shown to anyone	1	The data is sent over HTTPS	1

11	The system should be able to mark a user as sick	Give an indication to the user if there is need to worry. Give the system a way to use cluster detection algorithms in an area.	1	SicknessStatus in database changes.	3
12	The system should provide feedback to a user's request based on the result of a request	The application has to know if something is wrong when communicating	1	HTTP response codes and HTTP response body	2
13	The system should let the user store small videos on medical recording	Diary for users to look back and see how they felt at a medical recording. Can see correlations between the BG values, insuline values and how the user felt	1	Video is stored together with the medical record in the database	2
14	The system should protect the user's passwords	Ensure that if the passwords leak they are not reusable	1	Password is stored in database after being hashed and salted	2
15	The system should protect the user's identity	De-identify the user to ensure confidentiality	1	Email is removed after user has created their account	1

Some of the Volere template is excluded since it is deemed irrelevant because of the short time this project was developed on. However, as can be seen, we have created the functional requirements for the application.

4.2.5 Non-functional requirements

The non-functional requirements are defined by looking and the target group and usage of the system. We create an early detection system by monitoring individuals with type 1 diabetes and help them self-record and self-monitor.

Usability

The system is developed to create an easy gateway for later extension and usability. We aim to develop an easy to use and easy to understand API for the next developer and that can be used to create mobile applications for individuals with type 1 diabetes, together with warning systems for the average user and authorities.

Security

The data that is being stored and transferred is considered private health data. Therefore it is important to send the data encrypted and securely by identifying and authenticating the user. If there is any stolen medical data, it should be de-identified and secured before it is stored in the database. Even though health data is confidential, we aim to create an application that requires as little as possible authentication interaction form the user, to ensure that it will not feel like a burden to open it.

Performance & Throughput

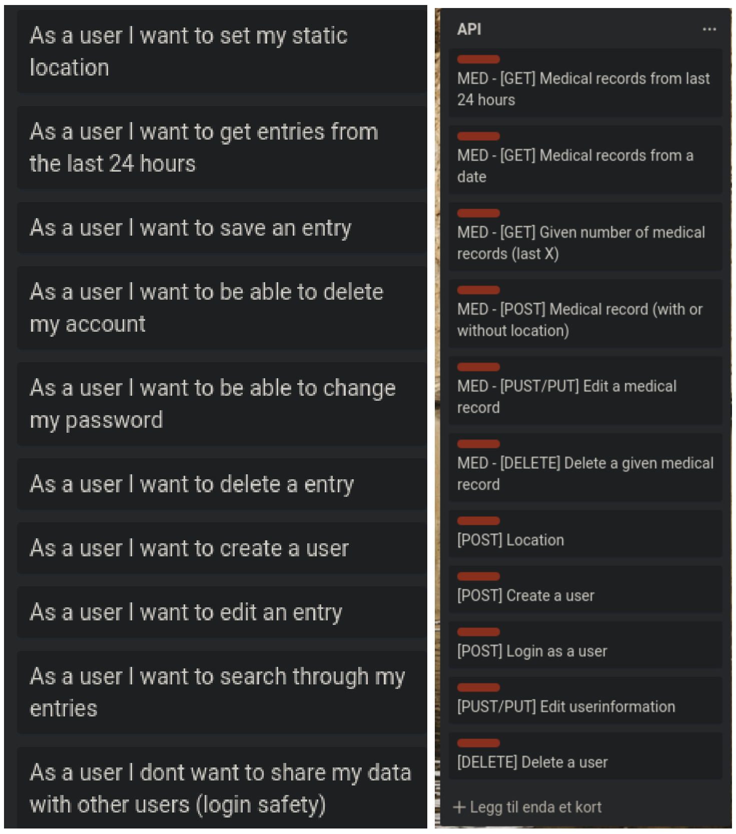
The system is defined as a real-time monitoring application for disease surveillance, which means that performance is an important aspect. We aim to create a system with high throughput, together with low latency on detecting disease spread in society. Throughput of data is also an important aspect if there are several thousand users of the product. We, therefore, aim to create a system that can handle the throughput of a relatively large user base.

Integrability & Extensibility

As mentioned before the system is under the EDMON project. Therefore we aim to provide ways to integrate new components. Together with integrating components, an important aspect is to add new features and develop a new version of the application. This should be ensured by using microservices and extensions in such forms.

4.3 User Stories

To keep track of the requirements a product backlog is created. A snapshot of the early developed user stories can be seen in Figure 4.2. A user story is an informal description of a requirement/feature of a system. This also helped identify which technical solutions that had to be in place.


As a user I want to set my static location

As a user I want to get entries from the last 24 hours

As a user I want to save an entry

As a user I want to be able to delete my account

As a user I want to be able to change my password

As a user I want to delete a entry

As a user I want to create a user

As a user I want to edit an entry

As a user I want to search through my entries

As a user I dont want to share my data with other users (login safety)

API

MED - [GET] Medical records from last 24 hours

MED - [GET] Medical records from a date

MED - [GET] Given number of medical records (last X)

MED - [POST] Medical record (with or without location)

MED - [PUST/PUT] Edit a medical record

MED - [DELETE] Delete a given medical record

[POST] Location

[POST] Create a user

[POST] Login as a user

[PUST/PUT] Edit userinformation

[DELETE] Delete a user

+ Legg til enda et kort

Figure 4.2: User stories from the product backlog

The user stories created technical tasks as can be seen in Figure 4.2. There are tasks that the author used to develop the system and keep track of which parts of the requirements that is done at a given time. In the figure, the technical tasks of the API are shown, where each task defines which component it belongs to, and what type of HTTP request it is. There is also a short description to tell the developer how the given task should be implemented.

5 Design

In this chapter, the design of the project is described. The design of the EDMON system as a whole, how each request is designed to interact with the system, together with each part of the system such as the middleware and database is explained in detail. The chapter showcases the iterations of the design, and which improvements, or alterations are done for each version.

5.1 Identified Features of the System

To design the system, all features are a result of the review conducted in Chapter 2. In addition, the functional and non-functional requirements from Chapter 4 are added. There are a few aspects of the design that is to be explained below; first of all, the design of the API. After that how the authentication and access control, together with the middleware. Then the design of the database will be explained in detail, where each table will be defined. The system as a whole is showcased with all parts showing their role, together with a version log where we showcase the differences from the start of the project until the end. Lastly, it is shown how one can create and add microservices to the system.

Since the thesis is written under the EDMON project, there are suggestions for the system in related literature from the project [7]. This can be seen in Figure 5.1. Here it is showcased that the design should include a mobile application which receives data from different devices and transfers data through the internet to a back-end server which handles blood glucose prediction and has stable storage. This is referred to as the "Computing unit" and is the foundation of this thesis. The design also includes a field with "End users". This includes that the users should be able to gather their data, share it with the family together with sharing the disease cases with the general public. It is also highlighted that the system should support desktops, laptops, and smartphones.

5.1.1 EDMON system design

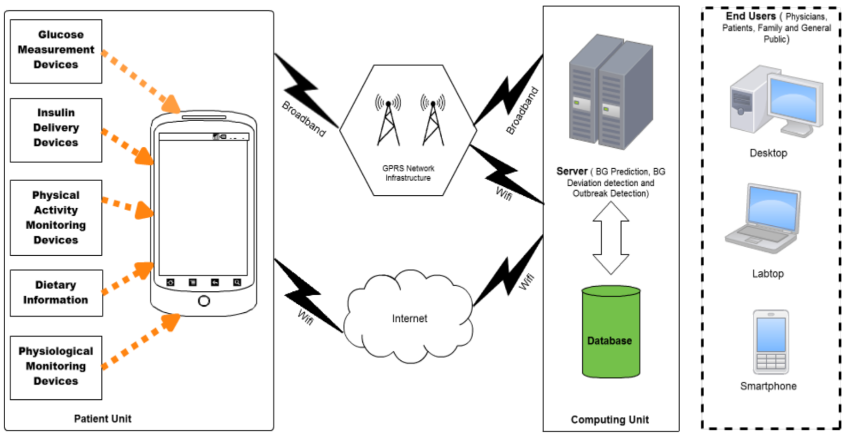

Glucose Measurement Devices
Insulin Delivery Devices
Physical Activity Monitoring Devices
Dietary Information
Physiological Monitoring Devices
Patient Unit
Broadband
GPRS Network Infrastructure
Wifi
Internet
Broadband
Wifi
Wifi
Server (BG Prediction, BG Deviation detection and Outbreak Detection)
Database
Computing Unit
End Users (Physicians, Patients, Family and General Public)
Desktop
Labtop
Smartphone

Figure 5.1: EDMON system design suggestion[7]

From the design in the EDMON papers, a few assumptions are made. The solution for this project had to be as general as possible to include all devices and types of frontend applications. The system should have an API which creators of applications can use, with standardization, security, and privacy in place.

5.2 Application programming interface

The application programming interface (API) is the connection interface of the application developers later in the EDMON project will use. Therefore the possibility to create and add more API end-points if needed is vital. Together with this, it has to support all functionality needed to create, edit, and delete information dependent on the individual's needs. The API in the project has an emphasis on the "user", defined as the user of the applications in this section. The user should have full control over their data, where they can choose what they want to share with the system. That being said, there should be an incentive for the user to share their data since it leads to more accurate calculations in the disease surveillance part of the system.

5.2.1 Data transfer design

The API is built with a RESTful design [1], where GET, PUT, POST and DELETE are the four types that are used to send all data. The data sent from a user to the system in JSON-format, as can be seen in Listing 5.1. A brief explanation of the different request types in the system is given below, but extensive documentation on how to use them is given in Appendix A.

HTTP body contents

While fetching (GET) and deleting data does not require any contents in the HTTP body, both editing and creating new data items in the database requires some form of input. This input is typed out as JSON and will vary from the request type one does. Below there will be given an example of each type of request while showcasing one requests HTTP body and design.

GET

The users' authentication information defines a get request to the system. The user sends a token that is specific to the user's saved token in the system. The data that a GET request retrieves is only the given users' health data. Because of this retrieving, for example, from the medical records, the user give the authentication token and the system returns the wanted data. This also includes the users' location(s).

1. RESTFul `https://restfulapi.net/` (Visited 2019-05-12)

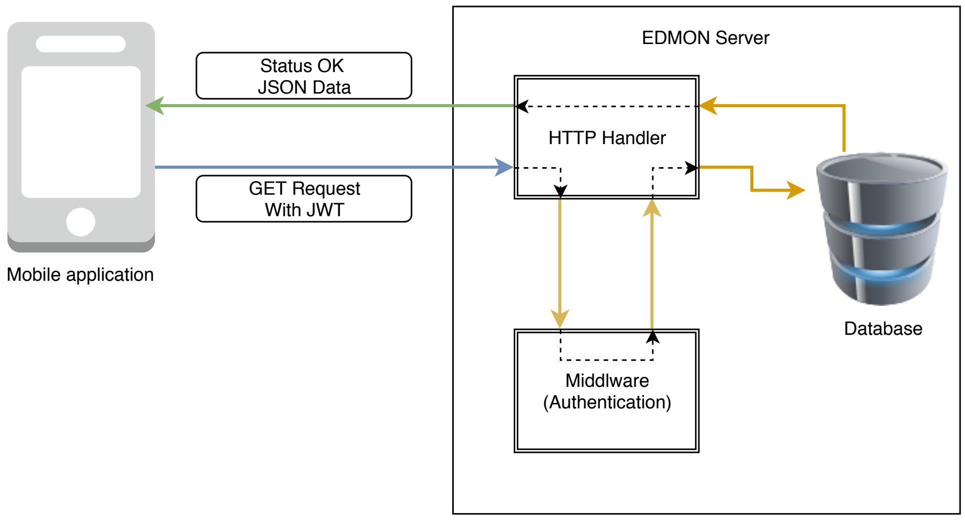


Figure 5.2: Example of a GET request to the system

As can be seen in Figure 5.2, a request of this type does not require any information besides the token. Since a request is done based upon the user, the system creates a session for the requester, fetch the user account if the credentials are correct, and execute the given query based on the GET call.

PUT

A PUT request, on the other hand, requires input from the user. This input is given as an HTTP body, where it should include the objects relevant information as described later in the chapter under the database design. This information should be given in JSON as can be seen in Listing 5.2. When a user sends a PUT request, it also has to contain the authorization token.

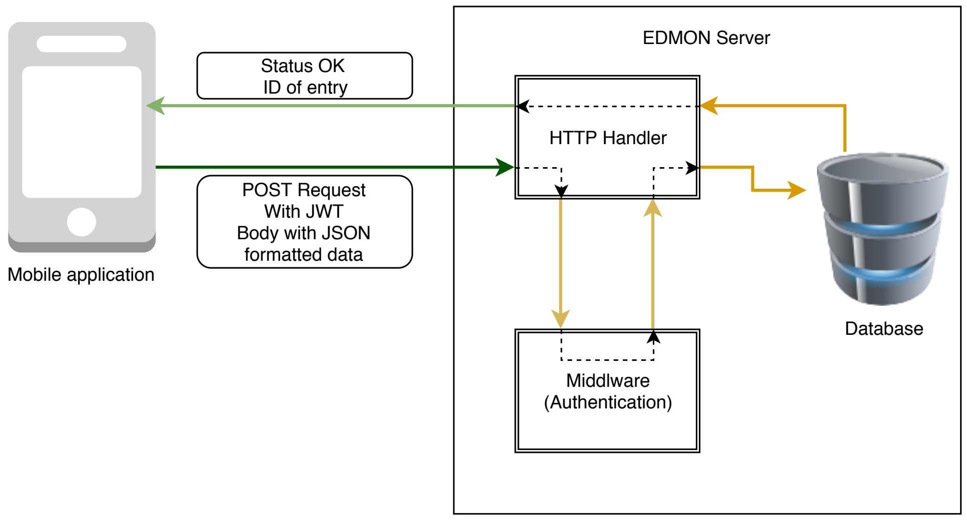


Figure 5.3: Example of a PUT request to the system

In Figure 5.3 one can see how a PUT is done from the user sending a request, to the server authenticating that the requester has access to edit the data until the user gets a response. There can be several responses dependent on what the user sends. Either the user can be blocked for lacking access. This happens if the authorization token is not correct or expired. Another issue could be that the data the user sent in the request body is not formatted correctly, or does not contain the correct information. How each request is formatted is specified in Appendix A.

POST

When sending data to the system, a POST request is executed. This request has to contain a body with the correctly formatted data, such as with the editing (PUT). The authorization token should also be included in the header so that the user is identified and only edit their own data. A POST request is the same as a PUT request, but when creating new data, the user does not provide the ID of the entry. See Figure 5.3 for a reference to how a PUT request is designed. To give the user feedback on the newly created element the system will return the ID created in the database to identify the object. This also gives the possibility to fetch the wanted item. The return value does vary a bit between the different POST requests.

DELETE

To delete an object from the system, one gives input on which object that is to be deleted by using the ID, such as the primary key. It is important to notice that if an object is deleted it is gone forever. As can be seen below in Figure 5.4 the user has to authenticate as with the previously stated examples, whereas the request URL has to contain the ID of the object to be deleted.

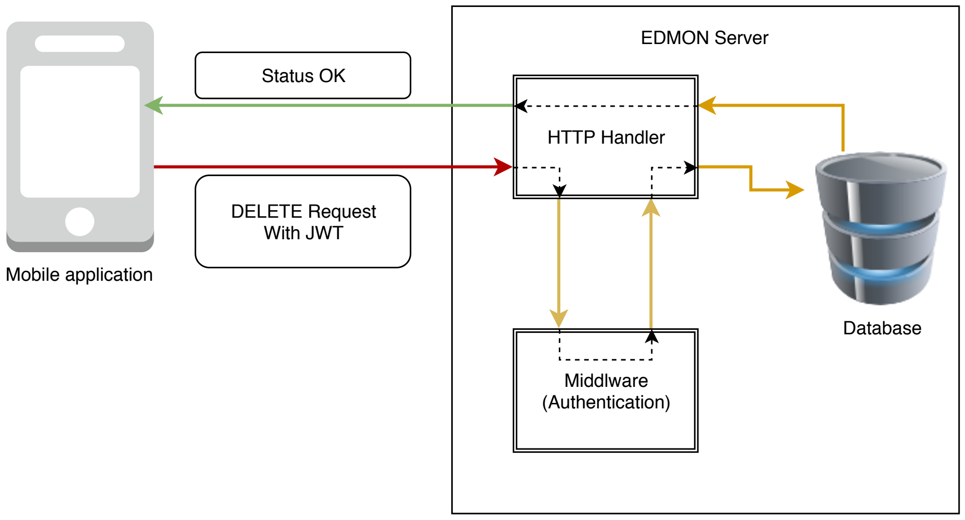


Figure 5.4: Example of a DELETE request to the system

5.2.2 Authentication and Access Control

Authentication and access control in the system is handled by JSON web tokens, as mentioned in Chapter 2. These tokens are session-based, which mean that as long as they are refreshed before the expiration by the application, a user does not need to log in using their username and password.

When a user signs up for the system, he or she has to provide a username, password, and an email address. The username and password are used to authenticate if they do not have an active token, while the email is removed because of de-identification. The original thought by using an email was to send a verification email to the user. This is a feature that is discussed later in the report. To create a user, an HTTP call to the `https://www.example.org/user/CreateUser` has to be done. The call has to contain the essentials which can be seen in Listing 5.1. The body of the HTTP has to contain the three values while being formatted as JSON. If the data is wrongfully formatted an HTTP status saying so is returned.

Listing 5.1: HTTP body contents when creating a user

```
{
    "username": "TestUsername",
    "email": "testuser@testemail.com",
    "password": "password123",
}
```

When the user wants to log in to the application, they have to provide their username and password. This is done by sending a POST request to `https://www.example.org/auth/login` which contains a body with the data in JSON format, as can be seen in Listing 5.2. This provides an easy route for developers to log in to the server and get a JWT in return.

Listing 5.2: HTTP body contents when creating a user

```

{
  "username": "TestUsername",
  "userid": "1",
  "password": "password123"
}
```

When a user has provided the correct login information, their identity has been verified, and the server therefore sends an answer to the application with the newly created token. It also includes an expiration date, which provides an easy way of knowing when one has to refresh the token. If this expiration is not met, one has to log in again, as stated above. In Listing 5.3, an example of a token that is returned from the server is provided. It is returned as JSON and as can be seen is has a field named token which holds the JSON web token for the user to authenticate with later on.

Listing 5.3: HTTP body contents when creating a user

```
{
    "code": 200,
    "expire": "2019-05-12T17:43:00+02:00",
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
       eyJleHAiOjE1NTc2NzU3ODAsIm9yaWdfaWF0IjoxNTU3
       NjY4NTgwLCJwYXNzd29yZCI6IiQyYSQwNCRQMXJ0bklwZ
       2ZWdVJYSmRzeDhSYkhlSUp3SjYwdFEwV1ouZjBLMVd2
       TVIyV05EQW85WnBLSyIsInVzZXJpZCI6OSwidXNlcm5
       hbWUiOiJMT0xPTE9MIn0.XAF3pej_0gwsNqoTUDC4
       VEqgu5kmtKYUoA6OiCXWc3U"
}
```

The token provided by the server is the users' way of authenticating all other HTTP requests that can be seen in Appendix A. The authentication functionality in the server is a part of the middleware.

5.3 Database design

To handle the stable storage of the medical data, a Postgres database is used [2]. The design of the database has a focus on being as general as possible to support later development of applications and exporting data from other applications into this system. The main goal is to support all types of medical data, while safely storing sensitive user information such as passwords.

When designing and creating functionality for the database, it was a strong focus on it to be usable for the EDMON project as a whole. Considerable time was spent creating functions for all use cases. It features the possibility to add, create, edit, and delete all of the data that is stored, together with functions in the system for executing these. All database access is done on individual users credentials, and as mentioned earlier in the chapter, this is based on JWT access control.

The database has three tables; User, Medical records, and locations. These are the basis of each recording. After a detailed explanation, the database as a whole is set into perspective, and the design is shown with foreign keys in place to illustrate how each table is dependant on the other.

5.3.1 User

The design of the user table can be seen in Table 5.1. It has a focus on the users' credentials, location, and medical records.

Firstly, each user should create a unique username. Since the data should be anonymous, the user should be de-identified as soon as possible. This is done by adding functionality to remove the email of the user. The reason for choosing to have an email in the database is that this email could be used for sending out a verification mail to the user, where the individual would verify that they are a real user and not a computer-generated one.

2. Postgres `https://www.postgresql.org/` (Visited 2019-05-12)

Table 5.1: Database: User table explanation

Name	***Type***	***Explanation***
uid	int, serial, PRIMARY KEY, NOT NULL	The identification of a user which is unique to the given user.
username	varchar(250) NOT NULL	An identifier chosen by the user that the system then uses to authenticate
email	varchar(250)	The email of a given user, which is removed when de-identifying
password	varchar(250) NOT NULL	Password chosen by the user, hashed & salted before stored
Tokens	text	JSON Web Token holder
Created	timestamp NOT NULL	Time of creation of the given user
activationcode	varchar(250)	Meant to be the code sent to the user for veryfing the email
staticlocation_id	int	The foreign key of the static location of a user
dynamiclocation_id	int	The foreign key of the dynamic location of a user
MedRecs	int, serial NOT NULL	Foreign key to the one-to-many relationship to medical records.
IsSick	boolean NOT NULL	Marks the user as sick or healthy. Updated each hour with EDMON algorithm.

Each user can have two different locations. Firstly, they can set a static location. The static location is defined as the hometown or where the individual lives. This is useful information for both the system and the user. Because the system knows where the user lives, it could send out warnings to the area where individuals live about current infectious disease threats. Likewise, for the user, it is relevant since it could give them the possibility to see if there is an outbreak in their city. The dynamic location is used as a current location for the user, for example, if the user is on holiday. This gives the opportunity to get feedback based on the current position or city where the user is located. Both of these locations is a foreign key to another table, which is explained later in this chapter.

A user sets a password when creating an account. The password is stored in the database for comparison when trying to log in to the system at a later time or to fetch the JSON web token. The passwords are hashed and salted [3] to ensure that if they are leaked to the public, it is still hard to get the passwords out in plaintext. As a result of this, each time a user tries to identify, their passwords are compared to the hashed and salted version stored in the database.

Under the creation of a user, a MedRecs ID is assigned. This ID is then later used to fetch the given users medical records from a table, talked about later on in this chapter. This acts as a foreign key, and if a user deletes all of the recorded medical data, this is used to identify which records belong to it. A detailed explanation of the fields is available in Table 5.1.

5.3.2 Medical Record

A medical record in the system is a recording of health-related data at a given time. It is defined by its LOINC code, which has been called HL7 in the database. Each recording is unique but has a foreign key identifier to the user that created it. This foreign key is the med_id as can be seen in Table 5.2.

The value of the recording is an integer which could be the heart rate per minute or the stress level recorded in integers from one to six. When a recording is done the timestamp is set by the database with the built-in now() functionality of Postgres [4]. This value is used when fetching the last day, hour or minutes of recordings, and offer the later developers in the EDMON project the possibility to fetch custom data based on time.

3. Adding salting to hashing `https://auth0.com/blog/adding-salt-to-hashing-a-better-way-to-store-passwords/` (Visited 2019-05-12)
4. Postgres Documentation `https://www.postgresql.org/docs/9.1/functions-datetime.html` (Visited 2019-05-12)

Table 5.2: Database: Medical table explanation

Name	*Type*	*Explanation*
id	intserialPRIMARY KEY	Primary key and identifier of medical records
med_id	int NOT NULL	Foreign key to a user table. Identifies whom the medical record belong to.
value	int, serial NOT NULL	A value of the medical records measurement given as an integer (For example heart rate)
HL7	varchar(255) NOT NULL	Standardization code for identifying the value. (For example LOINC)
timestamp	timestamp NOT NULL,	The time when the medical record was recorded
Location_id	int	Foreign key to the location where the medical record was recorded
recording	bytea	A diary in the form of a video created by the user to further document how the individual felt at the time of the recording

Each recording can have a location. This is needed since the system later runs diagnostics on a given area to determine if there is an infectious disease outbreak. It could also be used on an individual level to have an overview of where the user was. An example is if the user remembers being sick in Oslo. One could easily send a request and fetch data from the last trip to Oslo and see if it was correct.

Lastly, each medical record contains a recording. This recording is meant to be either a picture or a short video that the user wants to save as a diary of how the condition was, or how the user felt at the given recording. For example, the user could save a short video explaining why the readings are not typical. A fever could have occurred, and this his information could then be used by the user at a later point to explain why the readings are irregular.

5.3.3 Location

Both the user and medical record table contains a reference to a location. This location is a gathering of the latitude and longitude but does not give any information about anything else than this, as is seen in Table 5.3. The reason why this is put outside the other tables is both for having the possibility to change the format of the location to for example being the city name or the zip code, together with not storing information together. If all medical recordings are leaked, then there is no way to trace the information back to the user or the location of the recording, without having all the data in the database.

Table 5.3: Database: Location table explanation

Name	*Type*	*Explanation*
loc_id	serial primary key, NOT NULL	Foreign key to both user and medical record table. Also the identifier of a given location record
latitude	float	The latitude of the location
longitude	float	The longitude of the location

Relations between the tables

As all of the table's design are laid out, the relationships between them may not be clear. The database is based upon several foreign keys, which defines how each table interacts with each out. These can be seen in Figure 5.5

A user has two different foreign keys to the location table. The first is the static location, and the second is the dynamic location. These set the users positions at all times. The user can choose not to give their location, and in that case, there is not a location record in the database with a foreign key to the user table. While the user can set two different locations, it only has one medical record ID. This is set on creating the user and is persistent through all of the different medical recordings that are done by the given user.

The relation between a user and a medical record is a one-to-many relationship, which means that the user can have many medical records. The MedRecs foreign key in the user table has a reference to the med_id column in the medical record table.

The medical record has a foreign key to a location, which is optional to create. A medical record can have one location, which is to indicate where the user is located at the time of the recording.

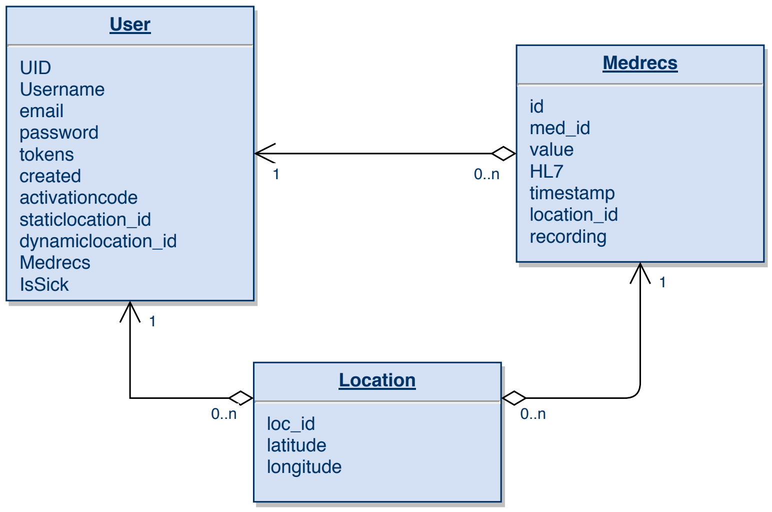


Figure 5.5: Database design

5.4 Version of the system

The project is created with an agile methodology [5] with iterative changes. The design of the system evolved throughout the project, and therefore, there are several version of the design. When starting with the project, a design for the foundation of the system was drawn, as can be seen in Figure 5.6.

This design only included a user together with a medical record without any information other than value. This design lacked a database and stored everything in memory. After meetings with the supervisors and reading relevant literature, the need for standardization was set to be in the next iteration. Together with this a prototype was developed using Golangs HTTP library [6]. This was created to showcase the strengths and weaknesses of the implementation while giving the author an overview of which needs a health-related system such as this needed. The design can be seen in Figure 5.7

The database was added to the system, together with expanding the current API to handle creating a user, and adding medical records to the given user. When this was done, the author realized the lack of support for the HTTP library when it comes to authentication, database access, and secure communication. Therefore the current implementation was set aside. With the gained knowledge of developing the prototype, a new implementation was started. The design

5. Agile manifest `https://agilemanifesto.org/` (Visited 2019-05-11)
6. Golang Http `https://golang.org/pkg/net/http/` (Visited 2019-05-11)

of this included creating a middleware which should support authentication of the user, together with a stronger tool for handling HTTP requests [7] that also allowed for easy integration of other libraries for authentication and access control.

In this design, the functionality of snaps was added. After reading the literature, it seemed like no other solution had this functionality, and it was added as a way to represent the mood or form of the user at a given time.

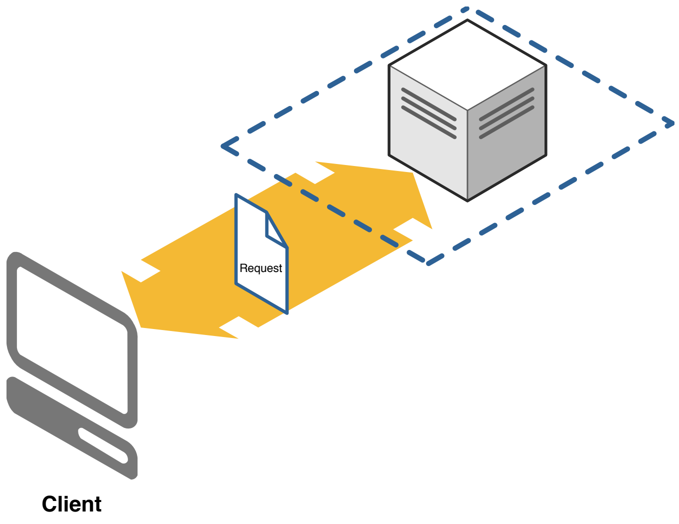


Figure 5.6: First design of the server

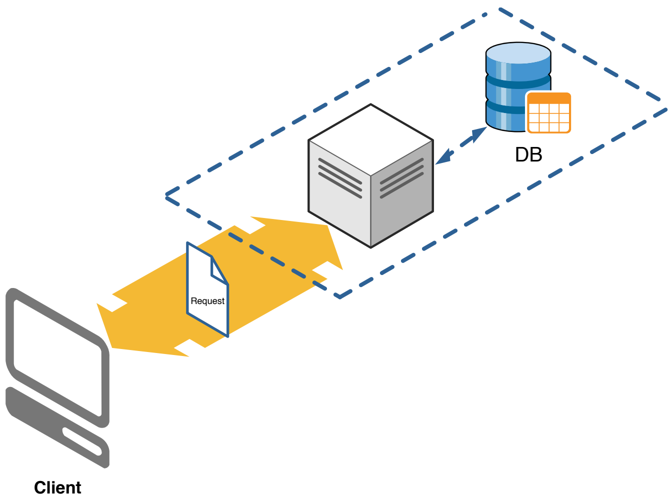


Figure 5.7: Second design of the system

7. Gin framework `https://github.com/gin-gonic/gin` (Visited 2019-05-11)

5.4.1 System design

The last design of the system contained everything from the API end-point, middleware to the stable storage and can be seen below in Figure 5.8. This is the current design of the system as a whole. It includes an API which the user can send requests to, a middleware which authenticates the user, together with stable storage in the form of a database.

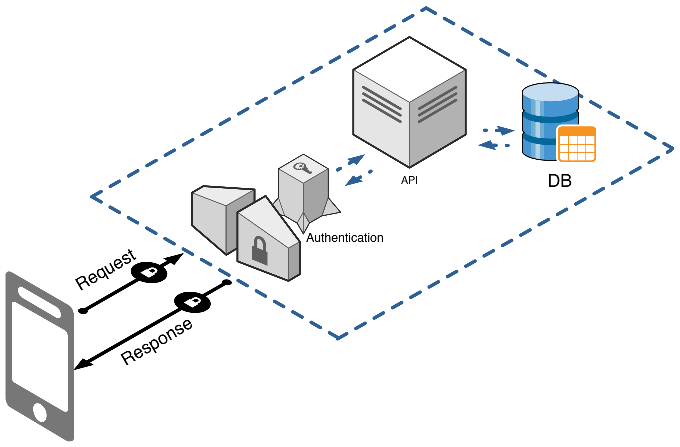


Figure 5.8: System design

Each request from a user goes to the systems through the authentication service which validates the token that is sent with the request. If a user has the correct authorization token the system will fetch the requested data from storage and send it back to the requester, as can be seen in Figure 5.9.

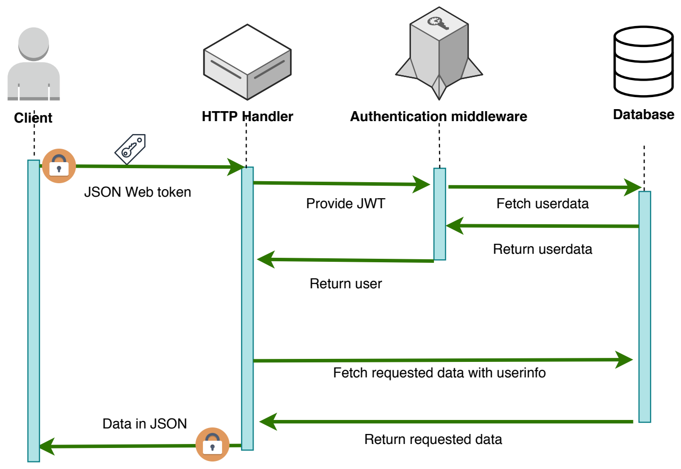


Figure 5.9: Access granted for user example

On the other hand, if the user can not provide the correct authorization token, the middleware denies further access and deny the users request of data as can be seen in Figure 5.10. Both when access is granted and denied showcases the workflow of the system as a whole.

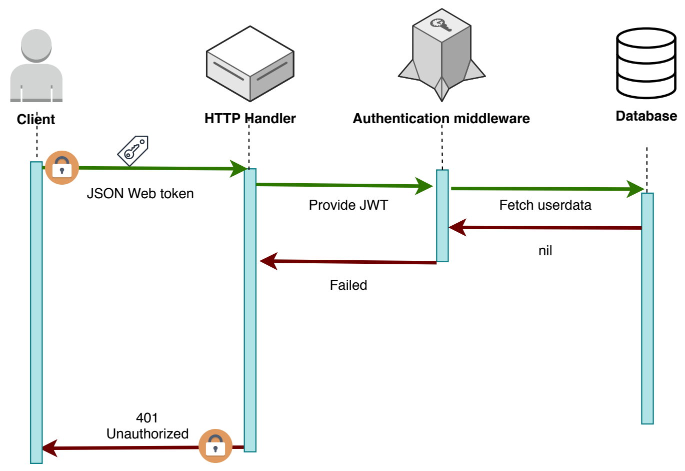


Figure 5.10: Access denied for user example

5.4.2 Extension of the system

Since the system is a part of a bigger project, EDMON, there has been an emphasis on creating a server solution which could easily be expanded with microservices or other projects in the MI&T group at the university. Together with maintainability and an emphasis on creating a code structure that could be understood at first glance, it was essential to create a system that could be used for creating the infection detection system for people with type 1 diabetes. The code is structured as in Table 5.4 and has the following folders to separate the functionality.

Table 5.4: Filesystem

Name	*Subfolders*	*Functionality*
api	apiauth	Create and set up the routes for medical records. URL: `https://www.example.org/auth/...`
	apilocation	Create and set up the routes for medical records. URL: `https://www.example.org/location/...`
	apimedical	Create and set up the routes for medical records. URL: `https://www.example.org/medrec/...`
	apiuser	Create and set up the routes for medical records. URL: `https://www.example.org/user/...`
createdb	-	Contains scripts to set up, tear down and seed the database with both real and fake data
db	-	Contains the database helper. Connects to, pings and creates a database session
middleware	-	Contains the middleware. JWT functionality and EDMON algorithm are placed here.
models	locations	The model of a location and how a location record is structured. Database functionality for everything about a location record
	medicalrecords	The model of a medical record and how a medical record is structured. Database functionality for everything about a medical record
	users	The model of a user and how a user's database table is structured. This is the main logic of the systems interaction with the database and a everything a user does is through this model

Because of this, the middleware was added. The middleware is meant to handle authentication, but also be where the system is expanded. For now, it includes the skeleton functionality for the algorithm to detect infectious diseases in people with type 1 diabetes, which is to be executed each hour.

Adding microservices and expanding the functionality of the system is discussed and explained in detail in Chapter 6.

6 Implementation

This chapter describes the implementation process of the thesis, together with the reasoning behind each choice of the design process. The dependency of the code is explained together with information about the reasoning behind the structure of the code.

6.1 Project Dependency

The project's components have dependencies to each other. In this section, these are explained, and the implementation of the project is showcased. Firstly the general components are touched upon, after that each component; from the API and models to the middleware, database and how to generate the certification keys. The author also highlights parts of the code, which may be of interest. Then an explanation of how to create and add microservices to the system are described.

6.1.1 General

There are several components in the system, each with a given purpose. As can be seen in Figure 6.1, all components have a dependency on another. In the system, a component is defined as each smaller part of the system. They are the building blocks. Each component is divided into smaller parts represented by the tables in the database. This division is done to create a clear and distinct difference where each function of each functionality.

When the system boots, the main function is in charge of setting up the correct settings, together with initializing the routes, middleware, and the goroutine for the infection detection system. Firstly a connection is created to the database, where an instance is saved for later usage. After that, the Gin framework [1] creates and sets up all the routes. The routes can be seen in Appendix A. As soon as the routes are initialized, the middleware is created, where the JWT middleware is set up to ensure authentication and access control of the routes.

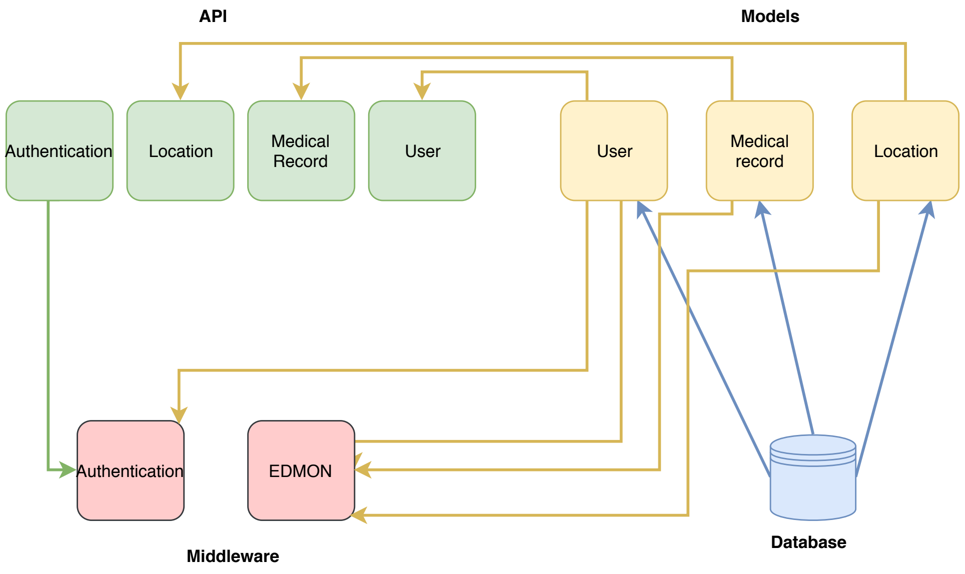


Figure 6.1: Code dependency

6.1.2 API

There are three different elements within the API component. Since there are different routes in the system, there is a need for having more than one element to separate which data we are working on. As can be seen in Figure 6.2, the three elements are the User, Medical records, and lastly the location. Each one of these has its grouping, which is to separate the URL-paths from each other. The paths are as follows:

- `www.example.org/user/...`

1. Gin framework `https://github.com/gin-gonic/gin` (Visited 2019-05-14)

- `www.example.org/medrec/...`
- `www.example.org/location/...`

The first part of the URL is to separate which data the user is working, by either creating new, editing existing data or deleting data. Each element is dependent on the models component to use the given model for a user, medical record, or a location. Each request is based upon a user. This user is from the user model and used to operate on throughout the session. For example, when fetching the static location from the user, the call made to the model is based upon a given user that is fetched when authenticating. This creates the opportunity to have a separate session for each user and divides the workload.

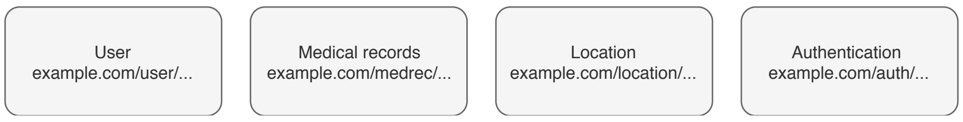


Figure 6.2: Components of the API

6.1.3 Models

As can be seen in Figure 6.3, the models are divided into three different elements. Even though the model is divided into smaller elements, the user model is the one which makes most of the requests to the database. The reason behind this is that each call to the database is done on a user, and not through a medical record or a location. Even though the calls are done through the user, the model for a medical record or a location is separate from the user. This was done to ensure the separation between the three, and to create a better overview of which table that is worked on at a given time.

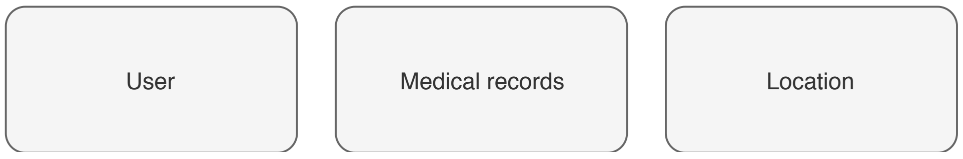


Figure 6.3: Components of the model

For each table in the database, there is also created a structure around each type to create a grouping of information. This structure holds the same information as the database tables, but it is represented in a way that the data can be worked on in the code. An example of one of these structures can be seen in Listing 6.1. The location structures shown contains an ID, together with the latitude and longitude represented as float64. This is identical to the table in the database, which allows us to fetch the data directly into an instance of the structures.

Listing 6.1: Location struct

```go
// LocRec is a struct to hold the location of a given user
type LocRec struct {
	ID        int     `json:"loc_id"`
	Latitude  float64 `json:"latitude"`
	Longitude float64 `json:"longitude"`
}
```

As can be seen in Listing 6.1 each value in the struct is represented as a JSON object. This allows us to bind an instance of the location to an incoming request by the user. The body of the HTTP request is bound to each value and look like in Listing 6.2. If the user does not want to edit a location, but rather create one, there is no need to give the location id.

Listing 6.2: Body of a location HTTP request

```json
{
    "loc_id": 1,
    "latitude": 69.649208,
    "longitude": 18.955324,
}
```

6.1.4 Database

As mentioned in Chapter 5 a Postgres database is used in the system. The database is an open-source relational database. After the design of the database, it is set up using SQL scripts such as the one seen below in Figure 6.4. Here the values and attributes of the table such as foreign keys and primary keys are defined. The SQL scripts set up all relationships, and after that, a small seeding of the database is done to test the features of the system through requests on the data in the system.

Figure 6.4: SQL script to set up location table

```
-- Table: location
    CREATE TABLE location (
    loc_id      serial  NOT NULL,
    latitude    float  NOT NULL,
    longitude   float  NOT NULL,
    CONSTRAINT location_pk PRIMARY KEY (loc_id)
);
```

The database component is connected to the system through the given models. As can be seen in Figure 6.1 it is not dependant on any other component, but rather it is a dependency. The system as a whole has a connection to the database, while each user that sends a request to the system has an own session to execute queries. Because of this, one could, in theory, change the database type as long as the tables have the same design. This makes migrating to another database possible if this is desired later in the EDMON project.

6.1.5 Middleware

As previously stated, the middleware has two jobs; to create a JWT authentication middleware, and serve the system with microservices.

The authentication part of the middleware has dependencies to the user model. These are used to authenticate the user through the database. The components get data from the API call, where a middleware has been set up to catch the data that a user sends as a claim. A claim is the data a user has to send to the server in order to identify. It is used to verify that the user is the person that is who he or she claims to be. The request is then forwarded.

The last functionality of the middleware is to run the EDMON algorithm. As per now, this is a simplified version of the intended EDMON algorithm. It is created to simulate the workload of a real-world example. The simplified algorithm fetches all the users, together with their medical records. After that, it sums the values together and divides them on each other. The simplified EDMON algorithm ensures that there is a small workload to the system that simulates the intended algorithm in later development.

6.1.6 System illustration

As all the components are presented, an example can be seen in Figure 6.5 on how they interact with each other. The example shown is a user that has logged in to the system and sends a request to fetch all of the medical records that are stored. As can be seen, when a user makes a request, it is first redirected to through the HTTP request to the middleware for authentication. If the user has provided the correct token access, the request goes through.

Since access is granted, the request goes through the user part of the API. Each request is done on the given user but is redirected to the model for medical records and fetch the medical records. The data is saved in an array and then returned to the user API where the data is parsed to JSON before the user gets the requested data back.

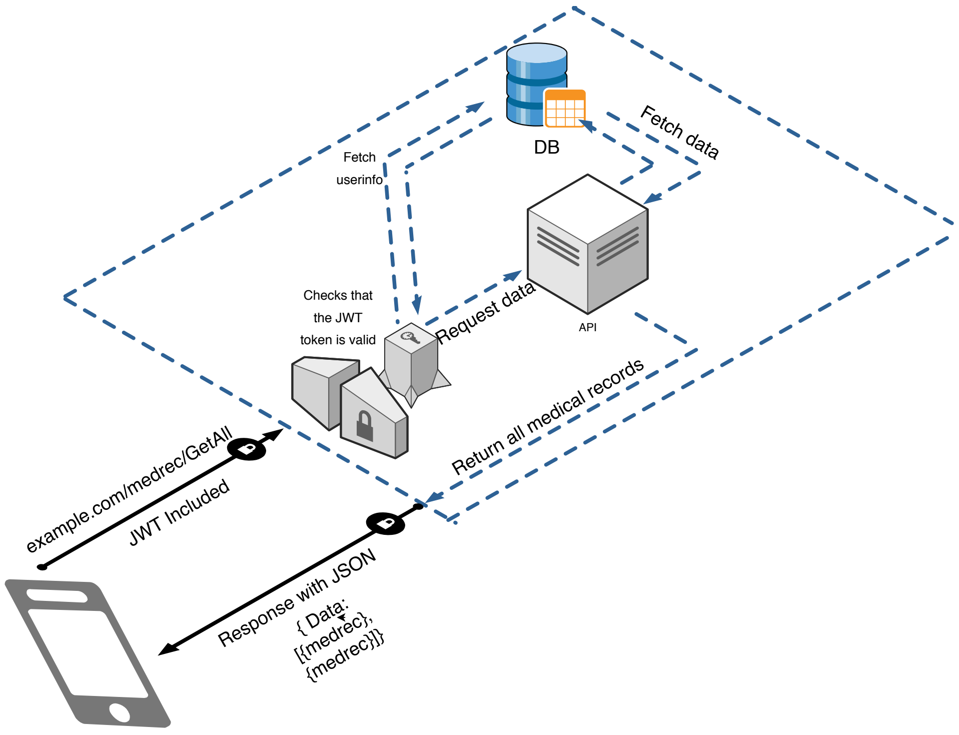

DB
Fetch data
Fetch userinfo
API
Checks that the JWT token is valid
Request data
Return all medical records
example.com/medrec/GetAll
JWT Included
Response with JSON
{ Data: [{medrec}, {medrec}]}

Figure 6.5: Example of a request to `www.example.org/medrec/GetAll`

7 Test and results

This section describes the results of the tests conducted. To read more about how they are done, see Chapter 3.

7.1 EDMON algorithm

The execution times of the simple algorithm that mimics the EDMON intended algorithm when using different amounts of medical records can be seen in Table 7.1. The time to execute grows with the number of elements that have to be fetched and analyzed. Together with this, we can see an increase in time when the algorithm is run on 2.9 million elements. In the table one can also see that the standard deviation is significant when calculating on 2.9 million elements. On the other hand, the EDMON algorithm uses around six seconds to execute with 290 000 medical records from 70 users.

Table 7.1: Times when executing the EDMON algorithm

# of records in DB	Mean (s)	Standard deviation
29 000	0.150	0.010
58 000	0.358	0.058
87 000	0.654	0.056
116 000	1.037	0.066
145 000	1.579	0.101
290 000	5.796	1.127
2 900 000	648.271	35.933

As seen in the Figure 7.1, the time spent executing stays fairly low at all times, and grows steadily with the number of records. The measurement with 2.9 million medical records is excluded from the graph for accurate readings. As can be seen in the graph, it is close to exponential growth in time depending on the number of elements.

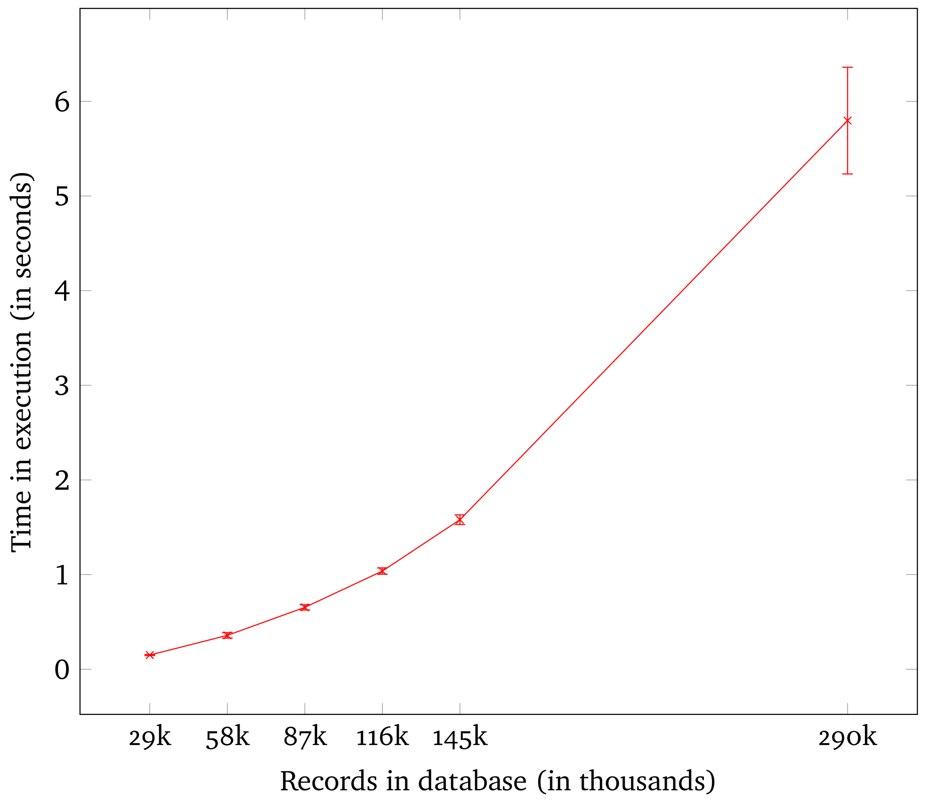


Figure 7.1: Server performance executing EDMON algorithm

7.2 Average times

When measuring the average times of the setup and tear down script, the results can be seen below in Table 7.3. As the table shows, the average time grows with the number of records that are created, and it leads to linear growth. The results can be seen in Figure 7.2, where the fact that the execution time of more elements is linear becomes clear. It is worth noting that for a user to create, fetch, and delete a total of 600 medical records with 300 locations, takes around 6.5 seconds when authentication is in place.

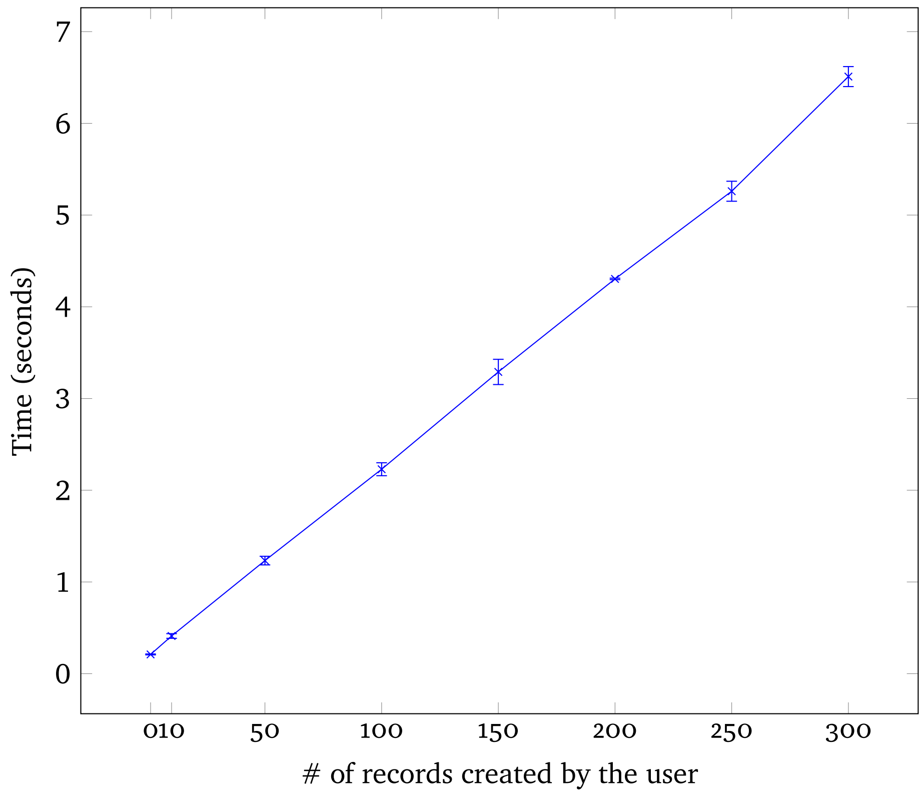


Figure 7.2: Creation of user, login, creating records and deleting time measurement

Table 7.3: Creation of user, login, creating records and deleting time measurement

# of created records	Mean (s)	Standard deviation
1	0.210	0.010
10	0.411	0.051
50	1.233	0.094
100	2.229	0.141
150	3.290	0.275
200	4.304	0.171
250	5.260	0.218
300	6.510	0.218

7.2.1 Concurrent requests

When testing concurrent requests with a different number of request on the various API calls, the mean and standard deviation is calculated. This can be seen in Table 7.5. The creation of a user has an average time of 0.01 seconds when only 10 other requests are sent, while it starts to get longer when the number of requests is higher. For example, with 5000 requests, user creation takes 0.77 seconds, which is a noticeable difference. This is the case with all type of requests in this test. It is worth noting that the standard deviation for both creating a user and authentication (login) gets higher with more concurrent users.

Table 7.5: Average time of concurrent requests

# of requests	Request	Mean (s)	σ (s)
10	create user	0.011369	0.003542
	log in	0.009945	0.003946
	create med. rec.	0.005407	0.000741
100	create user	0.073985	0.033374
	log in	0.057195	0.034586
	create med. rec.	0.051408	0.022681
1000	create user	0.675765	0.253713
	log in	0.329046	0.116809
	create med. rec.	0.347396	0.113942
5000	create user	0.773496	1.171404
	log in	0.564387	0.920808
	create med. rec.	0.469078	0.227804

The time when concurrently creating users is illustrated in Figure 7.3. Here one can see that the average time flattens out after 1000 concurrent requests while having a steep growth from 100 concurrent requests and upwards. It is also possible to see that the standard deviation is high when a lot of concurrent creation of users is executed.

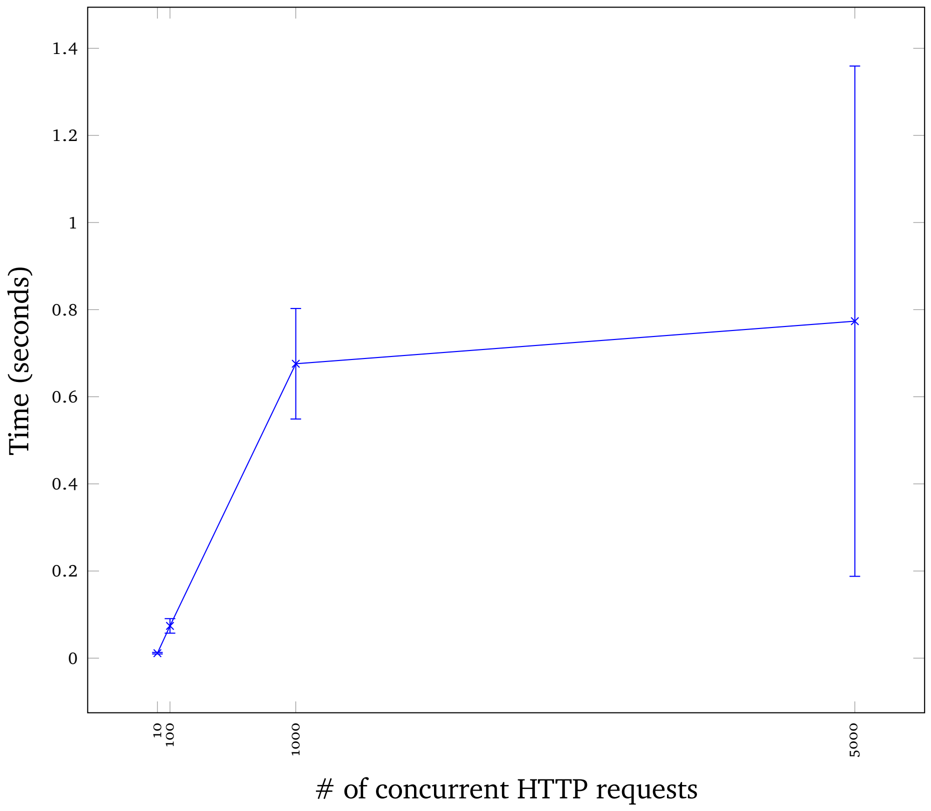


Figure 7.3: Average time when concurrently creating new users

As with the creation of users, the authentication of a user has a noticeably high standard deviation. It is worth noting that with both of these each request is still responded to within a second, but both graphs seem to follow the same pattern as seen in Figure 7.4 and Figure 7.5 From 100 concurrent requests and upwards there is a steep growth, and after this, the graph seems to even out more. When creating medical records, the growth is still fairly steep, while the standard deviation is a bit lower. Each request is still shorter than the average creation and authentication of a user.

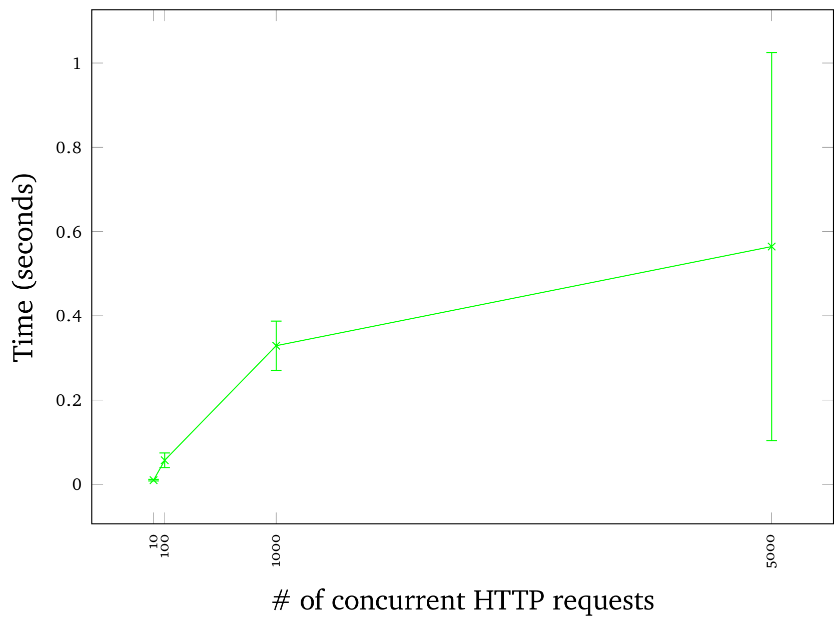


Figure 7.4: Average time with concurrent login's

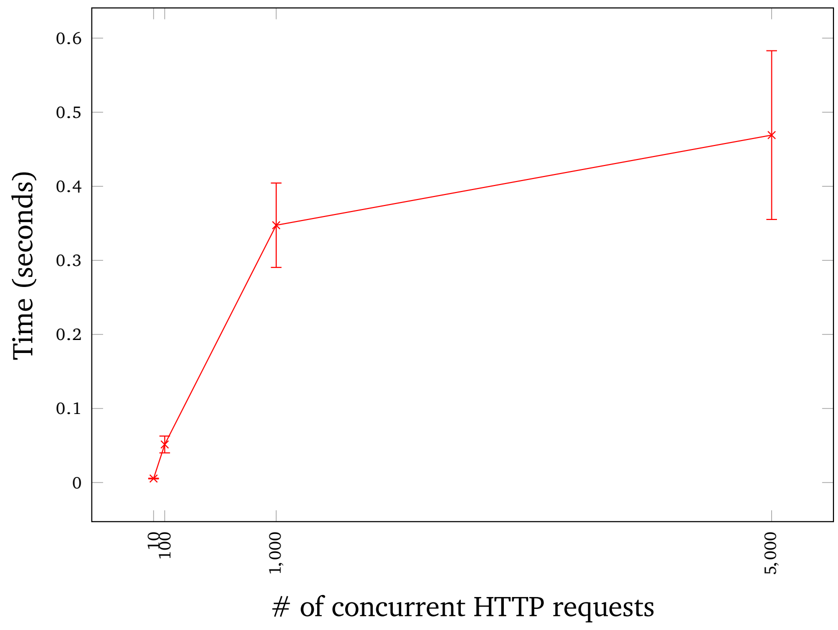


Figure 7.5: Average time when concurrently creating medical records

7.3 Profiling

In this subsection, the profiling results are shown. First of we start with the CPU and memory trace of the setup and tear down script with psrecord. As can be seen in Figure 7.6, the memory impact on the machine is constant at 18 MB, this because the memory that psrecord records is of the program itself, which is an 18 MB file. The CPU usage, on the other hand, lies around 10 to 15 % when setting up and tearing down 10 users, but it increases to 20 to 25 % when more data is sent.

7.3.1 CPU & Memory

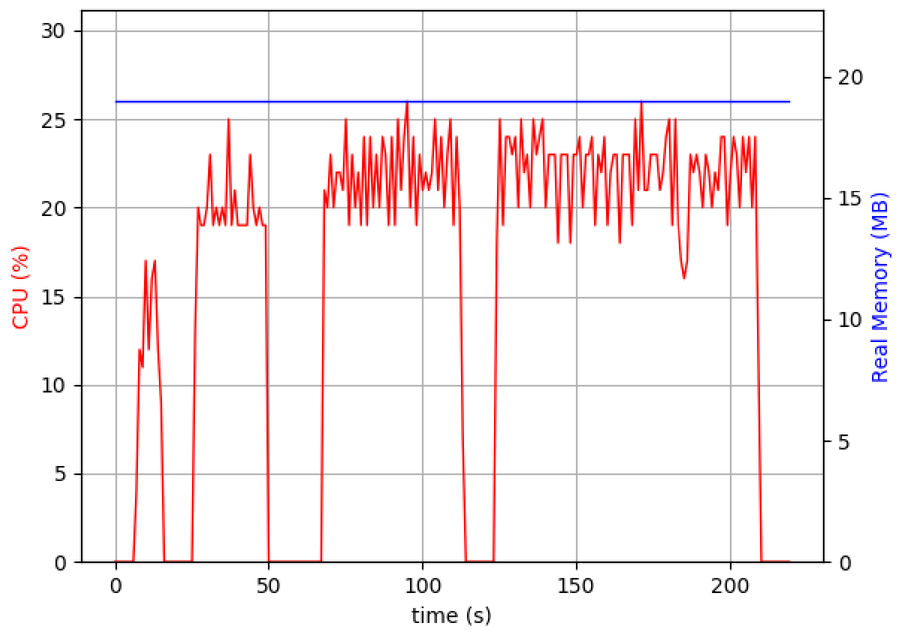


Figure 7.6: CPU and memory usage during setup and tear down script

On the other hand, as can be seen in Figure 7.7, the CPU usage in Linux tells us that there is 100% usage when 500 accounts are spamming the server with requests. The memory usage goes up to 40%, but then rapidly decreases as soon as the requests are over.

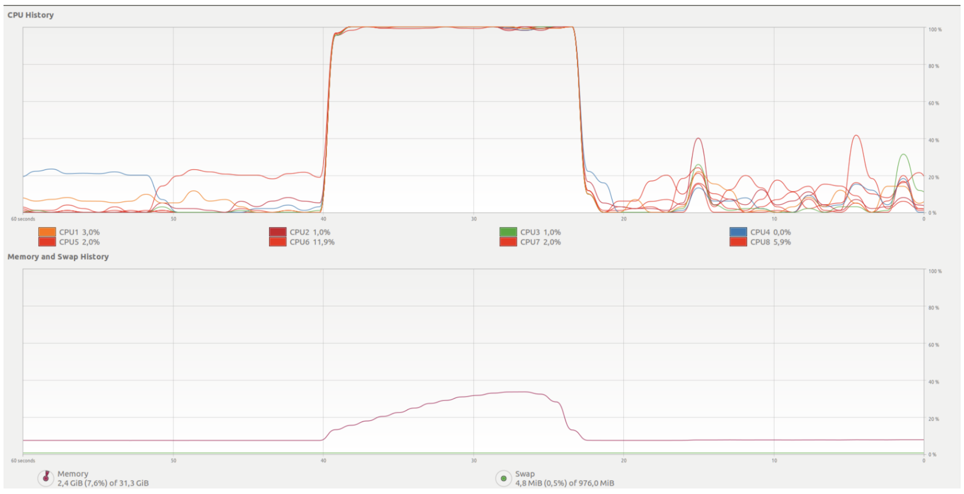


Figure 7.7: CPU & mem. when concurrently setting up and tearing down 500 accounts

7.3.2 Pprof results

The results of creating a user in the server by the pprof profiling tool can be seen in Figure 7.9. Here one can see that 50% of the time is spent encrypting passwords, while around 20% is spent on querying the database. In this case, the querying is writing to the database. When spamming the server for these 30 seconds, the system created 100.000 new users.

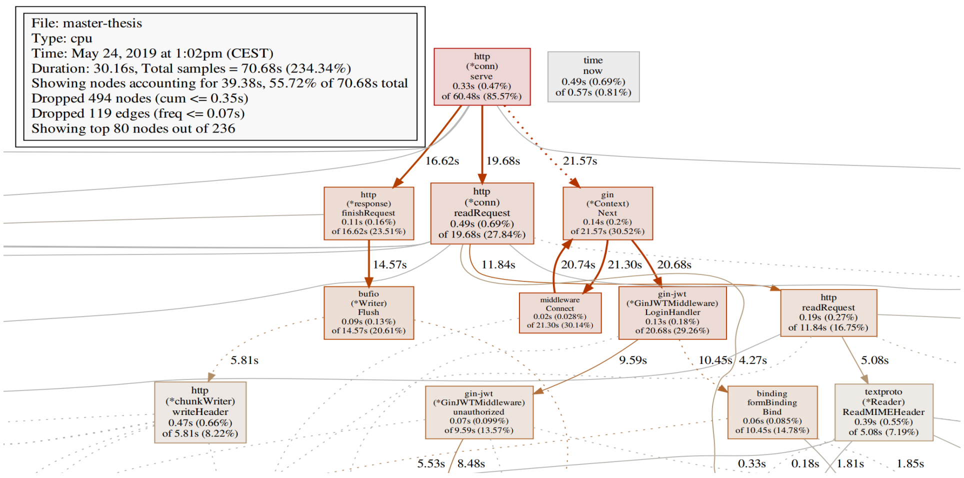

File: master-thesis
Type: cpu
Time: May 24, 2019 at 1:02pm (CEST)
Duration: 30.16s, Total samples = 70.68s (234.34%)
Showing nodes accounting for 39.38s, 55.72% of 70.68s total
Dropped 494 nodes (cum <= 0.35s)
Dropped 119 edges (freq <= 0.07s)
Showing top 80 nodes out of 236
http (*conn) serve 0.33s (0.47%) of 60.48s (85.57%)
time now 0.49s (0.69%) of 0.57s (0.81%)
http (*response) finishRequest 0.11s (0.16%) of 16.62s (23.51%)
http (*conn) readRequest 0.49s (0.69%) of 19.68s (27.84%)
gin (*Context) Next 0.14s (0.2%) of 21.57s (30.52%)
bufio (*Writer) Flush 0.09s (0.13%) of 14.57s (20.61%)
middleware Connect 0.02s (0.028%) of 21.30s (30.14%)
gin-jwt (*GinJWTMiddleware) LoginHandler 0.13s (0.18%) of 20.68s (29.26%)
http readRequest 0.19s (0.27%) of 11.84s (16.75%)
http (*chunkWriter) writeHeader 0.47s (0.66%) of 5.81s (8.22%)
gin-jwt (*GinJWTMiddleware) unauthorized 0.07s (0.099%) of 9.59s (13.57%)
binding formBinding Bind 0.06s (0.085%) of 10.45s (14.78%)
textproto (*Reader) ReadMIMEHeader 0.39s (0.55%) of 5.08s (7.19%)

Figure 7.8: Log in profiling

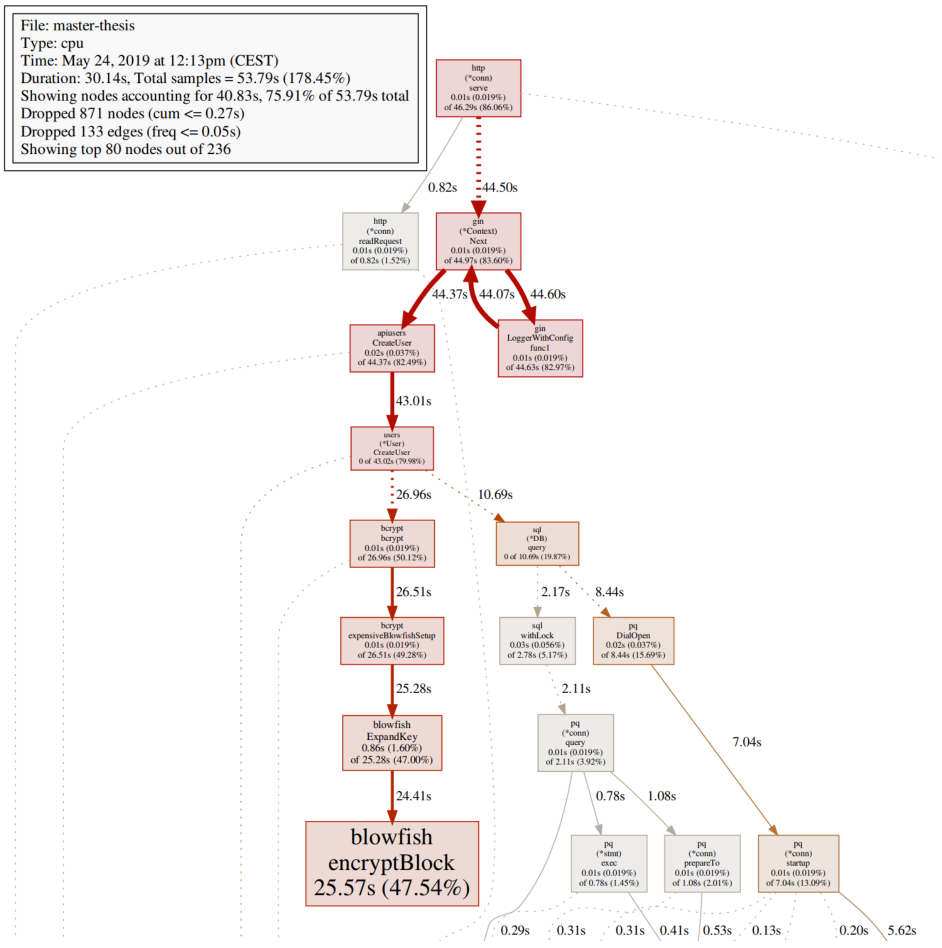

File: master-thesis
Type: cpu
Time: May 24, 2019 at 12:13pm (CEST)
Duration: 30.14s, Total samples = 53.79s (178.45%)
Showing nodes accounting for 40.83s, 75.91% of 53.79s total
Dropped 871 nodes (cum <= 0.27s)
Dropped 133 edges (freq <= 0.05s)
Showing top 80 nodes out of 236
http (*conn) serve 0.01s (0.019%) of 46.29s (86.06%)
0.82s
44.50s
http (*conn) readRequest 0.01s (0.019%) of 0.82s (1.52%)
gin (*Context) Next 0.01s (0.019%) of 44.97s (83.60%)
44.37s
44.07s
44.60s
apiusers CreateUser 0.02s (0.037%) of 44.37s (82.49%)
gin LoggerWithConfig func1 0.01s (0.019%) of 44.63s (82.97%)
43.01s
users (*User) CreateUser 0 of 43.02s (79.98%)
26.96s
10.69s
bcrypt bcrypt 0.01s (0.019%) of 26.96s (50.12%)
sql (*DB) query 0 of 10.69s (19.87%)
26.51s
2.17s
8.44s
bcrypt expensiveBlowfishSetup 0.01s (0.019%) of 26.51s (49.28%)
sql withLock 0.03s (0.056%) of 2.78s (5.17%)
pq DialOpen 0.02s (0.037%) of 8.44s (15.69%)
25.28s
2.11s
blowfish ExpandKey 0.86s (1.60%) of 25.28s (47.00%)
pq (*conn) query 0.01s (0.019%) of 2.11s (3.92%)
7.04s
24.41s
0.78s
1.08s
blowfish encryptBlock 25.57s (47.54%)
pq (*stmt) exec 0.01s (0.019%) of 0.78s (1.45%)
pq (*conn) prepareTo 0.01s (0.019%) of 1.08s (2.01%)
pq (*conn) startup 0.01s (0.019%) of 7.04s (13.09%)
0.29s
0.31s
0.31s
0.41s
0.53s
0.13s
0.20s
5.62s

Figure 7.9: Creating user profiling

As can be seen in Figure 7.8 the time spent is more spread out, where 27% of the execution time is spent reading the input by the HTTP library, and the Gin framework parsing spends around 30% using the JSON web token authentication login-handler. During the profiling, around 4 million authentications (logins) are accepted by the server.

Lastly, in Figure 7.10, it is clear that the most prominent time is to authenticate the user by using the JWT. The interaction with the GIN library is around 80 % of the servers execution time, where 80% of this time is spent authenticating and 20% is spent on writing the medical record.

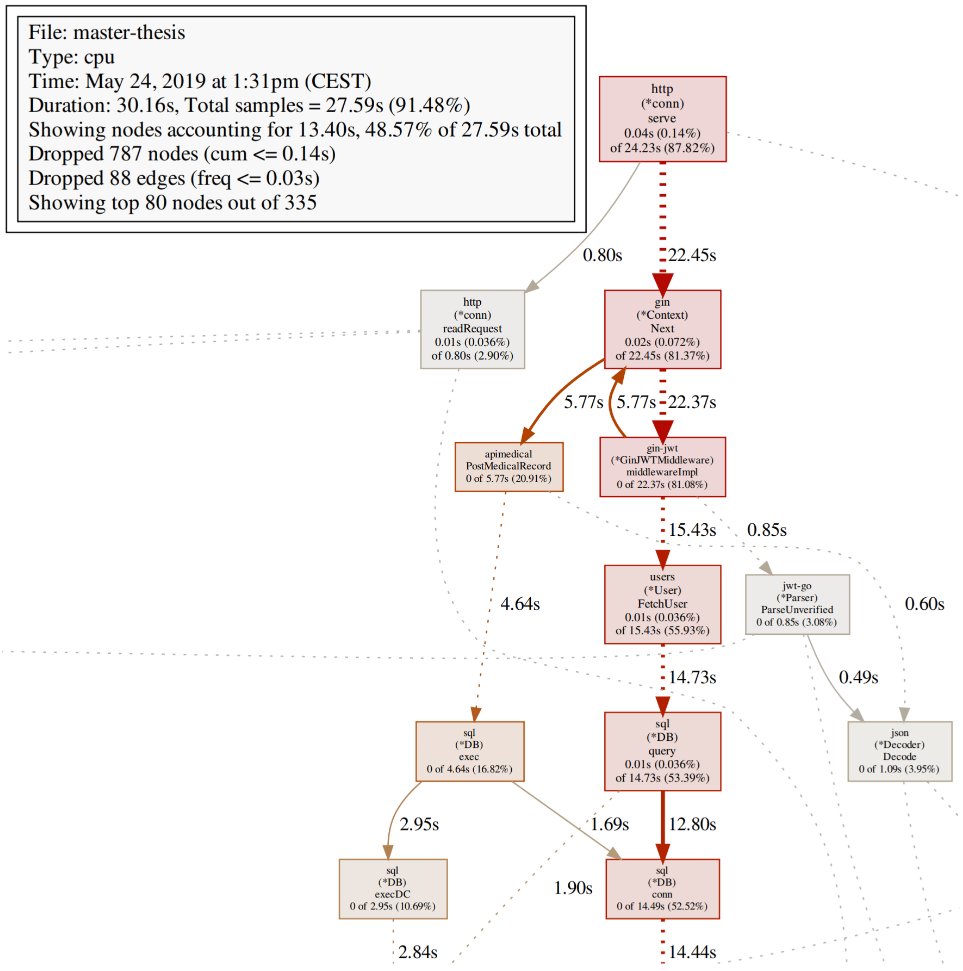

File: master-thesis
Type: cpu
Time: May 24, 2019 at 1:31pm (CEST)
Duration: 30.16s, Total samples = 27.59s (91.48%)
Showing nodes accounting for 13.40s, 48.57% of 27.59s total
Dropped 787 nodes (cum <= 0.14s)
Dropped 88 edges (freq <= 0.03s)
Showing top 80 nodes out of 335
http (*conn) serve 0.04s (0.14%) of 24.23s (87.82%)
0.80s
22.45s
http (*conn) readRequest 0.01s (0.036%) of 0.80s (2.90%)
gin (*Context) Next 0.02s (0.072%) of 22.45s (81.37%)
5.77s
5.77s
22.37s
apimedical PostMedicalRecord 0 of 5.77s (20.91%)
gin-jwt (*GinJWTMiddleware) middlewareImpl 0 of 22.37s (81.08%)
15.43s
0.85s
4.64s
users (*User) FetchUser 0.01s (0.036%) of 15.43s (55.93%)
jwt-go (*Parser) ParseUnverified 0 of 0.85s (3.08%)
0.60s
14.73s
0.49s
sql (*DB) exec 0 of 4.64s (16.82%)
sql (*DB) query 0.01s (0.036%) of 14.73s (53.39%)
json (*Decoder) Decode 0 of 1.09s (3.95%)
2.95s
1.69s
12.80s
sql (*DB) execDC 0 of 2.95s (10.69%)
1.90s
sql (*DB) conn 0 of 14.49s (52.52%)
2.84s
14.44s

Figure 7.10: Medical record creation profiling

7.4 Throughput

The results form Apache benchmark with 100 concurrent clients can be seen in Listing 7.1. Here we can see that the number of failed requests is 0, where all 10.000 requests went through and got a response. The server handled around 800 requests per seconds, and the mean time for each request is around 123 ms before it got a response from the server.

On the other hand, when having a concurrency level of 150 clients it shows that almost 300 requests sent to the server failed as seen in Listing 7.2 Even though the server seems to handle more requests per second at 1262 requests per second around 2% of the requests failed. We can also see that the time per request is around 118 ms and that all 15000 requests are completed during 11.877 seconds.

Listing 7.1: Apache Benchmark: 100 concurrent clients 10k requests

```
Concurrency Level:        100
Time taken for tests:     12.391 seconds
Complete requests:        10000
Failed requests:          0
Keep-Alive requests:      10000
Total transferred:        1520000 bytes
Total body sent:          2680000
HTML transferred:         60000 bytes
Requests per second:      807.06 [#/sec] (mean)
Time per request:         123.906 [ms] (mean)
```

Listing 7.2: Apache Benchmark: 150 concurrent clients with 15k requests

```
Concurrency Level:        150
Time taken for tests:     11.877 seconds
Complete requests:        15000
Failed requests:          297
Total transferred:        2302869 bytes
Total body sent:          4020000
HTML transferred:         109899 bytes
Requests per second:      1262.92 [#/sec] (mean)
Time per request:         118.773 [ms] (mean)
```

8 Discussion

In this section, the system features, implementation and test results will be thoroughly discussed. Firstly, an evaluation of the test and experiment results is discussed, together with the profiling results. Then the research questions in Chapter 1 is brought up again, where we look at the project as a whole and discuss if the goals and questions are answered. Lastly, an evaluation of what could be improved on is discussed, together with a section for future work that could be implemented.

8.1 Evaluation of results

In this section, the result of the test is to be evaluated. The test and experiment conduct an overview of the number of requests the server can handle, and a hint to how the server performs in regards to memory and CPU usage, together with its limitations. Each test will be discussed below. They are run enough times to get a representative sample of each test, then in case there are any huge outliers, some of them are removed after sorting. Because of this, a good average and standard deviation are calculated for all the requests that are tested. Even though there are enough runs of each test, one critique is to be made on the fact that they are run on a local machine with fairly limited hardware. Since both the server and the test framework is on the same machine, they have to share the resources, resulting in a possible shortcoming of hardware. Even though this shows some problems, there is a positive side with this. Since all test is on the same machine as the server, we eliminate the possibility of latency and loss and execute them under perfect conditions.

EDMON Algorithm

The EDMON algorithm performs fairly well on the system. As could be seen in Chapter 7, the number of records had an impact on the standard deviation, but the mean time of execution grew as expected. In the test, there was a problem with too many records for each run of the algorithm, but this seems to be a side effect of the algorithm loading everything into memory at once, and the data has to be swapped in and out of memory in several cases. This could be the reason for the huge variance in time spent executing.

The algorithm itself is supposed to run once per hour. After analyzing the data set, it seems as the average number of recording per hour for six weeks is around 4 recordings. This is a result of the data set containing around 29 000 records for 7 users over 6 weeks. This means that one has to have a user-base of over 700 000 users before the server reaches 2.9 million records within an hour. Moreover, it is still worth mentioning that even with 2.9 million records, the algorithm spent around 10-12 minutes executing. This results in idle time of 48-50 minutes each hour. While there is a lot of idle time, it is essential to remember that memory might become an issue when loading large data sets. Therefore there could have been created a pool of workers in the form of Goroutines, that fetches the users' data continuously and spreads the workload over an hour, instead of having a heavy algorithm run once per hour. This would also decrease the chance of running out of memory since there was a lot of data fetched by the EDMON algorithm, and therefore impacting the serving of users in real time.

Performance

When testing the performance of the server in regards to execution time when running the setup and tear down script described in Chapter 7, the result shows that the when using more records the graph grows linearly. This is a good sign since the user tries to authenticate, set up and tear down its data, sequentially. This showcases that if a user is without internet or loses its connection, the server is still able to handle the creation of several medical records from the same user, within a reasonable time frame. In this case, the user uploads 600 medical records, with 300 locations in under seven seconds, while the server is under a fair amount of traffic.

Under the testing of concurrent requests using the python script in combination with the bash script, one can notice that the requests are not running in true concurrency. The bash script sets up all the python scripts in memory and then executes them some at the time. This means that there is a period before they are run, and as a result, they seem to run almost sequentially. Even though this is the case, it highlights the throughput and time of execution for the given type of request. As can be seen in the graph, the time grows with a higher number of requests. This because the requests queue up and have to wait for a response from the server. As can be seen in the graph when trying 5000 concurrent requests, the standard deviation is a lot higher than with the previous tests. This may be because they have to wait a bit longer since the server is busy at the time of the request. Even though the standard deviation may seem high, the request time is still under 2 seconds at worst. This may be a bit too long and cause the user to get impatient, but it is acceptable since it is showcased under such a high load.

Profiling

Profiling is done during run-time to provide insight into how the system performs and to try and identify possible bottlenecks. When recording memory and CPU usage with psrecord, the memory is static at 18MB. The compiled program itself is 18 MB, so it seems as the library is not able to fetch the total usage of memory with a golang implementation, but rather only measures the program file in memory. On the other hand, it does capture CPU usage in percentage. With the setup and tear down script the CPU usage is stable at 25% when it is sequential. This is expected, together with the fact that the execution time is longer when there are a higher number of medical records created.

When using Linux's system monitor on the other hand, the CPU usage is at 100%. Since everything is run locally, this could indicate that setting up clients and request with the python script is highly demanding. One can not conclude that the CPU usage of the machine is purely the server. The memory usage is stable at around 40% at maximum, which indicates that there is still more memory to be used. If one assumes that 30% of the machine's memory is used to run the server, this is still not regarded as a high percentage. This may be because Golang offers garbage collection, together with freeing memory at the end of each request.

As can be seen in Chapter 7, when creating an account most of the time is spent encrypting the passwords before it is stored in the database. Even though this might not sound good, it is a necessary trade off that has to be made to ensure that the users' data is safe and private. If anyone where to get a hold of the password this may lead to the user losing control over several accounts since the password may be reused. In the time-span of 30 seconds, the server created 100 000 new accounts. This is not a real-life scenario, but it showcases that the server can handle creating more than a sufficient amount of users within a reasonable period.

When authenticating, most of the execution time goes to check that the password is correct and creating a JSON web token. Therefore most of the time is spent in the Gin library login handler. In 30 seconds, there were over 4 million authentications done, which showcases that the server can handle more than enough authentications in a given period. This is meant to be done once, and after it is done, one can refresh the token instead of login in again.

Lastly, the creation of medical records is profiled. Interestingly most of the execution time goes to authentication and checking that the JSON web token is correct and that it belongs to the user. Meanwhile, a relatively short amount of time is spent creating and writing the medical records to the database. In the 30 seconds, over 41 000 records are created. This is fewer than with creating a user and authenticating, but this could be because we have to ensure the security and privacy of a user before saving the data to the database. This is a trade-off one has to expect, especially since we do not want a user creating medical records or fetching medical records for other users. Therefore it is logical that most of the time is spent authenticating the JSON web token.

Stress test

To test the liability and maximal throughput of the server Apache benchmark was used to create a small denial of service attack essentially. As can be seen in Chapter 7, 100 concurrent clients sent 10 000 requests, which averaged to around 800 requests per second. This showcases that the server can receive a high amount of requests. With 150 concurrent clients, on the other hand, around 2% of the requests failed. This is a result of the default maximum number of clients connected to the Postgres database is 100. As the implementation stands per now, each user gets a connection to the database when the session is started. A drawback of the implementation is, therefore, showed to be the number of clients which concurrently can connect. This could be fixed with changing the default value to more clients, but a result of this is that the server requires more hardware in the form of CPU and memory. Another possible fix could be to re-write the database code to create a single connection to the database, but this could result in queuing and longer wait time for requests. Since this is a special case, and the server returns an HTTP status code to indicate if something went wrong, it is not seen as a huge drawback. It is also worth mentioning that if it is too many concurrent clients spamming the server with requests, the database will itself crash and need a restart.

8.2 Research questions

In this section the research questions from Chapter 1 is addressed. First, the subquestions are evaluated and discussed; thereafter the main research question is brought up and discussed.

Subquestion one

How can a system for disease surveillance be designed, and which drawbacks or advantages is there of the state of the art systems?

There are several systems today which offers ways to store medical data. As mentioned in Chapter 2, some of them has interoperability with hundreds of devices and offers solutions that integrate with electronic health records. Most of the huge technology companies in the world, such as Apple, Samsung, and Google, offer solutions such as these. Many of them now also offer storage and analysis on diabetes data and provide open API's for developers to integrate their devices to, together with fetching user data. They also include authentication and security.

Even though these seem like an ideal solution, there are a few drawbacks by using them. Firstly, all of the data that is stored is sent directly to a huge corporation, where the data is stored at locations that you have no control over. You also have no guarantee that the data is truly deleted, and not just put to cold storage when you want to erase all of your medical information. Lastly, since the data is in the hands of a huge corporation, there is the risk of the data being shared with 3rd parties that can sell your information. This could lead to insurance companies getting a hold of it and not giving you insurance because of a minor detail in your health history.

This thesis offers safe storage for user information at a known location within Norway. The users can fully trust that the data is not shared in any way, while their privacy and security are first in line. It also offers the possibility to store images or small videos with each medical record. The author did not find any real-time surveillance systems using diabetes data to create infectious detection in society. So while the current state of the art has a lot to offer regarding storing, scaling, and interoperability, the infectious disease detection part of the project is an original contribution.

Subquestion two

How can secure data storage for mobile applications for individuals with diabetes be created?

To secure the data from a mobile application for individuals with type 1 diabetes encryption, hashing and salting, and secure authentication is used on the server. Firstly HTTPS is used to encrypt the data sent from a user to the server. This provides the security of a man-in-the-middle attack on the server where the certificates provide authentication from the server. To ensure that the passwords from the user are safe, they are hashed and salted before adding them to the database. Because of this, there are no parts of the system that interacts with the password after a user is created.

To ensure the identity of a user when authenticating JSON web tokens are used. These provide a way to authenticate the user and is talked more about in the next sub question.

Subquestion three

How can user's authentication be ensured with the least amount of user-interaction in the front-end application?

To ensure the authentication of a user with the least amount of user-interaction, JSON web tokens is implemented on the server. Since users may feel like it is a hassle to log in using a username and password at all times when using the application, they provide a way of authentication as long as the application has saved the token. It eases up the threshold for the user to open the mobile application while ensuring that a user can not interact with other users data. As long as the token is refreshed before the expiration, it provides a way for the user to get the least amount of interaction when authenticating in the front-end application such as a mobile application.

Subquestion four

How can an electronic health system that is to be expanded by other researchers and students be designed?

Since the EDMON project is ongoing at the University of Tromsø and the server solution provided in the thesis is the start of a bigger project, it was an important aspect that the server was designed with expansion in mind. Therefore the code and server solution is built to provide support for microservices and uses a modular approach to ensure that the extension is possible. As the server stands a simple algorithm that mimics the EDMON intended algorithm runs each hour. All that is needed here is to change which values to fetch from the database and some of the mathematical functions done on them.

All functionality is split into different folders, and extensive information on how they depend on each other is given in Chapter 5 and Chapter 6. Golang was also chosen to create a code-base that is possible to add functionality to by using goroutines. A thorough documentation of the API calls can also be found in Appendix A.

Research question

"How can an electronic health system server to aid individuals with diabetes type 1, while creating a warning system of disease outbreak with security and privacy in mind, be designed?"

Throughout the work on the thesis, the author learned a lot from the state of the art review and used some of the state of the art technology as inspiration. Especially in security, together with standardization. The OhioT1DM data set provided insight into how real data would be and what was needed to standardize it. With the goals and design in mind, the goal was to create a general enough solution, using encryption and authentication. For an individual with type 1 diabetes, the most important aspect is to have a stable solution which can provide insight into the condition and aid in self-monitoring. Therefore a database was designed to support the storage of all types of data, where the mobile application is the only limitation. The subquestions provide insight into how everything is solved and a small discussion around each question.

8.3 Future work

This section covers the future work of the thesis, together with some of the planned work on the project going forward at the University of Tromsø.

Firstly, one of the main opportunities for the project is to create a mobile application which can collect data from individuals with type 1 diabetes. The mobile application will also provide a real-world test platform for the server, where the real throughput and performance is tested. To do this, one has to set up a certificate for HTTPS. A straightforward approach here would be to buy a domain and set up HTTPS through a service such as Let's Encrypt [1]. To ensure that all of the users' data is safe all data in the database could also be encrypted, not just the password. This could lead to problems in the performance of the server, so it would have to be tested and evaluated.

Secondly, the disease surveillance and infectious detection algorithm that is currently being worked upon in the MI&T group at the university could be implemented. By integrating it with the server, it could provide the population with a real-time warning system, and provide more accurate readings as more and more data is collected from the individuals with type 1 diabetes.

As mentioned earlier a project, to use the location and individuals marked as sick by the infectious disease detection algorithm is worked on. Together with a cluster detection algorithm that determines if there is a reason to worry of an infectious disease outbreak in a given area. By providing these two, a solution to show the infected areas such as Healthmap [46], can be created.

Lastly, a cooperation with the FullFlow [66] project, and implementing full FHIR support could lead to interoperability from the server to open EHR solutions such as at DIPS [26]. By creating a solution which supports this one would allow doctors, general practitioners, and health personnel a way to fetch the users data. The data could then be used in diagnostics by certified health personnel.

1. Let's Encrypt `https://letsencrypt.org/` (Visited 2019-05-26)

9 Conclusion

Throughout the thesis, several aspects of creating, designing and implementing a back end server for self-monitoring for individuals with type 1 diabetes, with the functionality to create a diseases surveillance and infection detection system, is designed and implemented. In the report, several aspects, such as security, privacy, and standardization, are discussed and used to design the system. There is a considerable emphasis on creating an expandable system where other researchers and students can build upon the solution with their work.

After testing the server implemented handles up to 1000 requests per second, depending on which type of request. From the OhioT1DM dataset with real user data, one can conclude that the requests per second are sufficient for a real-life application. A review of the state of the art technology is conducted, and several aspects from the applications found, are used to design and implement the system. This helped the author gain knowledge about the field and requirements of the thesis while providing insight into the current solutions.

The thesis contributes with a general back end solution with security and privacy in mind, where the code is structured for other researchers to add microservices. Going forward, the EDMON project is to implement the infection detection systems, together with cluster detection algorithms to create a real-time warning system for disease in the population. This thesis is the first building block in designing and creating the EDMON platform. The author could not find any other real-time monitoring system of individuals with type

1 diabetes, which provides the same functionality, such as the possibility to store videos with each medical record. The solution provides authentication, privacy, and security, through JSON web tokens. These give the user minimal interaction with the authentication part of a mobile application, where the application itself can refresh the token if used frequently. Moving forward, monitoring other chronic conditions such as chronic obstructive pulmonary disease and asthma is also of interest for the project.

Bibliography

[1] (2019). Idf diabeted atlas 8th edition, [Online]. Available: www.diabetesatlast.org (visited on 02/20/2019).

[2] (2019). Diabetesforbundet: Diabetes type 1, [Online]. Available: https://www.diabetes.no/om-diabetes/diabetes-type-1/ (visited on 03/12/2019).

[3] (2016). Global health observatory by the world health organization, [Online]. Available: https://www.who.int/gho/mortality_burden_disease/en/ (visited on 02/20/2019).

[4] (2019). World health organization, [Online]. Available: http://www.who.int (visited on 04/19/2019).

[5] (2018). Diabetes fact sheet by the who, [Online]. Available: https://www.who.int/en/news-room/fact-sheets/detail/diabetes (visited on 04/19/2019).

[6] (2019). The global diabetes community, [Online]. Available: https://www.diabetes.co.uk/controlling-type1-diabetes.html (visited on 03/12/2019).

[7] A. Z. Woldaregay, E. Årsand, A. Giordanengo, D. Albers, L. Mamykina, T. Botsis, and G. Hartvigsen, "EDMON - a wireless communication platform for a real-time infectious disease outbreak detection system using self-recorded data from people with type 1 diabetes," *15th Scandinavian Conference on Health Informatics SHI2017*, p. 7, 2017.

[8] P. Kostkova and P. Kostkova, "A roadmap to integrated digital public health surveillance: The vision and the challenges," p. 7,

[9] R. Heffernan, F. Mostashari, D. Das, A. Karpati, M. Kulldorff, and D. Weiss, "Syndromic surveillance in public health practice, new york city," *Emerging Infectious Diseases*, vol. 10, no. 5, pp. 858–864, May 2004, ISSN: 1080-6040, 1080-6059. DOI: 10.3201/eid1005.030646. [Online]. Available: http://wwwnc.cdc.gov/eid/article/10/5/03-0646_article.htm (visited on 05/08/2019).

[10] A. Fouillet, N. Fournet, N. Caillère, A. Musset, L. Mercier, C. Durand, C. Caserio-Schönemann, and L. Josseran, "SurSaUD® software: A tool to support the data management, the analysis and the dissemination of results from the french syndromic surveillance system," *Online Journal of Public Health Informatics*, vol. 5, no. 1, Apr. 4, 2013, ISSN: 1947-2579.

[Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3692763/ (visited on 05/08/2019).

[11] T. B. Murdoch and A. S. Detsky, "The inevitable application of big data to health care," *JAMA*, vol. 309, no. 13, p. 1351, Apr. 3, 2013, ISSN: 0098-7484. DOI: 10.1001/jama.2013.393. [Online]. Available: http://jama.jamanetwork.com/article.aspx?doi=10.1001/jama.2013.393 (visited on 05/09/2019).

[12] B. K. Smith, H. Nachtmann, and E. A. Pohl, "Improving healthcare supply chain processes via data standardization," *Engineering Management Journal*, vol. 24, no. 1, pp. 3–10, Mar. 2012, ISSN: 1042-9247, 2377-0643. DOI: 10.1080/10429247.2012.11431924. [Online]. Available: http://www.tandfonline.com/doi/full/10.1080/10429247.2012.11431924 (visited on 05/09/2019).

[13] R. L. Wears, "Standardisation and its discontents," *Cognition, Technology & Work*, vol. 17, no. 1, Feb. 2015, ISSN: 1435-5558, 1435-5566. DOI: 10.1007/s10111-014-0299-6. [Online]. Available: http://link.springer.com/10.1007/s10111-014-0299-6 (visited on 05/09/2019).

[14] (2019). Health level 7, [Online]. Available: https://www.hl7.org/about/index.cfm?ref=nav (visited on 05/05/2019).

[15] (2019). International society for disease surveillance, [Online]. Available: https://www.healthsurveillance.org/ (visited on 02/04/2019).

[16] (2019). International society for disease surveillance - about page, [Online]. Available: https://www.healthsurveillance.org/page/About (visited on 02/04/2019).

[17] M. Meingast, T. Roosta, and S. Sastry, "Security and privacy issues with health care information technology," in *2006 International Conference of the IEEE Engineering in Medicine and Biology Society*, Aug. 2006, pp. 5453–5458. DOI: 10.1109/IEMBS.2006.260060.

[18] A. Appari and M. E. Johnson, "Information security and privacy in healthcare: Current state of research," *International Journal of Internet and Enterprise Management*, vol. 6, no. 4, p. 279, 2010, ISSN: 1476-1300, 1741-5330. DOI: 10.1504/IJIEM.2010.035624. [Online]. Available: http://www.inderscience.com/link.php?id=35624 (visited on 05/09/2019).

[19] (2019). Gdrp health data, [Online]. Available: https://edps.europa.eu/data-protection/our-work/subjects/health_en (visited on 02/04/2019).

[20] (2019). Datatilsynet, [Online]. Available: https://www.datatilsynet.no/ (visited on 03/12/2019).

[21] Helseutvalget, *Et nytt system for enklere og sikrere tilgang til helsedata*, 2016. [Online]. Available: https://www.regjeringen.no/no/dokumenter/et-nytt-system-for-enklere-og-sikrere-tilgang-til-helsedata/id2563907/.

[22] (2019). Microservices, [Online]. Available: https://microservices.io/ (visited on 04/19/2019).

[23] (Jun. 2017). Diabetes - fhi, [Online]. Available: https://www.fhi.no/nettpub/hin/ikke-smittsomme/diabetes/ (visited on 02/04/2019).

[24] P. J. Denning, D. E. Comer, D. Gries, M. C. Mulder, A. Tucker, A. J. Turner, and P. R. Young, "Computing as a discipline," *Communications of the ACM*, vol. 32, no. 1, pp. 9–23, Feb. 1, 1989, ISSN: 00010782. DOI: 10.1145/63238.63239. [Online]. Available: http://portal.acm.org/citation.cfm?doid=63238.63239 (visited on 05/11/2019).

[25] N. Menachemi and T. H. Collum, "Benefits and drawbacks of electronic health record systems," *Risk Management and Healthcare Policy*, vol. 4, pp. 47–55, May 11, 2011, ISSN: 1179-1594. DOI: 10.2147/RMHP.S12985. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3270933/ (visited on 05/07/2019).

[26] (2019). Dips, [Online]. Available: https://www.dips.com/no (visited on 02/04/2019).

[27] (2019). What is openehr, [Online]. Available: https://www.openehr.org/about/what_is_openehr (visited on 03/07/2019).

[28] (2019). Healthvault, [Online]. Available: https://international.healthvault.com (visited on 03/07/2019).

[29] (Oct. 2018). Process for becoming a healthvault solution provider, [Online]. Available: https://docs.microsoft.com/en-us/healthvault/publishing/become-a-solution-provider (visited on 02/04/2019).

[30] (2019). Nightscout, [Online]. Available: http://www.nightscout.info/ (visited on 02/04/2019).

[31] (2019). Mysugr application, [Online]. Available: https://mysugr.com/ (visited on 10/05/2019).

[32] E. Arsand, R. Varmedal, and G. Hartvigsen, "Usability of a mobile self-help tool for people with diabetes: The easy health diary," in *2007 IEEE International Conference on Automation Science and Engineering*, Sep. 2007, pp. 863–868. DOI: 10.1109/COASE.2007.4341807.

[33] (2019). Apple health kit, [Online]. Available: https://developer.apple.com/healthkit/ (visited on 02/04/2019).

[34] (2019). Google fit, [Online]. Available: https://www.google.com/fit/ (visited on 02/04/2019).

[35] (2019). Samsung health, [Online]. Available: https://www.samsung.com/no/apps/samsung-health/ (visited on 02/04/2019).

[36] (2019). Carezone, [Online]. Available: https://carezone.com/home (visited on 02/04/2019).

[37] (2019). Nomoreclipboard, [Online]. Available: https://nomoreclipboard.com/ (visited on 02/04/2019).

[38] K. Hope, "Syndromic surveillance: Is it a useful tool for local outbreak detection?" *Journal of Epidemiology & Community Health*, vol. 60, no. 5, pp. 374–374, May 1, 2006, ISSN: 0143-005X. DOI: 10.1136/jech.2005.035337. [Online]. Available: http://jech.bmj.com/cgi/doi/10.1136/jech.2005.035337 (visited on 05/08/2019).

[39] I. Rodríguez-Rodríguez, M.-Á. Zamora-Izquierdo, and J.-V. Rodríguez, "Towards an ICT-based platform for type 1 diabetes mellitus management," *Applied Sciences*, vol. 8, no. 4, p. 511, Apr. 2018. DOI: 10.3390/app8040511. [Online]. Available: https://www.mdpi.com/2076-3417/8/4/511 (visited on 08/05/2019).

[40] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (IoT): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645–1660, Sep. 2013, ISSN: 0167739X. DOI: 10.1016/j.future.2013.01.010. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0167739X13000241 (visited on 05/10/2019).

[41] M. Verma, K. Kishore, M. Kumar, A. R. Sondh, G. Aggarwal, and S. Kathirvel, "Google search trends predicting disease outbreaks: An analysis from india," *Healthcare Informatics Research*, vol. 24, no. 4, pp. 300–308, Oct. 2018, ISSN: 2093-3681. DOI: 10.4258/hir.2018.24.4.300. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6230529/ (visited on 05/07/2019).

[42] E. D. Karimuribo, E. Mutagahywa, C. Sindato, L. Mboera, M. Mwabukusi, M. Kariuki Njenga, S. Teesdale, J. Olsen, and M. Rweyemamu, "A smartphone app (AfyaData) for innovative one health disease surveillance from community to national levels in africa: Intervention in disease surveillance," *JMIR Public Health and Surveillance*, vol. 3, no. 4, e94, Dec. 18, 2017, ISSN: 2369-2960. DOI: 10.2196/publichealth.7373. [Online]. Available: http://publichealth.jmir.org/2017/4/e94/ (visited on 05/06/2019).

[43] O. Granberg, J. G. Bellika, E. Årsand, and G. Hartvigsen, "Automatic infection detection system," p. 5, [Online]. Available: http://ebooks.iospress.nl/publication/11039.

[44] N. Kanhabua and W. Nejdl, "Understanding the diversity of tweets in the time of outbreaks," in *Proceedings of the 22nd International Conference on World Wide Web - WWW '13 Companion*, Rio de Janeiro, Brazil: ACM Press, 2013, pp. 1335–1342, ISBN: 978-1-4503-2038-2. DOI: 10.1145/2487788.2488172. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2487788.2488172 (visited on 05/10/2019).

[45] N. Collier and S. Doan, "Syndromic classification of twitter messages," *arXiv:1110.3094 [cs]*, Oct. 13, 2011. arXiv: 1110.3094. [Online]. Available: http://arxiv.org/abs/1110.3094 (visited on 05/10/2019).

[46] (2019). Healthmap, [Online]. Available: https://www.healthmap.org/en/ (visited on 05/10/2019).

[47] (2019). Diseasedaily healthmap, [Online]. Available: http://www.diseasedaily.org/about (visited on 05/10/2019).

[48] (Nov. 2016). What is diabetes? niddk, [Online]. Available: https://www.niddk.nih.gov/health-information/diabetes/overview/what-is-diabetes (visited on 02/04/2019).

[49] (Nov. 2017). Monogenic diabetes (neontal diabetes mellitus and mody), [Online]. Available: `https://www.niddk.nih.gov/health-information/diabetes/overview/what-is-diabetes/monogenic-neonatal-mellitus-mody` (visited on 02/04/2019).

[50] "Classification and diagnosis of diabetes," *Diabetes Care*, vol. 39, S13–S22, Supplement 1 Jan. 2016, ISSN: 0149-5992, 1935-5548. DOI: `10.2337/dc16-S005`. [Online]. Available: `http://care.diabetesjournals.org/lookup/doi/10.2337/dc16-S005` (visited on 05/08/2019).

[51] C. Marling and R. Bunescu, "The OhioT1dm dataset for blood glucose level prediction," p. 4,

[52] (Mar. 2015). Dka (ketoacidosis) and ketones, [Online]. Available: `http://www.diabetes.org/living-with-diabetes/complications/ketoacidosis-dka.html` (visited on 02/05/2019).

[53] (Dec. 2018). Hyperglycemia (high blood glucose), [Online]. Available: `http://www.diabetes.org/living-with-diabetes/treatment-and-care/blood-glucose-control/hyperglycemia.html` (visited on 01/05/2019).

[54] (Aug. 2016). Hypoglycemia (low blood glucose), [Online]. Available: `https://www.niddk.nih.gov/health-information/diabetes/overview/preventing-problems/low-blood-glucose-hypoglycemia` (visited on 03/05/2019).

[55] (2018). Snl omd datatilsynet, [Online]. Available: `https://snl.no/Datatilsynet` (visited on 03/13/2019).

[56] (Mar. 2019). Evolution of http (mozilla), [Online]. Available: `https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/Evolution_of_HTTP` (visited on 01/05/2019).

[57] (Aug. 2008). The transport layer security (tls) protocol — rfc 5246., [Online]. Available: `https://tools.ietf.org/html/rfc5246` (visited on 01/05/2019).

[58] D. Naylor, A. Finamore, I. Leontiadis, Y. Grunenberger, M. Mellia, M. Munafò, K. Papagiannaki, and P. Steenkiste, "The cost of the "s" in HTTPS," in *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies - CoNEXT '14*, Sydney, Australia: ACM Press, 2014, pp. 133–140, ISBN: 978-1-4503-3279-8. DOI: `10.1145/2674005.2674991`. [Online]. Available: `http://dl.acm.org/citation.cfm?doid=2674005.2674991` (visited on 02/05/2019).

[59] O. f. C. Rights (OCR). (Jul. 9, 2012). Methods for de-identification of PHI, HHS.gov, [Online]. Available: `https://www.hhs.gov/hipaa/for-professionals/privacy/special-topics/de-identification/index.html` (visited on 01/05/2019).

[60] O. Uzuner, Y. Luo, and P. Szolovits, "Evaluating the state-of-the-art in automatic de-identification," *Journal of the American Medical Informatics Association*, vol. 14, no. 5, pp. 550–563, Sep. 1, 2007, ISSN: 1067-5027, 1527-974X. DOI: `10.1197/jamia.M2444`. [Online]. Available: `https://`

academic.oup.com/jamia/article-lookup/doi/10.1197/jamia.M2444 (visited on 05/07/2019).

[61] M. Jones, J. Bradley, and N. Sakimura, "JSON web token (JWT)," RFC Editor, RFC7519, May 2015, RFC7519. DOI: 10.17487/RFC7519. [Online]. Available: https://www.rfc-editor.org/info/rfc7519 (visited on 05/10/2019).

[62] (2019). Json web token webpage, [Online]. Available: https://jwt.io/ (visited on 10/05/2019).

[63] C. D. Shaw, "How can healthcare standards be standardised?" *BMJ Quality & Safety*, vol. 24, no. 10, pp. 615–619, Oct. 2015, ISSN: 2044-5415, 2044-5423. DOI: 10.1136/bmjqs-2015-003955. [Online]. Available: http://qualitysafety.bmj.com/lookup/doi/10.1136/bmjqs-2015-003955 (visited on 05/11/2019).

[64] (2019). Health level 7 introduction to hl7 standards, [Online]. Available: http://www.hl7.org/implement/standards/index.cfm?ref=nav (visited on 05/11/2019).

[65] (2019). Logical observation identifiers names and codes, [Online]. Available: https://loinc.org/about/ (visited on 05/05/2019).

[66] (2019). Fullflow, [Online]. Available: https://ehealthresearch.no/en/projects/fullflow (visited on 05/10/2019).

[67] W. A. Zebene, E. Årsand, B. Taxiarchis, and H. Gunnar, "An early infectious disease outbreak detection mechanism based on self-recorded data from people with diabetes," *Studies in Health Technology and Informatics*, pp. 619–623, 2017, ISSN: 0926-9630. DOI: 10.3233/978-1-61499-830-3-619. [Online]. Available: http://www.medra.org/servlet/aliasResolver?alias=iospressISBN&isbn=978-1-61499-829-7&spage=619&doi=10.3233/978-1-61499-830-3-619 (visited on 05/08/2019).

[68] T. Botsis, O. Hejlesen, J. Bellika, and G. Hartvigsen, "Blood glucose levels as an indicator for the early detection of infections in type-1 diabetes," English, *Advances in Disease Surveillance*, vol. 4, p. 147, 2007.

[69] E. Årsand, O. A. Walseth, N. Andersson, R. Fernando, O. Granberg, J. G. Bellika, and G. Hartvigsen, "Using blood glucose data as an indicator for epidemic disease outbreaks," p. 6,

[70] J. N. Lauritzen, E. Arsand, K. Van Vuurden, J. G. Bellika, O. K. Hejlesen, and G. Hartvig-sen, "Towards a mobile solution for predicting illness in type 1 diabetes mellitus: Development of a prediction model for detecting risk of illness in type 1 diabetes prior to symptom onset," in *2011 2nd International Conference on Wireless Communication, Vehicular Technology, Information Theory and Aerospace & Electronic Systems Technology (Wireless VITAE)*, Chennai, India: IEEE, Feb. 2011, pp. 1–5, ISBN: 978-1-4577-0786-5. DOI: 10.1109/WIRELESSVITAE.2011.5940877. [Online]. Available: http://ieeexplore.ieee.org/document/5940877/ (visited on 05/16/2019).

[71] (2019). Prisma website, [Online]. Available: `http://www.prisma-statement.org/` (visited on 05/19/2019).

[72] (2019). Awesome go: A currated list of aewsome go frameworks, libraries and software, [Online]. Available: `https://github.com/avelino/awesome-go` (visited on 05/13/2019).

[73] S. Robertson and J. Robertson, *Mastering the requirements process*, 2nd ed. Upper Saddle River, NJ: Addison-Wesley, 2006, 560 pp., OCLC: ocm62697079, ISBN: 978-0-321-41949-1.

Appendices

A

Appendix 1

Documentation of the servers API is provided below. It is also available at `https://documenter.getpostman.com/view/7682478/S1TU2yHY?version=latest`. Each API call contains the following;

1. A name with the syntax: [NAME] Function
2. The URL and path of the given request
3. Information about the header
4. The body of the request
5. An example of how to use the given request with curl

EDMON

GET [Location] Get static

localhost:8000/location/GetStaticLocation

Get the users static location

HEADERS

Authorization

Bearer {{token}}

Example Request

[Location] Get static

```
curl --location --request GET "localhost:8000/location/GetStaticLocation" \
  --header "Authorization: Bearer {{token}}"
```

POST [Location] Set static

localhost:8000/location/PostStaticLocation

Set the users static location

HEADERS

Content-Type

application/json

Authorization

BODY

```
{
    "latitude": {{LATITUDE}},
    "longitude": {{LONGITUDE}}
}
```

Example Request

[Location] Set static

```
curl --location --request POST "localhost:8000/location/PostStaticLocation" \
  --header "Content-Type: application/json" \
  --header "Authorization: " \
  --data "{
    \"latitude\": {{LATITUDE}},
    \"longitude\": {{LONGITUDE}}
}"
```

POST [Locaiton] Set dynamic

```
localhost:8000/location/PostDynamicLocation
```

Set the users dynamic location

HEADERS

Content-Type

application/json

Authorization

Bearer {{token}}

BODY

```
{
    "latitude": {{LATITUDE}},
    "longitude": {{LONGITUDE}}
}
```

Example Request

[Locaiton] Set dynamic

```
curl --location --request POST "localhost:8000/location/PostDynamicLocation" \
  --header "Content-Type: application/json" \
  --header "Authorization: Bearer {{token}}" \
  --data "{
    \"latitude\": {{LATITUDE}},
    \"longitude\": {{LONGITUDE}}
}"
```

PUT [Location] Edit static

localhost:8000/location/EditStaticLocation

Edit the users static location

HEADERS

Content-Type

application/json

Authorization

Bearer {{token}}

BODY

```
{
    "latitude": {{LATITUDE}},
    "longitude": {{LONGITUDE}}
}
```

Example Request

[Location] Edit static

```
curl --location --request PUT "localhost:8000/location/EditStaticLocation" \
  --header "Content-Type: application/json" \
  --header "Authorization: Bearer {{token}}" \
  --data "{
    \"latitude\": {{LATITUDE}},
    \"longitude\": {{LONGITUDE}}
}"
```

PUT [Location] Edit dynamic

localhost:8000/location/EditDynamicLocation

Edit the dynamic location of a user

HEADERS

Content-Type

application/json

Authorization

BODY

```
{
    "latitude": {{LATITUDE}},
    "longitude": {{LONGITUDE}}
}
```

Example Request

[Location] Edit dynamic

```
curl --location --request PUT "localhost:8000/location/EditDynamicLocation" \
  --header "Content-Type: application/json" \
  --header "Authorization: " \
  --data "{
    \"latitude\": {{LATITUDE}},
    \"longitude\": {{LONGITUDE}}
}"
```

DEL [Location] Delete static

localhost:8000/location/DeleteStaticLocation/

Deletes the users static location

HEADERS

Authorization

Bearer {{token}}

Example Request

[Location] Delete static

```
curl --location --request DELETE "localhost:8000/location/DeleteStaticLocation/" \
  --header "Authorization: Bearer {{token}}"
```

DEL [Location] Delete dynamic

localhost:8000/location/DeleteDynamicLocation/

Delete the users dynamic location

HEADERS

Authorization

Bearer {{token}}

PARAMS

Example Request

[Location] Delete dynamic

```
curl --location --request DELETE "localhost:8000/location/DeleteDynamicLocation/" \
  --header "Authorization: Bearer {{token}}"
```

GET [MedRec] Get all medical records

https://localhost:8000/medrec/GetAll

Fetch all the medical records for a user

HEADERS

Authorization

Bearer {{token}}

Example Request

[MedRec] Get all medical records

```
curl --location --request GET "https://localhost:8000/medrec/GetAll" \
  --header "Authorization: Bearer {{token}}"
```

GET [MedRecs] Get all with given LOINC

localhost:8000/medrec/GetAllLoinc/:loinc

Fetch all medical records with a given LOINC code for a given user

HEADERS

Authorization

Bearer {{token}}

PATH VARIABLES

loinc

LOINC-CODE

Example Request

[MedRecs] Get all with given LOINC

```
curl --location --request GET "localhost:8000/medrec/GetAllLoinc/:loinc" \
  --header "Authorization: Bearer {{token}}"
```

GET [MedRecs] Get last X days

localhost:8000/medrec/GetPastDays/:day

Gets the past X days of recordings

HEADERS

Authorization

Bearer {{token}}

PATH VARIABLES

day

{{date}}

Example Request

[MedRecs] Get last X days

```
curl --location --request GET "localhost:8000/medrec/GetPastDays/:day" \
  --header "Authorization: Bearer {{token}}"
```

PUT [MedRecs] Edit

localhost:8000/medrec/EditRecord/

Edit an existing medical record (update)

HEADERS

Content-Type

application/json

Authorization

Bearer {{token}}

BODY

```
{
    "Value": {{ID}},
    "HL7": "LOINC-CODE",
    "Timestamp": "2016-01-02 15:04:01",
    "ID": 4
}
```

Example Request

[MedRecs] Edit

```
curl --location --request PUT "localhost:8000/medrec/EditRecord/" \
  --header "Content-Type: application/json" \
  --header "Authorization: Bearer {{token}}" \
  --data "{
    \"Value\": {{ID}},
    \"HL7\": \"LOINC-CODE\",
    \"Timestamp\": \"2016-01-02 15:04:01\",
    \"ID\": 4
}"
```

POST [MedRecs] Post

http://localhost:8000/medrec/CreateRecord

Create a new medical record

HEADERS

Content-Type

application/json

Authorization

Bearer {{token}}

PARAMS

BODY

```
{
    "Value": {{value}},
    "HL7": "LOINC-CODE",
    "Timestamp": "2019-05-05 12:25:01",
    "Location_id": {{ID}},
    "recording": {{BASE64-VIDEO}}
}
```

Example Request

[MedRecs] Post

```
curl --location --request POST "http://localhost:8000/medrec/CreateRecord" \
  --header "Content-Type: application/json" \
  --header "Authorization: Bearer {{token}}" \
  --data "{
    \"Value\": {{value}},
    \"HL7\": \"LOINC-CODE\",
    \"Timestamp\": \"2019-05-05 12:25:01\",
    \"Location_id\": {{ID}},
    \"recording\": {{BASE64-VIDEO}}
}
```

DEL [MedRecs] Delete

localhost:8000/medrec/DeleteRecord/:id

Deltes a record (Give an ID)

HEADERS

Authorization

Bearer {{token}}

PATH VARIABLES

id

{{ID}}

Example Request

[MedRecs] Delete

```
curl --location --request DELETE "localhost:8000/medrec/DeleteRecord/:id" \
  --header "Authorization: Bearer {{token}}"
```

GET [User] Get

https://localhost:8000/user/GetUser/

Fetch all data about your user

HEADERS

Content-Type

application/json

Authorization

Bearer {{token}}

Example Request

[User] Get

```
curl --location --request GET "https://localhost:8000/user/GetUser/" \
  --header "Content-Type: application/json" \
  --header "Authorization: Bearer {{token}}"
```

POST [User] Create user

https://localhost:8000/user/CreateUser

Create a new user

HEADERS

Content-Type

application/json

Authorization

Bearer {{token}}

PARAMS

BODY

```
{
    "username": "Username",
    "email": "test@email.com",
    "password": "password"
}
```

Example Request

[User] Create user

```
curl --location --request POST "https://localhost:8000/user/CreateUser" \
  --header "Content-Type: application/json" \
  --data "{
    \"username\": \"Username\",
    \"email\": \"test@email.com\",
    \"password\": \"password\"
}

"
```

PUT [User] Edit

```
localhost:8000/user/EditUser
```

Edit the information on a user

HEADERS

Content-Type

application/json

Authorization

Bearer {{token}}

BODY

```
{
    "username": {{USENAME}},
    "email": {{EMAIL}},
    "staticlocation": [2,2],
    "dynamiclocation": [2,2],
    "SicknessStatus": true
}
```

Example Request

[User] Edit

```
curl --location --request PUT "localhost:8000/user/EditUser" \
  --header "Content-Type: application/json" \
  --header "Authorization: Bearer {{token}}" \
  --data "{
    \"username\": {{USENAME}},
    \"email\": {{EMAIL}},
    \"staticlocation\": [2,2],
    \"dynamiclocation\": [2,2],
    \"SicknessStatus\": true
}
```

DEL [User] Delete

localhost:8000/user/DeleteUser/

Delete your user

HEADERS

Authorization

Bearer {{token}}

PARAMS

Example Request

[User] Delete

```
curl --location --request DELETE "localhost:8000/user/DeleteUser/" \
  --header "Authorization: Bearer {{token}}"
```

POST Authenticate

https://localhost:8000/auth/login

Provide your username and password to get the JSON web token from the server

HEADERS

Content-Type

application/json

Authorization

Bearer {{token}}

BODY

```
{
  "username": {{USERNAME}},
  "userid": {{UID}},
  "password": {{{PASSWORD}}
}
```

Example Request

Authenticate

```
curl --location --request POST "https://localhost:8000/auth/login" \
  --header "Content-Type: application/json" \
  --data "{
  \"username\": {{USERNAME}},
  \"userid\": {{UID}},
  \"password\": {{{PASSWORD}}
}"
```