



NTNU

Kunnskap for en bedre verden

Bacheloroppgave

IE303612 - Bacheloroppgave

Track'a'Pet

10021

10019

10050

Totalt antall sider inkludert forsiden: 100

Innlevert Ålesund, 31.05.2018

Obligatorisk egenerklæring/gruppeerklæring

Den enkelte student er selv ansvarlig for å sette seg inn i hva som er lovlige hjelpemidler, retningslinjer for bruk av disse og regler om kildebruk. Erklæringen skal bevisstgjøre studentene på deres ansvar og hvilke konsekvenser fusk kan medføre. **Manglende erklæring fritar ikke studentene fra sitt ansvar.**

Du/dere fyller ut erklæringen ved å klikke i ruten til høyre for den enkelte del 1-6:		
1.	Jeg/vi erklærer herved at min/vår besvarelse er mitt/vårt eget arbeid, og at jeg/vi ikke har brukt andre kilder eller har mottatt annen hjelp enn det som er nevnt i besvarelsen.	<input checked="" type="checkbox"/>
2.	Jeg/vi erklærer videre at denne besvarelsen: <ul style="list-style-type: none"> • ikke har vært brukt til annen eksamen ved annen avdeling/universitet/høgskole innenlands eller utenlands. • ikke refererer til andres arbeid uten at det er oppgitt. • ikke refererer til eget tidligere arbeid uten at det er oppgitt. • har alle referansene oppgitt i litteraturlisten. • ikke er en kopi, duplikat eller avskrift av andres arbeid eller besvarelse. 	<input checked="" type="checkbox"/>
3.	Jeg/vi er kjent med at brudd på ovennevnte er å <u>betrakte som fusk</u> og kan medføre annullering av eksamen og utestengelse fra universiteter og høgskoler i Norge, jf. Universitets- og høgskoleloven §§4-7.	<input checked="" type="checkbox"/>
4.	Jeg/vi er kjent med at alle innleverte oppgaver kan bli plagiatkontrollert	<input checked="" type="checkbox"/>
5.	Jeg/vi er kjent med at NTNU vil behandle alle saker hvor det foreligger mistanke om fusk.	<input checked="" type="checkbox"/>
6.	Jeg/vi har satt oss inn i regler og retningslinjer i bruk av kilder og referanser på biblioteket sine nettsider	<input checked="" type="checkbox"/>

Publiseringsavtale

Studiepoeng: 20

Veileder: Girts Strazdins

Fullmakt til elektronisk publisering av oppgaven

Forfatter(ne) har opphavsrett til oppgaven. Det betyr blant annet enerett til å gjøre verket tilgjengelig for allmennheten (Åndsverkloven §2).

Alle oppgaver som fyller kriteriene vil bli registrert og publisert i Brage med forfatter(ne)s godkjenning.

Oppgaver som er unntatt offentlighet eller båndlagt vil ikke bli publisert.

Jeg/vi gir herved NTNU en vederlagsfri rett til å

gjøre oppgaven tilgjengelig for elektronisk publisering:

ja nei

Er oppgaven båndlagt (konfidensiell)?

ja nei

(Båndleggingsavtale må fylles ut)

- Hvis ja:

Kan oppgaven publiseres når båndleggingsperioden er over?

ja nei

Er oppgaven unntatt offentlighet?

ja nei

(inneholder taushetsbelagt informasjon. Jfr. Offl. §13/Fvl. §13)

Dato: 30.05.2018

I. PREFACE

The group would like to use this opportunity to thank Anna Kristine Fiskerstrand for illustrating the logo and some of the icons used. In addition, Girts Strazdins for excellent feedback and follow-up throughout the whole process.

II. TABLE OF CONTENTS

I. PREFACE	I
II. TABLE OF CONTENTS	II
III. SUMMARY	VII
IV. TERMINOLOGY	VIII
CONCEPTS	VIII
ABBREVIATIONS	IX
1 INTRODUCTION	1
1.1 BACKGROUND	1
1.2 PURPOSE AND APPROACH	1
1.3 THESIS STRUCTURE	1
2 THEORETICAL BASIS	2
2.1 LAWS AND REGULATIONS	2
2.1.1 <i>Universal Design of ICT</i>	2
2.1.2 <i>Personal data act</i>	2
2.1.3 <i>General Data Protection Regulation</i>	2
2.2 INFORMATION SECURITY	3
2.2.1 <i>Hashing of passwords</i>	3
2.2.2 <i>Tokens</i>	3
2.2.3 <i>Cryptography</i>	4
2.2.4 <i>HTTPS</i>	6
2.2.5 <i>Common attacks</i>	6
2.3 STANDARDS	7
2.3.1 <i>HTTP</i>	7
2.3.1 <i>GPS Standards</i>	8
2.3.2 <i>JSON</i>	9
2.4 AGILE DEVELOPMENT	10
2.5 OBJECT ORIENTED PROGRAMMING	10
2.5.1 <i>Coupling</i>	10
2.5.2 <i>Cohesion</i>	10
2.5.3 <i>Class</i>	11
2.5.4 <i>Interfaces</i>	11
2.5.5 <i>Unit Testing</i>	11
2.5.6 <i>Camel case</i>	11
2.6 DESIGN PATTERNS AND PRINCIPLES	11
2.6.1 <i>Inversion of Control</i>	11

BACHELOR THESIS

2.6.2	<i>Model-view-controller</i>	12
2.7	MAN-MACHINE INTERACTION	12
2.7.1	<i>Gestalt principles</i>	12
2.7.2	<i>Responsiveness</i>	14
2.7.3	<i>Alerts</i>	14
2.8	ANDROID AND ANDROID STUDIO	14
2.8.1	<i>Manifest.xml</i>	14
3	MATERIALS AND METHODS	16
3.1	PROJECT ORGANIZATION	16
3.2.1	<i>Project team</i>	16
3.2.2	<i>Workflow</i>	16
3.2.3	<i>Pair Programming</i>	17
3.3	PRE-PLANNING	17
3.4	ARCHITECTURE	18
3.4.1	<i>Spring Boot</i>	18
3.4.2	<i>PostgreSQL</i>	18
3.4.3	<i>JDBC Template</i>	18
3.4.4	<i>REST</i>	19
3.4.5	<i>Servlet Container</i>	20
3.4.6	<i>Java Servlets</i>	20
3.5	PROGRAMMING LANGUAGES	21
3.5.1	<i>Arduino</i>	21
3.5.2	<i>Java</i>	21
3.6	CODING CONVENTION	21
3.6.1	<i>Naming</i>	21
3.6.2	<i>Structure</i>	21
3.6.3	<i>Comments</i>	22
3.6.4	<i>Language</i>	23
3.6.5	<i>Implementation</i>	23
3.7	EXTERNAL LIBRARIES – MICROCONTROLLER	23
3.7.1	<i>GSM</i>	23
3.7.2	<i>Adafruit_GPS_Alt</i>	23
3.7.3	<i>AltSoftSerial</i>	23
3.7.4	<i>MemoryFree</i>	23
3.7.5	<i>ArduinoDES</i>	23
3.8	EXTERNAL LIBRARIES - BACKEND	24
3.8.1	<i>Bcrypt</i>	24

BACHELOR THESIS

3.8.2	<i>Liquibase</i>	24
3.8.3	<i>JSON web token</i>	25
3.8.4	<i>Project Lombok</i>	26
3.9	EXTERNAL LIBRARIES – ANDROID	26
3.9.1	<i>Gson</i>	26
3.9.2	<i>CircleImageView</i>	26
3.9.3	<i>Spectrum</i>	27
3.9.4	<i>GraphView</i>	27
3.10	DEVELOPMENT TOOLS	27
3.10.1	<i>IntelliJ IDEA Ultimate</i>	27
3.10.2	<i>Android Studio</i>	27
3.10.3	<i>Bitbucket</i>	27
3.11	SUPPORT TOOLS	28
3.11.1	<i>Network Analyzer</i>	28
3.12	TESTING METHODS	29
3.12.1	<i>Unit Testing</i>	29
3.12.2	<i>Integration Testing</i>	29
3.13	TESTING TOOLS	29
3.13.1	<i>Mockito</i>	29
3.13.2	<i>JUnit</i>	29
3.14	MATERIALS	29
3.14.1	<i>Arduino Uno rev3</i>	30
3.14.2	<i>Arduino GSM shield 2</i>	30
3.14.3	<i>Adafruit Ultimate GPS Breakout</i>	30
3.15	DOCUMENTATION	30
3.15.1	<i>Confluence</i>	30
3.15.2	<i>Jira</i>	31
3.15.3	<i>Minutes of Meeting</i>	32
3.15.4	<i>Retrospective</i>	32
4	RESULTS	33
4.1	ARCHITECTURE	33
4.2	BACKEND ARCHITECTURE	34
4.2.1	<i>Programming language and platform</i>	34
4.2.2	<i>Overall structure</i>	35
4.2.3	<i>Components</i>	35
4.2.4	<i>Database Versioning</i>	37
4.2.5	<i>Hosting and deployment</i>	37

BACHELOR THESIS

4.3	MICROCONTROLLER	38
4.3.1	<i>Hardware</i>	38
4.3.2	<i>Mobile Subscription</i>	39
4.3.3	<i>Software</i>	40
4.4	FRONTEND ARCHITECTURE	42
4.4.1	<i>Programming language</i>	42
4.4.2	<i>Overall structure</i>	42
4.4.3	<i>Components</i>	42
4.5	USE CASE DIAGRAM	45
4.6	CLASS DIAGRAM	46
4.6.1	<i>Frontend</i>	46
4.6.2	<i>Backend</i>	46
4.7	DATABASE DESIGN	47
4.7.1	<i>Entity relationships</i>	48
4.7.2	<i>Overview of attributes</i>	48
4.8	SECURITY	50
4.8.1	<i>Protection from SQL injection</i>	50
4.8.2	<i>Protection from Broken Authentication</i>	50
4.8.3	<i>Protection from sensitive data exposure</i>	51
4.8.4	<i>Microcontroller</i>	51
4.9	DESIGN	52
4.9.1	<i>Wireframes</i>	52
4.9.2	<i>Layout</i>	54
4.9.3	<i>User feedback and alerts</i>	62
4.9.4	<i>Design Concerns</i>	64
4.10	TESTING	64
4.10.1	<i>Regression Testing</i>	64
4.10.2	<i>Manual Testing</i>	64
4.11	BUSINESS ANGLE	65
4.11.1	<i>Market</i>	65
4.11.2	<i>Hardware Cost</i>	66
4.11.3	<i>Software Cost</i>	66
4.11.4	<i>Income</i>	67
4.11.5	<i>Financial Feasibility</i>	67
5	DISCUSSION	68
5.1	TECHNICAL RESULT	68
5.1.1	<i>Microcontroller</i>	68

BACHELOR THESIS

5.1.2	<i>Frontend</i>	68
5.1.3	<i>Backend</i>	69
5.1.4	<i>Device registration</i>	70
5.1.5	<i>Security</i>	70
5.1.6	<i>Design</i>	71
5.1.7	<i>Testing</i>	71
5.1.8	<i>Complexity</i>	71
5.1.9	<i>Future Development</i>	71
5.1.10	<i>Existing solutions</i>	72
5.2	PROJECT EXECUTION	73
5.2.1	<i>Development Methodology</i>	73
5.2.2	<i>Organization</i>	73
5.2.3	<i>Version Control</i>	73
5.2.4	<i>Use Of Sources</i>	73
6	CONCLUSION	74
7	REFERENCES	75
	TABLE OF FIGURES	82
	TABLE OF TABLES	84
	TABLE OF CODE SNIPPETS	85
	APPENDIX	86
	SOURCE CODE	86
	APPENDIX 1 – PRELIMINARY REPORT	1
	APPENDIX 2 – RETROSPECTIVE	1
	APPENDIX 3 – MEETING NOTES	1
	APPENDIX 4 – BURNDOWN CHARTS	1

III. SUMMARY

About one in three Norwegian households own at least one pet. Up until recently, there were no good solutions for keeping track of pets that could go missing. To help pet owners safeguard their pets, we wanted to develop a system for easily tracking them. Our goal was to develop an Android application that displayed pet location data. The data would come from a tracker on the pet, and saved to a remote server.

Development of this system has led to a system that offers security, safety and awareness surrounding pet movement and well-being. The system comprises of an Arduino tracker device, a backend REST service and a frontend Android application. Security has been paramount in this project, and most recent development practices has been implemented to retain satisfactory information security.

The process of creating this system have been a steep learning curve. The team has invested significant effort in creating a system which is both scalable and modular, in compliance with privacy and security principles.

IV. TERMINOLOGY

CONCEPTS

Confidentiality	Confidentiality means when information is protected from exposure or disclosure to unauthorized people or systems. It ensures that only those with the correct rights and privileges to access data can do so.
Integrity	Integrity means that the information is whole, complete, and uncorrupted.
Availability	Availability means enabling authorized users or systems to access information without obstruction or interference, and to get it in the right format.
Tracker	The whole microcontroller system. Including microcontroller, GPS, GSM and battery.
Microcontroller	A small computer on a single integrated circuit (Brain 2018).
Server	A server is a computer that provides data, resources and services to other computers.
Client	A client is a computer which receives data, resources and services from a computer.
Web Server	A web-server is a computer that provide access to websites over internet.
Repository	A storage for development code, provides an overview over historical progress.
System	A system is a collection of two or more components which makes up a complex interconnected system, where each component has a specific task.
NMEA SENTENCE	A way of structuring GPS data mainly for marine equipment. The NMEA abbreviation stands for National Marine Electronics Association (DePriest, NMEA data 2018).
End-point	Web server URL, access point for a client application.

BACHELOR THESIS

ABBREVIATIONS

API	Application Programming Interface
GPS	Global Positioning System
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
SSL	Secure Sockets Layer
TLS	Transport Layer Security
CA	Certificate Authority
ACME	Automatic Certificate Management Environment
ICT	Information and Communication Technology
IDE	Integrated Development Environment
JDBC	Java DataBase Connectivity
JPA	Java Persistence API
JSON	JavaScript Object Notation
ORM	Object Relational Mapping
REST	REpresentational State Transfer
SQL	Structured Query Language
UML	Unified Modeling Language
URL	Uniform Resource Locator
URI	Uniform Resource Identifier
NMEA	National Marine Electronics Association
GPGGA	Global Positioning System Fix Data
IoC	Inversion of Control
CLI	Command-Line Interface
Java EE	Java Enterprise Edition
M2M	Machine-To-Machine
IoT	Internet of Things

BACHELOR THESIS

VM	Virtual Machine
GDPR	General Data Protection Regulation
OWASP	Open Web Application Security Project
AES	Advanced Encryption Standard
DES	Data Encryption Standard

1 Introduction

1.1 BACKGROUND

About 37 percent of all Norwegian households own a pet. A majority of those pets are cats (Aftenposten 2008). When first discussing this project, there were no available solutions for easy pet tracking in Norway. We wanted to develop a solution that would allow pet owners to keep track of their pets through their phone, and notify them if they wandered too far.

By creating such a system, we aimed to achieve a greater understanding of mobile and distributed applications, as well as information security and embedded programming. We hoped this would give us experience that would be useful in the job market.

1.2 PURPOSE AND APPROACH

As a pet owner, the security of a household pet is often paramount. Pets often disappear over long periods of time and to know with certainty where they are would give pet owners comfort. Furthermore, pet owners sometimes dedicate a lot of time and effort into finding lost pets, which could have been used on other leisure activities.

We would like to mitigate some of these adverse effects of owning pets, by using current technology to implement a location system for pet owners.

Our solution would consist of a mobile application, microcontroller tracker, and a backend for storing information and communicating with the trackers and mobile devices. The mobile app should be simple to use and provide satisfactory information security.

1.3 THESIS STRUCTURE

The report is divided into seven main parts which will describe different aspects of the project.

- **Introduction:** Contains definition of the purpose and problem of the project.
- **Theoretical basis:** Introduces and define terms essential to the report.
- **Materials and methods:** A description of architecture, procedures, services and testing methods in the project.
- **Results:** Contains a description of the end-product.
- **Discussion:** Contains discussion of selected solutions and other alternatives.
- **Conclusion:** Will give answers to our issues, conclusion of the final product and further development.
- **References:** Contains a list of references and attachments.

2 THEORETICAL BASIS

This chapter will describe the theoretical background essential to our assessments and decisions.

2.1 LAWS AND REGULATIONS

This section describes relevant laws and regulations that we need to take into account in the project.

2.1.1 UNIVERSAL DESIGN OF ICT

In Norway, we want a society where everyone can participate. Therefore, from 1 July 2014 all new ICT solutions developed must be universally designed. From 1 January 2021, existing solutions should comply with universal design requirements. This is a legal requirement for both the public and private sector (Difi 2017).

§ 1 Purpose of the regulation

“The purpose of the regulation is to ensure universal design of information and communication technology, without causing an undue burden on businesses. Universal design means that the design or adaptation of the main solution in information and communication technology is such that it can be used by as many as possible.” (Lovdata 2017)

2.1.2 PERSONAL DATA ACT

The purpose of the personal data act is to protect individuals from their privacy being violated through the processing of personal information. We will not store any sensitive information in our database. However, we will store user's e-mail address, and optionally their name and phone number. This is information can be used to identify individual people, which makes it personal data, according to the act.

The personal data act states that planned and systematic measures must be taken to ensure satisfactory information security, with respect to confidentiality, integrity and availability when processing personal data (Datatilsynet 2017).

2.1.3 GENERAL DATA PROTECTION REGULATION

The GDPR is a new EU privacy policy that will come into force by 2018 in Norway. The regulation imposes numerous tasks on businesses that handle personal information and give the consumer a variety of rights.

Extended duties for businesses include informing users how they process personal data, to investigate privacy implications, and to notify users about security breaches.

For the consumer, rights include the right to gain access to all information a company has stored about them, the right to data portability, and they may require that all information about them is deleted.

A summary of the GDPR state that businesses are not allowed to process personal information unless they have consent from the one in question (European Union 2016).

2.2 INFORMATION SECURITY

This section will provide a detailed outline of the different security measures we have implemented in the project, with relevant theory.

2.2.1 HASHING OF PASSWORDS

One should exercise great care when storing personal or sensitive information. A good rule of thumb is to only store information that is needed, and to ensure the confidentiality of sensitive information. This way, we can minimize the damage if our database is breached.

One of the most important pieces of information is user passwords. The majority of online users tend to use the same password on several online accounts (Irvine 2017). Theoretically, this means that if one site is breached, attackers can use the same credentials on several other sites.

Even if users do not reuse passwords, most of them are still easily cracked. They tend to follow the same patterns, are short, or are words you would find in a dictionary (Cohen 2017). To effectively safeguard passwords in a database, one should *hash* the passwords and store the hash instead of the plaintext password.

Hashing works by running the plaintext password through a special algorithm. The algorithm computes a single large value of a fixed size based on the bit value of the password. The hash is irreversible, meaning you cannot easily decode the plaintext value from a hash. The same exact password will always result in the same hash value, but two different passwords will generally never have the same hash (E. Whitman og J. Mattord 2014). However, a hashed password is not necessarily safe from a brute force attack.

Attackers are known for using *rainbow tables* when carrying out brute force attacks. A rainbow table is a long list of common passwords and their hash values. Attackers can use this to compare the user's hashed password with the hashed values in the table. This saves a lot of time since you do not have to hash all the values yourself. If the user has a weak or common password, the attackers can immediately find the plaintext version of it (E. Whitman og J. Mattord 2014). To combat this, an approach called *salting* is commonly used.

A salt is a randomly generated string of data that is appended to the password before it is hashed. The salt is not a secret, as it is often stored alongside the hashed password in a database. However, it negates the use of rainbow tables since the attacker no longer benefits from any time-memory trade-off (E. Whitman og J. Mattord 2014). This is because the resulting value of a password-salt hash will never be found in any rainbow table. Figure 1 provides a diagram of a hashing process with salt.

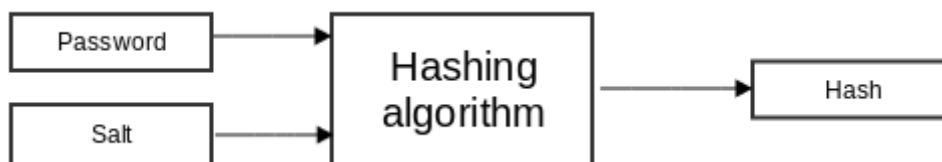


Figure 1: Password hashing process

2.2.2 TOKENS

An access token in information security is a piece of information that contains some data about the current security context of a user of a computer system. More specifically, it often contains an identifier for a user, along with their privileges (MSDN u.d.). This can for example be a unique user id and a privilege in form of a role, like 'user' or 'admin'.

The access token is generated when a user successfully logs in to the system. The user receives it from the server, and automatically saves it. When the user sends further requests to the server, a copy of the token is enclosed. The server decodes the token if it is encoded and reads its contents. If

BACHELOR THESIS

the information is correct, the request can go through. If not, the request is denied. The server also examines which privileges the user has, and denies and allows resources based on that (MSDN u.d.). Figure 2 shows a typical exchange of access tokens.

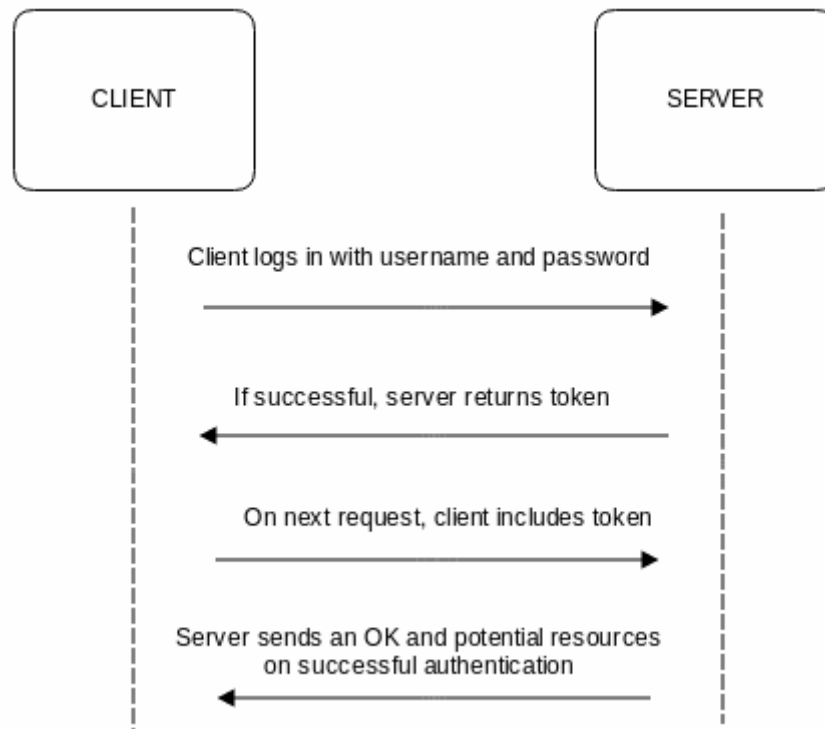


Figure 2: Typical flow of token exchange

Tokens can include a lot of other information than id and privileges. They often contain an expiration date, which means you must log in and get a new one when it expires. They can also for example include a session id to identify the current logon session, as well as several custom fields (MSDN u.d.).

2.2.3 CRYPTOGRAPHY

Cryptography is the study of encoding and decoding secret messages. Cryptographic techniques allow a sender to disguise data so that intercepted data appear as nonsense for an intruder (Kurose og Ross 2013).

2.2.3.1 Symmetric encryption

Symmetric encryption, also known as secret key encryption, is the oldest and best-known technique. A secret key is applied to the message to change the content in a particular way. The secret can be a number, a word, or a string of random letters. Both the sender and the recipient need to know the secret key, this way they can encrypt and decrypt all messages using this key (E. Whitman og J. Mattord 2014). Figure 3 provides an example of symmetric encryption using a shared key.

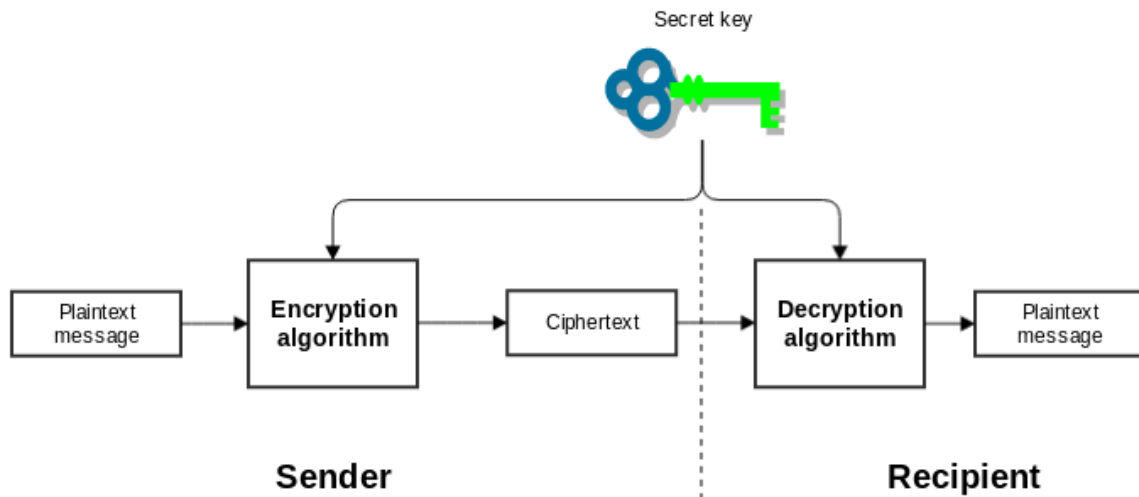


Figure 3: Symmetric encryption

2.2.3.2 Asymmetric encryption

In asymmetric encryption, also called public key encryption, there are a pair of related keys. The private key is kept protected and secret so only you know it, and the public key is made available to anyone who might want to send you a message. Anything encrypted with the public key can only be decrypted with the matching private key and vice-versa (E. Whitman og J. Mattord 2014). Figure 4 provides an example of asymmetric encryption.

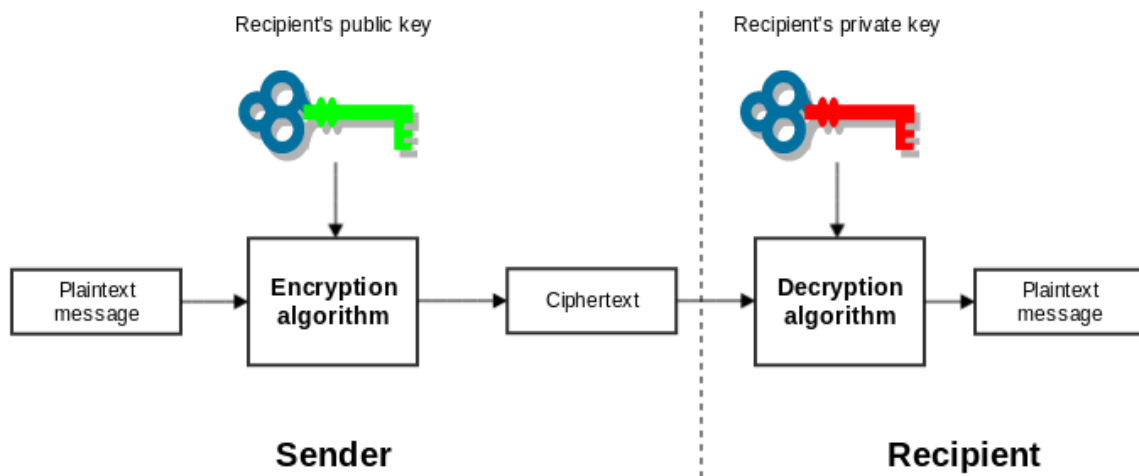


Figure 4: Public-key cryptography

Certificate

To use asymmetric encryption, we need a way for people to discover other public keys. To accomplish this, digital certificates is most commonly used. A certificate is a package of information identifying a user or server, and the user's public key.

When a key is needed, a query is sent to the other party. The other party returns a copy of the certificate, from where the public key can be extracted (Microsoft 2018).

BACHELOR THESIS

2.2.4 HTTPS

Communication between client and server over HTTP is not encrypted, meaning data is sent in plaintext. This makes it vulnerable to man-in-the-middle attacks, where an attacker secretly manages to come between the two communicating parties to listen or alter the communication.

HTTP Secure is an extension of HTTP used for secure communication, protecting integrity and confidentiality of data between the client and server. When connecting to a server using HTTPS, you can see a padlock icon in the URL bar, verifying that this is a trusted and secure connection. Figure 5 provides an example.



Figure 5: HTTPS connection to trackapet.uials.no

Typically, HTTPS is HTTP with an added encryption layer of SSL or TLS. These protocols use asymmetric encryption, described in section 2.2.3.2 (Comodo u.d.).

2.2.5 COMMON ATTACKS

This section describes relevant common security attacks and vulnerabilities.

2.2.5.1 Man-in-the-middle

A man-in-the-middle attack is an attack where someone intercepts, and possibly alters, a stream of communication between two parties, and convinces the parties that they are communicating directly. The attacker can insert new messages into the stream, making it look like they are coming from one of the legitimate parties, or alter existing messages (E. Whitman og J. Mattord 2014).

There are a number of ways a man-in-the-middle attack can occur. However, the most efficient way to be reasonably secure against them is to encrypt all communication with HTTPS.

2.2.5.2 SQL injection

An SQL injection attack occurs when malicious user input in a system is not properly validated before used to query a database. For example, an input field may expect an email value for use in a database query, like so:

```
String input = getInput();  
SELECT * FROM users WHERE email = ` + input + `;
```

If the input is a valid email, everything goes as expected. However, if the input is not sanitized, the user can insert a valid SQL query like so:

```
Input = name@mail.com' OR '1'='1
```

The resulting query would then be:

```
SELECT * FROM users WHERE email = `name@mail.com' OR '1'='1`;
```

Since 1=1 is always true, the query will return the account info for all the users in the table. The user can also insert code that would delete the whole table or database, or edit the information of other users (E. Whitman og J. Mattord 2014).

One way to mitigate SQL injections is to use an API that provides a parameterized interface, where input is not treated as query language (OWASP 2017).

2.3 STANDARDS

This part will describe standards used in the project.

2.3.1 HTTP

The Hypertext Transfer Protocol (HTTP) is a generic stateless application-level protocol which allows us to fetch resources, such as HTML documents. HTTP is the foundation of any data exchange on the web and it is a request/response protocol. A client sends a request to a server in form of a HTTP request message, and the server responds with a HTTP response message after receiving and interpreting the request message (Fielding og Reschke, RFC7230 2014).

2.3.1.1 HTTP request

The request message is built up from a request line, request headers, an empty line, and an optional body.

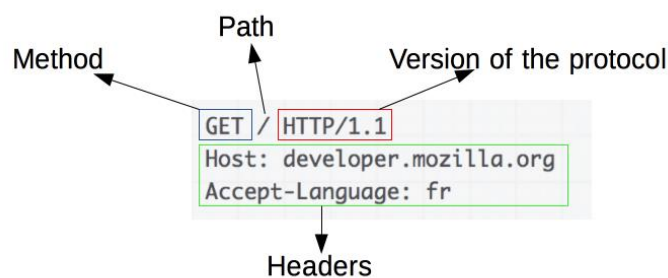


Figure 6: HTTP request (Mozilla 2018)

In Figure 6 we see an example of a HTTP request. The first line is the request line with three fields: the method field, the path or URL field, and protocol version field. Method defines the action to be applied to the resource, the most common methods are shown in Table 1. Path is a URI which identifies the resource to apply the request. In the third field of the request line, we have version of the protocol used. After the request line comes the request header fields. This allows the client to pass additional information about the request and the client itself (Kurose og Ross 2013).

Method	Description
GET	Retrieves whatever information identified by the request-URI
POST	Used to send data to the server
PUT	Asks the server to store the data
DELETE	Asks the server to delete the data

Table 1: HTTP methods

The optional part of a HTTP request is not shown in this example, but after the header fields there can be an empty line to separate the header and the body. Finally, the message body is used to carry the payload associated with the request (Fielding og Reschke, RFC7231 2014).

2.3.1.2 HTTP response

The response message is built up from a status line, response headers, an empty line and an optional body.

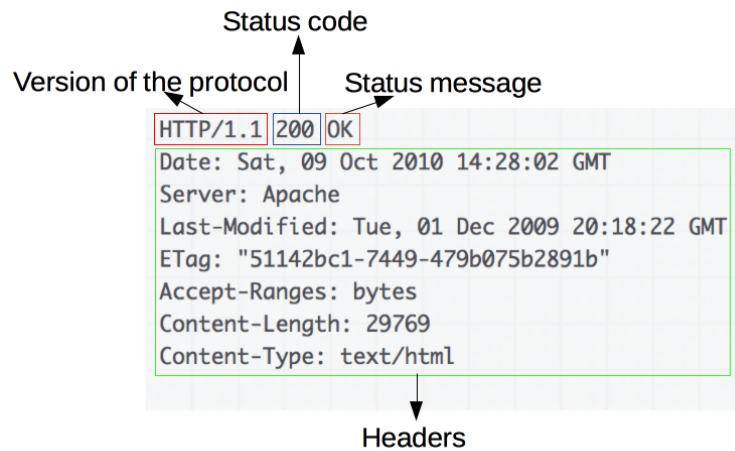


Figure 7: HTTP response (Mozilla 2018)

In Figure 7 we see an example of a HTTP response. The first line is the status line with three fields; protocol version field, a status code, and corresponding status message. The protocol version field shows the protocol used. Status code is a three-digit integer result code where the first digit represents the class of the response as we can see in Table 2. Status message gives a short textual description of the status code to help humans to understand the HTTP response.

After the status line is the response header fields, allowing the server to pass additional information about the response, server and further access to the resource (Kurose og Ross 2013).

Status code	Description
1xx (Informational)	Request received, continuing process
2xx (Successful)	The action was successfully received, understood and accepted
3xx (Redirection)	Further action must be taken to complete the request
4xx (Client error)	The request contains bad syntax or cannot be fulfilled
5xx (Server error)	The server failed to fulfill an apparently valid request

Table 2: HTTP status codes

As for the request, the response can also have an empty line and body to carry a payload (Fielding og Reschke, RFC7231 2014).

2.3.1 GPS STANDARDS

According to US Environmental Data Standards Council the standard for a GPS data message should be latitude before longitude (US Environmental Protection Agency 2017). This is also apparent from the old standards for GPS NMEA data sets, which sorts the coordinates latitude first, then longitude last (DePriest, NMEA data 2018). Furthermore, DePriest have described GGA NMEA sentence, which provides current fixed data from the GPS module. This standard sentence is formatted as shown in Figure 8.

GGA - essential fix data which provide 3D location and accuracy data.

```
$GPGGA,123519,4807.038,N,01131.000,E,1,08,0.9,545.4,M,46.9,M,,*47
```

Where:

GGA	Global Positioning System Fix Data
123519	Fix taken at 12:35:19 UTC
4807.038,N	Latitude 48 deg 07.038' N
01131.000,E	Longitude 11 deg 31.000' E
1	Fix quality: 0 = invalid
	1 = GPS fix (SPS)
	2 = DGPS fix
	3 = PPS fix
	4 = Real Time Kinematic
	5 = Float RTK
	6 = estimated (dead reckoning) (2.3 feature)
	7 = Manual input mode
	8 = Simulation mode
08	Number of satellites being tracked
0.9	Horizontal dilution of position
545.4,M	Altitude, Meters, above mean sea level
46.9,M	Height of geoid (mean sea level) above WGS84 ellipsoid
(empty field)	time in seconds since last DGPS update
(empty field)	DGPS station ID number
*47	the checksum data, always begins with *

Figure 8 – One type of NMEA sentence - GGA: Fix data (DePriest, NMEA data 2018)

2.3.2 JSON

JavaScript Object Notation is a lightweight data-interchange format for storing and exchanging data (Json.org 2018). It is easy for humans to read and write, and for machines to parse and generate.

Code Snippet 1 shows a Json array with two pets.

JSON data is structured in a specific way:

- data represented in key-value pairs
- colon (:) assigns value to a key
- comma (,) used to separate key-value pairs
- curly brackets ({ }) hold objects
- square brackets ([]) hold arrays

```
[
  {
    "id":1,
    "name":"Linus"
  },
  {
    "id":2,
    "name":"Pontus"
  }
]
```

Code Snippet 1: Json example

2.4 AGILE DEVELOPMENT

Agile development is an approach to software development which tries implement changes as soon as they arrive. Traditional development methods, such as plan-based development, have in many cases shown to be too rigid and inflexible leading to many projects failing to achieve their intended goals. Agile development tries to mitigate this by diverging from plan-based development style and incorporate change and refactoring as a common practice. In addition, it is deemed crucial to update the requirements underway in the development process (Sommerville 2016). Figure 9 shows the agile process.

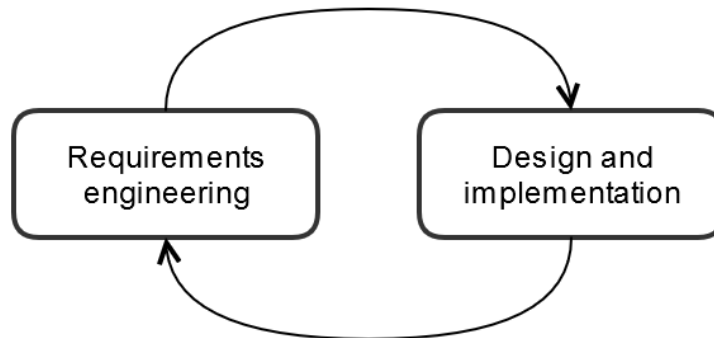


Figure 9: Agile development

2.5 OBJECT ORIENTED PROGRAMMING

Object oriented programming (OOP) is a human friendly approach to reduce the complexity of programming, by encapsulating logical entities in classes and abstractions. OOP is identified as a programming approach which offers state to objects, and mutability. This is different from functional programming, which does not rely on stateful data to function, rather that functions only depend on arguments. The OOP approach to break up complexity and offer a way to interact with objects and state offers a way to create complex systems with an easy overview of the whole program (Barnes 2012).

2.5.1 COUPLING

Coupling relates to the inter-connectiveness of the different software components. More specifically, how dependent each class is on other classes (Barnes 2012). The more independent each component is from each other, the more flexible the system will be.

2.5.2 COHESION

“A unit of code should always be responsible for one, and only one, task.” [(Barnes 2012), page 219]

Each method should only be responsible for one, and only one well-defined task. Each class should represent one single well-defined entity in the problem domain. By using these two principles the code will both be more readable, as well as more maintainable.

By keeping low coupling and high cohesion the developer can make software that are more durable at the same time as flexible. By lowering the dependence between components, the software is not as prone to erroneous bugs as the application increases in complexity over time.

BACHELOR THESIS

2.5.3 CLASS

A class is a logical unit which should encapsulate only one concept. It is a way to break up logic in programming to smaller concepts, and usually have a naming convention which highlights the concept behind the class. Classes can include fields, constructors and methods (Barnes 2012). A typical approach to thinking of a class is to associate it with a physical concept, such as an animal. For example, a duck. A duck weight (field), can produce sound (method) and must be created (constructor).

2.5.4 INTERFACES

“A Java Interface is a specification of a type (in the form of a type name and set of methods) that does not define any implementation for the method” [(Barnes 2012), page 355]. In other words, a class which implements an interface makes a promise to implement its own methods to match the interface methods.

2.5.5 UNIT TESTING

Unit tests are small and reusable test that test concepts and individual components in a system (Barnes 2012). These tests are often automated, so that the developer get feedback if any of the tests fail, and it gives the developer freedom to optimize the program without fear of implementing critical bugs. If the test pass, the program should operate according to specifications.

It is worth noting, to have full system integrity using unit testing, the programmer needs to implement different test for each scenario. Such as user input is empty, too large, negative, or wrong. The process of creating tests for each scenario can often be quite difficult, and time consuming.

2.5.6 CAMEL CASE

Camel case is a writing convention where the author uses capital letters to distinguish between words. Programmers use camel case to allow more meaningful naming in their programs without violation naming limitations (Rouse 2005).

2.6 DESIGN PATTERNS AND PRINCIPLES

In this section we will highlight some of the design patterns and principles we have used.

2.6.1 INVERSION OF CONTROL

IoC (Inversion of Control), sometimes referred to as Dependency Injection, is a design principle in which control of program flow is inverted to achieve looser coupling in a system (Tutorialsteacher u.d.).

For example, if you have a program that accepts user input through a command-line interface, you can invert control by implementing a graphical interface instead, where you have text boxes that can be interacted with in any order. By doing this, the interface framework controls the primary loop of the program and calls on your custom event handlers (Tutorialsteacher u.d.).

It is important to note that Inversion of Control is a very general term and may be achieved in a variety of ways.

2.6.2 MODEL-VIEW-CONTROLLER

Model-view-controller (MVC) is a design pattern for separating an application into three main parts that handles interaction, presentation, and business logic. The primary goal is to increase cohesion and achieve looser coupling by keeping the design clear and flexible (Freeman, et al. 2010).

The three parts each have their own responsibilities (Freeman, et al. 2010):

- **Model:** Handles application data and business logic. It is responsible for holding state, data, and business logic. It does not “know” of the view or controller, it only provides an interface the other parts can work with.
- **View:** Represents the user interface. The view gives the user a representation of the model, usually getting it directly from the model.
- **Controller:** Handles interaction with the user. It takes input from the user, parses it, and figures out what it means to the model.

Figure 10 shows how a user interacts with the different parts of the model-view-controller.

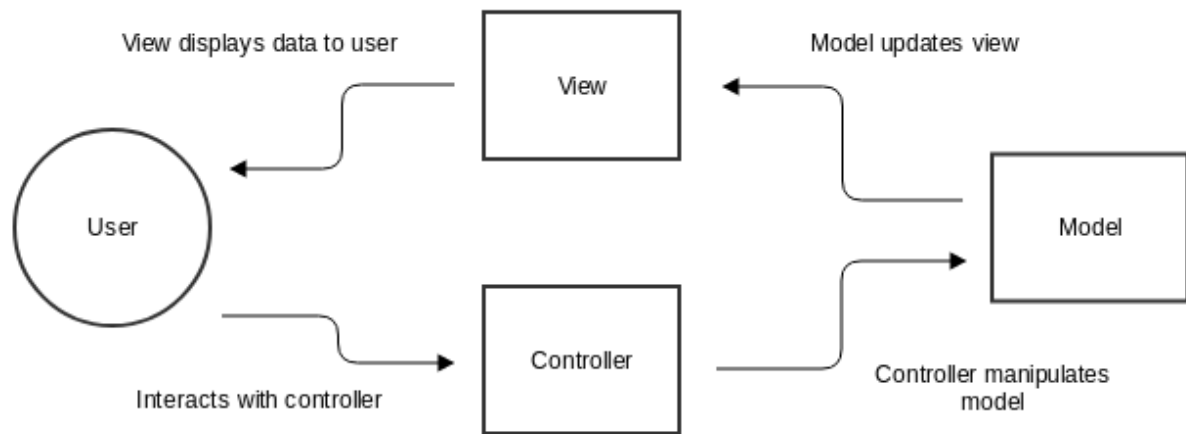


Figure 10: Overview of MVC interactions

2.7 MAN-MACHINE INTERACTION

This section will describe theory based on human-machine interaction.

2.7.1 GESTALT PRINCIPLES

Gestalt principles are guide lines to achieve visual logic and how to handle GUI composition. By following the gestalt principles as a developer and designer, the user satisfaction tends to be more in line with expectations (J. Johnson 2014).

2.7.1.1 Proximity

The gestalt principle proximity describes how humans tend to group objects with a similar spacing between them into the one logical group (J. Johnson 2014). If objects are presented as in Figure 11: Gestalt principle - Proximity there are two apparent logical groupings.

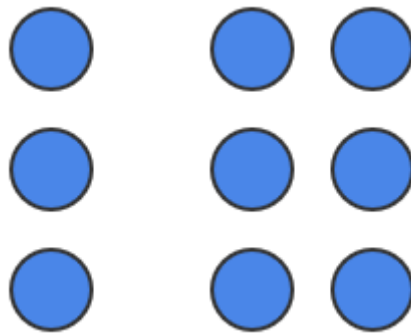


Figure 11: Gestalt principle - Proximity

2.7.1.2 Similarity

The gestalt principle similarity affects the tendency of grouping similar objects into related logical groupings (J. Johnson 2014). This is apparent in Figure 12: Gestalt principle – Similarity, where the triangles are one subset and the circles are another.

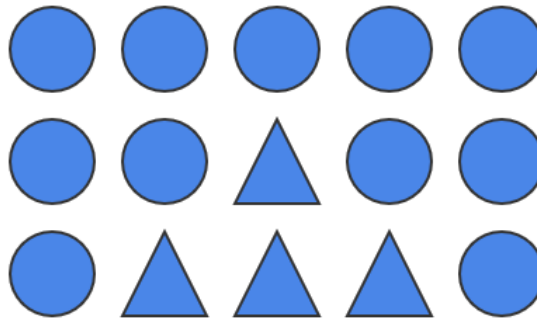


Figure 12: Gestalt principle – Similarity

2.7.1.3 Figure/ground

The gestalt principle figure/ground describes how the human mind separates visual information into a background and a foreground. Figure 13 demonstrates this principle. The blue circle is perceived as being in the foreground, while the green rectangle is perceived to be the background. When a small and large figure overlaps, humans tend to see the small figure as the foreground (J. Johnson 2014).



Figure 13: Gestalt principle – Figure/ground

2.7.1.4 Common fate

The gestalt principle common fate describes how objects that move together are perceived as grouped or belonging together. For example, if we had a group of identical objects and some of them wiggled in the same fashion, they would seem to belong together (J. Johnson 2014).

BACHELOR THESIS

2.7.2 RESPONSIVENESS

Responsiveness is the compliance with human time related expectations and in an application. It is closely related to how long the user needs to wait for the application to perform feedback to the user. Jeff Johnson defines the “maximum interval between events for perception that one event causes another event: 140 milliseconds” [(J. Johnson 2014), page 201]. It is considered best practice to give the user feedback, in the form of a loading icon, if a task is expected to exceed this time limit.

2.7.3 ALERTS

When giving feedback to the user in form of error messages or alerts, it is important to follow some basic guidelines. Humans have very poor peripheral vision, so messages should appear where the user is looking. For example, if a user inputs wrong email or password in a login field, the error message should appear as close to the login button as possible. Otherwise, the user might miss it and become frustrated with the system (J. Johnson 2014).

If the message you are displaying is an error message, it should be clearly marked as such. One way to mark an error message is to use the color red, as it is often connoted with danger. Another way is to use a symbol, like an exclamation mark or a traffic sign that users might already be familiar with. However, one should reserve these markings for errors, else they might be misinterpreted (J. Johnson 2014).

2.8 ANDROID AND ANDROID STUDIO

Android Studio offers a framework to develop applications from. Android development involves many different components, where the most essential components will be discussed in-depth further on. One of the most essential components in Android development is the manifest.xml file.

2.8.1 MANIFEST.XML

Manifest.xml represents the whole application and holds much of the system specific information. Manifest.xml contains among other things, permissions required by the app, activities and services belonging to the application, features used by the application and intent filters.

2.8.1.1 Permissions

Android is concerned with user security, and therefore any application which needs to use features and services deemed a potential risk of invasion of privacy needs to ask and receive permission before it can use compromising features or services (Android, Permission 2018). For example, to use locational data the following permission needs to be declared in the manifest.xml file, shown in Code Snippet 2.

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION"/>
```

Code Snippet 2: Declaring required permission

Furthermore, all security critical permissions need to be checked at runtime for the application to perform the given task. If the permission is not granted, the application either bypass the given feature or in worst case scenario, if lacking error handling, the application crashes. A simple implementation on permission checking is shown in Code Snippet 3: Runtime permission check for contacts.

```
//Check if have access to contacts, if not asks for permission
if (ContextCompat.checkSelfPermission(thisActivity,
    Manifest.permission.READ_CONTACTS)
    != PackageManager.PERMISSION_GRANTED) {

    // Permission is not granted
    // Should we show an explanation?
    if (ActivityCompat.shouldShowRequestPermissionRationale(thisActivity,
        Manifest.permission.READ_CONTACTS)) {

        // Show an explanation to the user *asynchronously* -- don't block
        // this thread waiting for the user's response! After the user
        // sees the explanation, try again to request the permission.
    } else {

        // No explanation needed; request the permission
        ActivityCompat.requestPermissions(thisActivity,
            new String[]{Manifest.permission.READ_CONTACTS},
            MY_PERMISSIONS_REQUEST_READ_CONTACTS);

        // MY_PERMISSIONS_REQUEST_READ_CONTACTS is an
        // app-defined int constant. The callback method gets the
        // result of the request.
    }
} else {

    // Permission has already been granted
}
```

Code Snippet 3: Runtime permission check for contacts (Android, Request permission 2018)

2.8.1.2 Features

Applications are required to announce which software and hardware features it is require, so that Android can keep track of which system features the application is dependent on. It is also possible to define if it is a critical feature or non-critical feature, giving the application more flexibility. One might use the camera, but the app is not dependent on it, unless the application is camera application. An example of declaring features is presented in Code Snippet 4: Declaring feature.

```
<uses-feature
    android:name="android.hardware.camera"
    android:required="false" />

<uses-feature
    android:name="android.hardware.location.gps" />
```

Code Snippet 4: Declaring feature

3 MATERIALS AND METHODS

This chapter describes the materials and methods used in the project.

3.1 PROJECT ORGANIZATION

This section will describe how our project was organized and how we worked to reach our goal.3.2

3.2.1 PROJECT TEAM

Because our project was self-defined by the team, we had no natural project owner, as per the Scrum methodology. The Scrum guide states that the product owner may only be one person, so we chose one of the team members to act as the sole product owner in this project (Schwaber og Sutherland 2017).

We did not have a formal Scrum master either. Our solution was to let each individual team member assume the role of Scrum master for a month before the next member took over. In doing this, we did not technically break any of the rules in the Scrum methodology, as the Scrum guide does not explicitly state that a Scrum master or product owner may not be part of the development team (Schwaber og Sutherland 2017).

In the beginning of the project, the team started working on separate components. One team member took responsibility for getting the microcontroller and all its components and libraries to work, another member took responsibility for the physical server, along with the servlet container and general server networking, while the final member took responsibility for the Spring Boot backend and the database architecture. The mobile application was a cooperation by all members of the development team.

3.2.2 WORKFLOW

The workflow is based on the scrum process as seen in Figure 14. The product owner is responsible for an ordered list of everything we know is needed in the product. This list is known as the product backlog.

The heart of scrum is a sprint, a time-box of 1-4 weeks consisting of sprint planning, daily scrums, development work, sprint review and sprint retrospective.

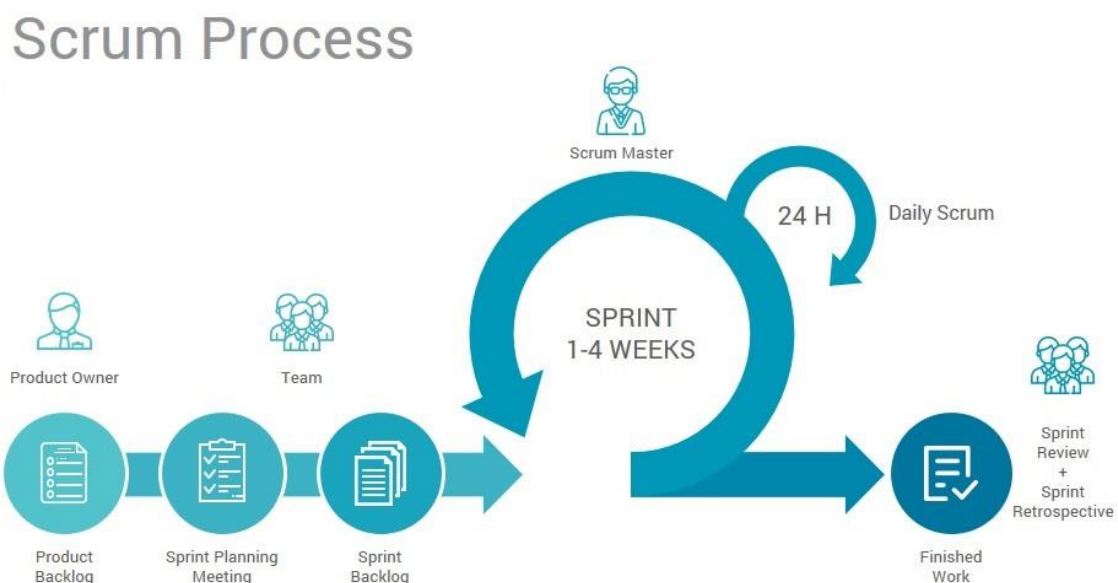


Figure 14: Scrum process (Brainhub u.d.)

BACHELOR THESIS

In the first part of a sprint, the entire scrum team comes together for a sprint planning meeting to decide the work to be performed in that sprint. Items selected from the product backlog represents the sprint backlog, a visible real-time view of work the development team plans to accomplish during the sprint. Once the sprint begins, the duration is fixed and cannot be changed, but the development team has authority to change the sprint backlog during the sprint.

Every day, the development team gather together for an event called daily scrum, also known as stand-up meeting. The daily scrums are maximum 15 minutes and help improve communication, identify obstacles, improve knowledge, eliminate other time-consuming meetings and promote quick decision-making.

A sprint review is held at the end of the sprint to inspect the increment. The sprint review is an informal meeting where the scrum team demonstrate a workable version to the stakeholders and collaborate about what have been done in the sprint, general status of the project and new ideas for the next sprint.

The final step before the next sprint can start is a sprint retrospective described in section 3.15.4.

This sprint process is repeated, and the scrum master ensures that scrum is followed through all steps (Schwaber og Sutherland 2017).

3.2.2.1 Planning Poker

Planning poker is an agile estimating technique for weighting the workload of each issue in relation to other issues. It is a dynamic process where each team member assign story points to each issue from a given scale. Each member picks a card representing the time or duration they think an issue is worth. The cards are revealed simultaneously before the estimate is discussed, and a consensus is reached. If there is a big deviation, team members can repeat the process after the discussion. This way, smearing effect within the group is avoided and team members reach a mutual understanding of the workload (Grenning 2002).

3.2.3 PAIR PROGRAMMING

Pair programming is a practice introduced by extreme programming where developers work in pair to develop software. The use of pair programming supports the idea of collective ownership and responsibility. Each line of code is looked at by at least two people, so pair programming works as an informal review process. Software errors can be discovered in an early stage and fixed without the use of time consuming code inspections and reviews. Another advantage is that developers can give input when refactoring can be beneficial (Sommerville 2016).

A study using student volunteers found that productivity with two people working independently seems to be comparable with pair programming. However, studies with experienced developers did not replicate these results (Sommerville 2016). Based on this, we could benefit on the use of pair programming in some situations.

We used pair programming for sharing knowledge and helping each other when stuck on a specific problem. When discovering a problem from our daily scrums, we would go back to our workstation and use pair programming to solve this specific problem. This way we reduced the risk of getting stuck and wasting too much time on one problem.

3.3 PRE-PLANNING

When planning the project, we first wrote a preliminary report (Appendix 2). This report provided a basic outline for what the final product would do, what it would consist of, and how we would create it.

3.4 ARCHITECTURE

This section details the components that make up the system's architecture.

3.4.1 SPRING BOOT

Spring Boot is an open-source solution for creating production-grade Spring-based applications. It is configured out of the box with the options that Spring view as "best". This leads to minimal configuration on the user's part, making it easy to get started (Webb, et al. 2018).

Spring Boot is just a part of the Spring framework. The Spring framework itself is an IoC (Inversion of Control) web container (See section 2.6.1). The IoC part means the responsibility for making things happen is moved into the framework, which results in less work for the developer, who can then focus on application code (R. Johnson 2005).

Spring and Spring Boot offer a lot of functionality for a wide variety of tasks. You can easily get started in seconds with REST applications, web sockets, streaming, web applications, and more. It also offers simplified security, tracing, metrics, health status. Additionally, it offers good support for SQL (Webb, et al. 2018).

3.4.2 POSTGRESQL

PostgreSQL is an open source relational database system available for all major operating systems. It is known for its reliability, data integrity, and correctness. It offers a wide range of features, functionality and supports many different data types. Additionally, it supports storage of pictures, video, sounds, and more (PostgreSQL u.d.).

PostgreSQL is highly sophisticated and customizable. Some of the features it can offer include recovery for a specific point in time, asynchronous replication, online backups, auto-increment columns through a sequence, table inheritance, and many more. Postgres can run stored procedures in over 12 programming languages, and it allows the creation of custom data types (PostgreSQL u.d.).

In addition to having many specific features, PostgreSQL can also ensure a great deal of security in running transactions. In being ACID-compliant, PostgreSQL rolls back changes if one statement in a transaction fails, ensures the database is in a valid state, ensures concurrent execution of transactions, and ensures data is not lost in event of crashes or power loss (Haerder og Reuter 1983).

3.4.3 JDBC TEMPLATE

JDBC (Java Database Connectivity) is an API for database-independent connectivity between Java and a variety of databases (Oracle u.d.). JDBC Template aims to simplify the usage of JDBC, and help avoid prevalent errors. It offloads work from developers by taking care of the central JDBC workflow. It executes SQL queries, initiates iteration over the results, and catches exceptions (Spring u.d.). Code Snippet 5 **Error! Reference source not found.** demonstrates a simple use case of JDBC Template:

```
public void updateUser(User user) {
    String sql = "UPDATE user_account SET name = ?, phone = ? WHERE email = ?";
    jdbcTemplate.update(sql, user.getName(), user.getPhone(), user.getEmail());
}
```

Code Snippet 5: Method updateUser in class UserRepository

The above method takes a User object as a parameter. It constructs the query it will run, with question marks where the variables will be inserted. A call to update() then executes the transaction, while managing resources and other tasks in the background.

BACHELOR THESIS

3.4.4 REST

REST (Representational State Transfer) is an architectural style that is based on several principles describing how resources exchanged over a network are addressed and defined. It was first described by Roy Fielding in his doctoral dissertation on architectural styles and design of network-based software architectures (Douglas u.d.).

RESTful applications can be characterized by the following traits (R. T. Fielding 2000):

- **Client-Server:** Separation of user interface and data storage. This Improves portability of the user interface, and improves scalability by simplifying the server components.
- **Stateless:** Server does not keep any state regarding clients. Each request to the server must contain all the necessary information needed to fulfill that request. This makes the application more scalable since the server can more quickly free resources, and relieves the need to manage resources across requests. Requests and responses are typically HTTP-based, using basic HTTP methods such as POST, GET, PUT, etc.
- **Cache:** Responses from the server are marked as cacheable or non-cacheable. If a response is marked as cacheable, then the client can reuse the data from the response for similar requests later. The advantage of this is that it can reduce or eliminate some interactions, which improves efficiency and scalability.
- **Uniform interface:** There is a uniform interface between components. The system architecture is simplified and visibility of interactions to the server is improved. The implementations of functionality are decoupled from the service they provide the client. Individual resources are identified in each request, by for example using URIs. Information is typically conveyed in HTML, XML, or JSON.
- **Layered system:** The architecture is designed in such a way that each component cannot know of anything beyond the immediate layer with which they are interacting. This reduces the overall system complexity and improves scalability. However, they also add some overhead and a certain degree of latency to the processing of data.
- **Code on demand:** Additional code in the form of scripts or applets can be downloaded from the server and executed on the client machine for extended functionality. This improves system extensibility, but also reduces visibility.

BACHELOR THESIS

3.4.5 SERVLET CONTAINER

The basic idea of a servlet container is to use Java to generate a web page dynamically on the server side. Essentially, the servlet container is part of a web server responsible for interacting with the servlets as seen in Figure 15.

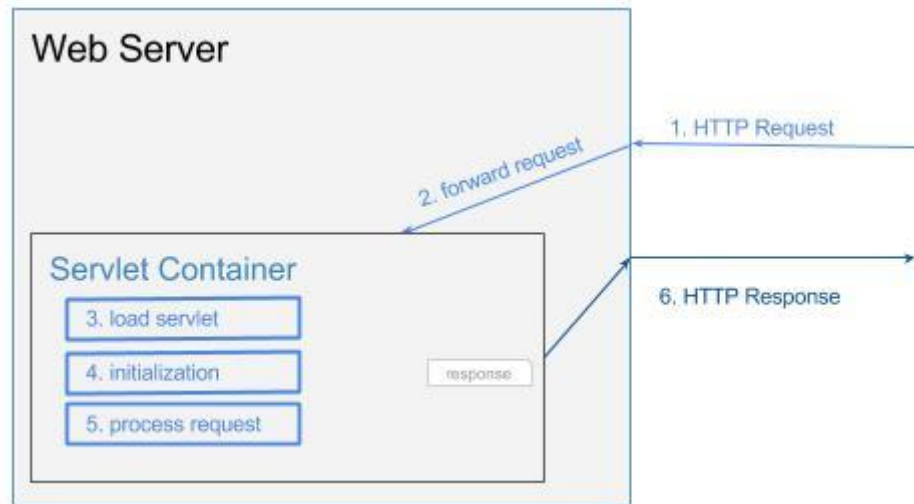


Figure 15: How web server and servlet container process requests (Programcreek 2013)

3.4.6 JAVA SERVLETS

A servlet is a server-side program running within a web server. The servlets receive a request and dynamically generates a response based on the request.

The Servlet interface defines three methods for the life cycle of a servlet, as seen in Figure 16. `init()` initialize a servlet, `service()` is used to service requests, and `destroy()` to remove a servlet from the server (Oracle 2016).

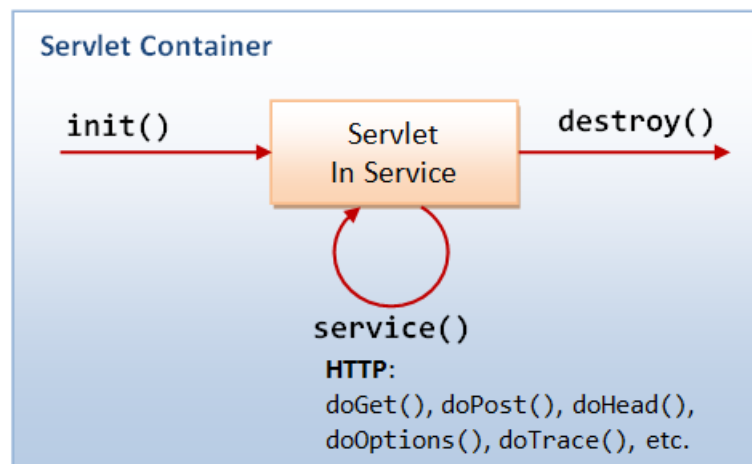


Figure 16: Servlet lifecycle (Servlet lifecycle 2012)

3.5 PROGRAMMING LANGUAGES

In this section we will discuss the programming languages the project is built from, as well as what defines those languages.

3.5.1 ARDUINO

“Arduino is an open-source electronics platform based on easy-to-use hardware and software.” (Arduino 2018) The idea behind this language is to create an easy approach toward programming and hardware configuration for microcontrollers. From small and easy systems, to large and complex systems. Arduino has been a platform for creative digitalization, by offering an easy approach for creators worldwide. Arduino was developed by Ivera Interactive Design Institute to create an easy tool for fast prototyping, specifically for students without any programming experience.

Arduino programming language is based on the open-source programming language Wiring (Arduino 2018), which is a framework for microcontrollers. Wiring is based on Processing, which is an open project by Ben Fry and Casey Reas (Wiring 2018). Ultimately, under the hood of the Arduino is a WinAVR compiler, which compiles programs in C and C++ (Igendel 2014).

3.5.2 JAVA

Java is both a compiled and an interpreted object-oriented programming language. It was developed under the supervision of James Gosling and Bill Joy at Sun Microsystems in the early 1990's (Niemeyer og Leuck 2013).

Java is popular for its speed and portability. The portability stems from the fact that Java code is compiled into a universal format; instructions for a Java Virtual Machine (JVM). This enables developers to write code that can run on any platform that supports the JVM, which includes anything from home computers to toasters. They only have to write this code once, without the need for adjusting it for every system (Niemeyer og Leuck 2013).

3.6 CODING CONVENTION

The code convention in this project should follow the guidelines from the object-oriented programming theories from Objects First with Java (Barnes 2012). We also have drawn much of our convention from Clean Code (Martin 2008).

3.6.1 NAMING

- Class names should start with a capital letter followed by small letters. Capital letters should be used to distinguish between words. For example 'MyClass'.
- Method and variable names should start with a small letter. Capital letters should again be used to distinguish between different words. For example 'myMethod()'.
- Final/Constant variables should be named using only capital letters, and underscore to indicate space between words. For example 'PI_SQUARED'.

3.6.2 STRUCTURE

Curly brackets should start at the same line as the method/class name. For example:

```
/*Get a random number between 0 and 100*/  
private int getRandomInteger () {  
    return Math.random() * 101;  
}
```

Code Snippet 6: Method convention

BACHELOR THESIS

Nested functions should be indented with tabulator for each layer. For example:

```
/*Check if number is positive and larger than*/  
private boolean isPositiveAndLargerThan(int test, int than){  
    boolean result = false;  
    if(test > 0){  
        if(test > than){  
            result = true;  
        }  
    }  
    return result;  
}
```

Code Snippet 7: Nested methods convention

To distinguish between methods, we use one empty line. This allows us to easier navigate between different methods. Shown as the following:

```
private int getId(){  
    return this.id;  
}  
  
private String getName(){  
    return this.name;  
}
```

Code Snippet 8: Method separation convention

3.6.3 COMMENTS

Our convention on comments are closely related to Clean Code (Martin 2008), where we aim to write our programs with only necessary comments where good naming does not adequately describe the functionality of the program. We also aim to comment our more complex methods and classes. We follow this approach because comments often lead to unnecessary noise in the code, plus the comments need to be updated once a method is refactored. This often leads to potential outdated comments, which does not give us any added information. Code Snippet 9 and Code Snippet 10 demonstrate what refactoring commented code can result in.

```
// Check to see if the employee is  
// eligible for full benefits  
if ((employee.flags & HOURLY_FLAG) &&  
    (employee.age > 65))
```

Code Snippet 9: X Bad code with comment. Should be refactored (Martin 2008)

```
if (employee.isEligibleForFullBenefits())
```

Code Snippet 10: ✓ Better code, with no unnecessary comment (Martin 2008)

BACHELOR THESIS

3.6.4 LANGUAGE

All our programming and comments should be in English, since English is the predominant language convention in programming. Additionally, it lets us at a later stage collaborate with an international environment.

3.6.5 IMPLEMENTATION

It was up to each developer stay true to our code convention, and to write clean code. Our basis was to write as clean code as possible from day one, and use refactoring when the code became unclear and messy. From time to time we also did some refactoring of each other's code, which would highlight potential misunderstandings and bad design decisions. From the start of the project our supervisor Girts Strazdins encouraged us to not include header fields such as version, author and date due to the collaborative and iterative nature of this project.

3.7 EXTERNAL LIBRARIES – MICROCONTROLLER

This section will describe the external libraries that were used on the microcontroller.

3.7.1 GSM

The GSM library enables the Arduino to perform phone functionality. For example sending and receiving SMS, place and receive phone calls, and access the phone GPRS network. The library abstracts the low-level communication between the SIM card and modem. It relies heavily on the Software Serial library for this communication (Arduino, GSM 2018).

3.7.2 ADAFRUIT_GPS_ALT

The Adafruit alternative GPS library is based on the original Adafruit GPS library, which performs the much of the communication with the GPS module. For instance, reading stream data in a background interrupt which then parses it. The library returns a stream of parsed NMEA sentences, which contains the GPS information from the attached GPS module (Adafruit, GPS Library 2018). The alternative library uses an alternative Software Serial implementation, which does not conflict with the GSM library (Rick, Adafruit Support 2014).

3.7.3 ALTSOFTSERIAL

The alternative software serial library implements an alternative software serial which allows the Arduino program to use two similar software serials implementations (PJRC 2018). The latest development is located at Github (PJRC 2017).

3.7.4 MEMORYFREE

The external MemoryFree library makes it possible to get an indication on memory usage on the microcontroller in real-time. This allows for resource management for Arduino programming and opens for more control over the Arduino debugging process (Muthu 2012).

3.7.5 ARDUINODES

The ArduinoDES library is designed to be fast and efficient and makes it possible to encrypt and decrypt using symmetric encryption algorithms DES and 3DES. (Spaniakos 2015)

3.8 EXTERNAL LIBRARIES - BACKEND

The sections in this chapter describe the external libraries which were used on the backend.

3.8.1 BCRIPT

Bcrypt is a file encryption utility primarily used for hashing passwords. It has become widely popular, and is the standard password hashing tool for many systems (PassLib u.d.).

Bcrypt hashes passwords into a fixed-sized string of 60 characters. A nice feature of Bcrypt is that the generated hashes have a built-in salt. As developers, this means we do not have to take any measures to generate and store the salts ourselves. A hashed password is shown in Code Snippet 11.

```
$2b$12$GhvMmNVjRW29ulnudl.LbuAnUtN/LRfe1JsBm1Xu6LE3059z5Tr8m
```

Code Snippet 11: Password hashed with Bcrypt

This string consists of three fields, separated by the '\$' character. **2b** indicates the version of Bcrypt used. **12** is a logarithmic parameter which indicates how many times the internal hash function is called. The last field is again separated into two subfields. The first 22 characters, **GhvMmNVjRW29ulnudl.Lbu**, is the randomly generated salt. This is unique for each user. **AnUtN/LRfe1JsBm1Xu6LE3059z5Tr8m**, the final 31 characters, is a checksum, generated by the internal algorithm when hashing the salt and password combination (PassLib u.d.).

3.8.2 LIQUIBASE

Liquibase is an open source library for managing changes in a database. It supports multiple database types, which ensures that you do not have to tailor your changes to a specific database dialect (Shmeltzer 2017).

Using Liquibase involves defining a file in the project structure that contains all the database changes. You can write it in XML, YAML, JSON or SQL. These changes can be anything from defining a new table, inserting data, or deleting relations. Every changeset that is not written in plain SQL is translated to the correct SQL dialect, meaning all members of the team can run different database systems without difficulty. When others download this changefile, Liquibase runs the newest changes once and only once (Liquibase u.d.). A changeset written in XML is shown in Code Snippet 12.

```
<changeSet author="developer" id="1">
  <createTable schemaName="public"
    tableName="device">
    <column name="id" type="integer">
      <constraints primaryKey="true"/>
    </column>
    <column name="user_email" type="varchar(100)">
      <constraints foreignKeyName="fk_device_user"
references="user_account(email)"/>
    </column>
  </createTable>
</changeSet>
```

Code Snippet 12: Liquibase changeset

The above code simply creates a new table with two columns.

One of the most attractive features of Liquibase is the ability to make database changes that applies to everyone's local database. Instead of creating a change script and making sure everyone on the team runs it on their database, the changeset you define in Liquibase is automatically run when you start the application. Liquibase keeps track of what changes are run, so that no changes are

BACHELOR THESIS

accidentally executed twice (Liquibase u.d.). This ensures that changes you make to the source code that deals with database transactions do not cause failures for someone that has forgot to apply the correct database changes.

3.8.3 JSON WEB TOKEN

JWT (JSON Web Token) is an open standard based on JSON used for creating and verifying access tokens (Auth0 u.d.). For a general explanation of access tokens, see section 2.2.2.

JWTs are lightweight, in that they are small and self-contained. They are small enough to easily be sent through the header in a HTTP request, and they are self-contained by having all the necessary information regarding the user in the body of the token, which reduces work on the server that authenticates them (Auth0 u.d.).

A JWT consists of three parts; the header, the payload, and a signature, each separated by a punctuation mark. An example token is given in Code Snippet 13.

```
eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJhcm5lQG50bnUubm8iLCJleHAiOjE1MjUzNDY5MzJ9.DFl0JuRBRlfrVM0HxE3WB4tTeGkbZmGAEg0B01-RNEIHof5V_Z9Adnrywwh8DDOQRg6Hou0CCLXbQARaU8KSA
```

Code Snippet 13: Encoded JSON web token

If we decode the first part, the header, we get the information shown in Code Snippet 14.

```
{  
  "alg": "HS512"  
}
```

Code Snippet 14: Decoded header

This tells us which hashing algorithm is used. In this case it is HMAC SHA512.

If we decode the payload, we get the claims shown in Code Snippet 15. Claims are essentially assertions about the user who carries the token.

```
{  
  "sub": "arne@ntnu.no",  
  "exp": 1525346932  
}
```

Code Snippet 15: Decoded payload

This token contains the claims “sub” and “exp”, which are short for “subject” and “expiration time”, respectively. These are *registered claims*, which are claims that are not mandatory to include, but recommended. The subject claim tells us who the holder of the token is. In this case, we have used an email address as a unique identifier. The expiration time is the number of seconds since the Unix epoch, January 1st, 1970. This particular claim tells us that the token expires on May 3rd, 2018, at 13:28. After that, the token will no longer be accepted, and a new one will have to be issued.

BACHELOR THESIS

The signature is a string generated by hashing the encoded header, encoded payload, and secret. See Code Snippet 16.

```
HMACSHA512(  
    base64UrlEncode(header) + "." +  
    base64UrlEncode(payload),  
    secret  
)
```

Code Snippet 16: Signature

In this case, the secret is only known by the server authenticating the token. We execute this step to ensure the token was not changed along the way. To create the signature, we must use the same hashing algorithm as specified in the header.

3.8.4 PROJECT LOMBOK

Project Lombok is a library for reducing boilerplate code in Java. It provides a collection of annotations that can replace common, unchanging code. For example, instead of manually writing getters and setters for all the fields in an entity class, you can instead place an annotation at the top of the class. The methods will be generated and hidden, but can still be called normally. This reduces a lot of visual noise (Kimberlin u.d.).

Notable annotations include (Project Lombok u.d.):

- **@NonNull** – Generates a null-check for parameters in a method.
- **@Getter/@Setter** – Generates getters/setters on annotated fields. Can also be used on class to generate getters and setters for all fields.
- **@Data** – Generates getters, setters, toString(), equals(), and a constructor with one parameter for fields that need special handling.

3.9 EXTERNAL LIBRARIES – ANDROID

This section will describe the external libraries that were used on android.

3.9.1 GSON

Gson is an open source library for effortlessly converting Java objects to their JSON representation, and vice versa (Gson 2017).

Using Gson is as simple as instantiating a Gson object and calling its toJson() and fromJson() methods. toJson() reads the Java class and creates a JSON representation with the same field names. However, it is also flexible in that it allows you to override the field names by using annotations. fromJson() converts a JSON string to a specified Java object that shares its field names (Gson 2017).

3.9.2 CIRCLEIMAGEVIEW

CircleImageView is an open source library created by Henning Dodenhof for easily displaying images as circular on Android (Dodenhof 2018).

CircleImageView is efficient in that it does not create a copy of the original image and can be used with any drawable image. Using the library is as simple as adding a CircleImageView object to the layout file and specifying its attributes. The image that uses this tag will appear as a circle in the Android application. The library is licensed under the permissive Apache license (Dodenhof 2018).

BACHELOR THESIS

3.9.3 SPECTRUM

Spectrum is an open source library created by Nathan Walters to make it easy to implement a way for the Android user to pick a color from a color pallet (Walters 2016). This library is subject to the MIT license.

The Spectrum library includes multiple ways to implement a color picker, such as a pop-up dialogue in an activity, a preference option, as well as an AppCompatActivity support and lastly as an integrated widget. To instantiate a color picker, the developer first needs to define which colors should be included in the pallet. This could be done from the colors.xml, and made even easier with an array of color references.

3.9.4 GRAPHVIEW

GraphView is an open source Android library for creating flexible diagrams, created by Jonas Gehring. It supports the creation of line graphs, bar graphs, point graphs, and even custom graph types. It is designed to be simple to get up and get started, meaning you can quickly set up and display a graph in any layout (Gehring 2018).

To implement it, you simply add a GraphView object to the layout file. To fill the graph with data, you programmatically add a series of data points to the instantiated object. Data points are normally number values, but dates and custom types are also supported (Gehring 2018).

3.10 DEVELOPMENT TOOLS

This section describes the different development tools that were used in the project.

3.10.1 INTELLIJ IDEA ULTIMATE

IntelliJ IDEA Ultimate is an integrated development environment (IDE) for writing software. It is a proprietary tool created by JetBrains (JetBrains 2018). We primarily used it in the development of the backend part of the system.

IntelliJ offers a great deal of useful features and language- and framework support. It provides agile navigation, error analysis, refactoring assistance, and it continuously analyzes your code to provide programming assistance. It offers code and chain completion, which helps you write code faster by giving suggestions based on what you are writing. Additionally, it offers an inline debugger, which shows values of variables directly in the source code. Finally, it has good support for the Spring framework, database tools, and SQL (JetBrains u.d.). These are just some of the features it provides and are the primary reasons we chose to use it as our development environment.

3.10.2 ANDROID STUDIO

Android Studio is a free IDE based on the community version of IntelliJ IDEA, developed by Google and JetBrains. It is designed specifically for Android application development (Google u.d.). We used Android Studio in the development of the mobile application.

Android Studio offers many of the same features as IntelliJ IDEA, and an additional set of tools for Android. It provides an Android emulator that lets you simulate a range of different devices to see how your app would look and run on those devices. It also provides several other useful features, such as a translations editor for localizing applications, and a graphical layout editor for drag-and-drop editing of user interfaces (Google u.d.).

3.10.3 BITBUCKET

Bitbucket is a scalable distributed version control system from Atlassian, making it easy to collaborate with your team. In the bitbucket cloud, Git repositories can be stored giving benefits such as conflict resolution, offsite source code backup, and the possibility to roll back and undo changes to source code (Bitbucket 2018).

BACHELOR THESIS

3.10.3.1 Gitflow

Gitflow is a Git workflow design. See Figure 17. Defining a branching model designed around the project release, it dictates which branches to set up and how to merge them. This design helps the team distinguish between code in development and finished work (Bitbucket 2018). A description of how to use Gitflow is shown in Table 3.

Description of branch	
Master	Only tracks released code.
Develop	Contains the complete history of the project.
Feature	Used for new development.
Release	Used for preparation of a new production release.
Hotfix	Used to create emergency fixes.

Table 3: How to use branches in Gitflow

The overall flow of Gitflow (Bitbucket 2018):

1. Develop branch is created from master
2. Release branch is created from develop
3. Feature branches are created from develop
4. A Finished feature is merged into develop
5. When the release branch is done it is merged into develop and master
6. A hotfix branch is created from master if an issue in master is detected
7. A finished hotfix is merged into develop and master

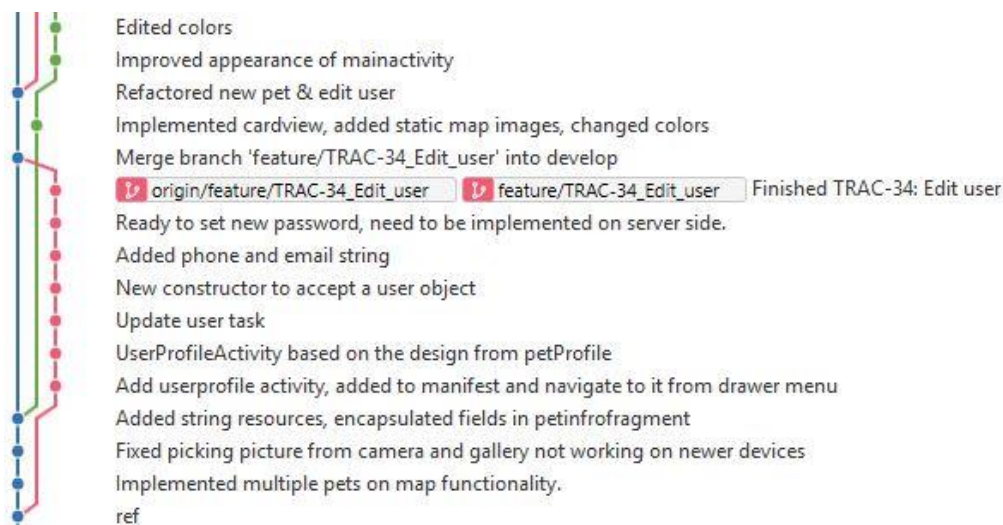


Figure 17: Example of Gitflow in our project

3.11 SUPPORT TOOLS

This section describes the different supporting tools that were used in the project.

3.11.1 NETWORK ANALYZER

Wireshark is a widely used network protocol analyzer that lets you see what is happening on your network at a microscopic level (Wireshark u.d.). Wireshark can be used to capture live network packets and to display packet data as detailed as possible. In our case, we used offline analysis with packet data imported from text files to be analyzed.

BACHELOR THESIS

On the ubuntu server with only a CLI available, we used a command-line version of Wireshark named TShark. It supports the same options as Wireshark described, but it is designed for use when an interactive user interface is not available (Wireshark u.d.).

Figure 18 shows how to start a capture in TShark, dumping all packet data to a pcap file.

```
trackadmin@trackapet:~$ sudo tshark -w /home/trackadmin/tshark/dump-30052018.pcap
```

Figure 18: Start a TShark capture

3.12 TESTING METHODS

In this section we will describe which parts of the system will be tested, along with which methods will be used to conduct the testing.

3.12.1 UNIT TESTING

Unit testing is used to test individual pieces of the software implementation. These are lightweight tests that give direct feedback to the developer if the test fails (see section 2.5.5). It is relative easy to find critical bugs and malicious behavior with unit testing.

3.12.2 INTEGRATION TESTING

Integration tests the software towards a web-service. These tests mock user interaction and checks if the web-service response accordingly (Sommerville 2016).

3.13 TESTING TOOLS

A testing tool is a library which offers an environment to write tests and perform testing of software. This section will highlight some of the testing tools we have used to test our system.

3.13.1 MOCKITO

Mockito is a Java testing framework for writing unit tests (Mockito 2018). To test an Android application and android specific methods, the testing environment must have access to a context, which Mockito provides a light-weight instance of.

3.13.2 JUNIT

JUnit is a unit testing framework to write light-weight and repeatable tests (JUnit 2018). JUnit offers a way to test expected results towards actual results. All methods which are to be tested must be annotated with `@Test`. JUnit also provides a setup phase, `@Before`, and a breakdown phase, `@After`.

3.14 MATERIALS

To develop our prototype, we needed the equipment shown in Table 4:.

	Name	Link
Microcontroller	Arduino Uno Rev3	https://no.rs-online.com/web/p/products/7154081/
GSM module	Arduino GSM shield 2	https://no.rs-online.com/web/p/processor-microcontroller-development-kits/8659010/
GPS module	Adafruit Ultimate GPS Breakout	https://no.rs-online.com/web/p/gps-modules/9054630/
Mobile subscription	Telenor Go Norge	https://www.telenor.no/bedrift/iot/m2m/

Table 4: Hardware components

BACHELOR THESIS

3.14.1 ARDUINO UNO REV3

Arduino Uno rev3 is based on the ATmega328P (Arduino, arduino.cc 2018), which is a high-performance Microcontroller 8-bit AVR RISC-based microcontroller combined with 32KB flash memory (Microchip.com 2018). The Arduino comes with 14 digital I/O pins, of which 6 are PWM enabled. It provides 6 analogue inputs, a 16 MHz quartz crystal, USB connection, a power jack, an ICSP header (Waldby 2017), and a reset button. The Arduino Uno delivers a ready out-of-the-box microcontroller which functions as a programming platform with significant customization flexibility at a low cost. Since Arduino is a well-established microcontroller, custom component is easy to come by.

3.14.2 ARDUINO GSM SHIELD 2

The Arduino GSM shield enables the microcontroller to use the mobile network to send SMS, make phone calls and access the internet using the onboard antenna. The microcontroller communicates with the GSM module using AT commands (Arduino, Arduino.cc 2018). The AT commands are abstracted using the built-in GSM library from Arduino (Arduino, GSM 2018). AT commands are basically the standard for communicating with GSM and GPRS network (QT 2009). The GSM shield uses pin 2 and pin 3 for software serial communication with quad-band M10 GSM/GPRS chip, which operates at GSM850MHz, GSM900MHz, DCS1800MHz and PCS1900MHz. The GSM shield has a downlink/uplink capacity of maximum 85.6kbps. The GSM shield also needs a SIM card backed by a network provider to operate.

3.14.3 ADAFRUIT ULTIMATE GPS BREAKOUT

The Adafruit Ultimate GPS breakout is built around a MTK3339 chipset, which is an All-in-One GPS system on a chip (Adafruit, Adafruit.com 2018). The MTK3339 offers a high-performance GPS chip that works great in dense urban environments, and has high tracking sensitivity (Mediatek Labs 2018).

The Adafruit GPS breakout also offers regulator so that the module can be powered from a range of 3.3 – 5V, as well as a practical LED to indicate fix state of the module. The LED blinks at a 1Hz while searching for satellites and at a 15Hz when it has a fix. Other features include connectors to external antenna and built-in datalogging, which can log data for up to 16 hours to preserve energy consumption on the microcontroller.

3.15 DOCUMENTATION

This section will describe tools and methods used for project management.

3.15.1 CONFLUENCE

Confluence is a content collaboration tool, developed by Atlassian. Confluence offers a central location to store your team's work and a way to organize it in a logical and simple-to-find way (Atlassian u.d.).

We used Confluence to store and share documentation, like meeting notes, retrospectives, photos and wireframes. Confluence has specific ways to handle all these kinds of documentation, including editors for wireframes and various diagrams. Figure 19 shows the dashboard for our project, which is what you see when you log in to Confluence. On the left, you have quick access to rooms for some of the different types of documentation. In the middle, you have a collection of some of the diagrams and photos stored on Confluence.

BACHELOR THESIS

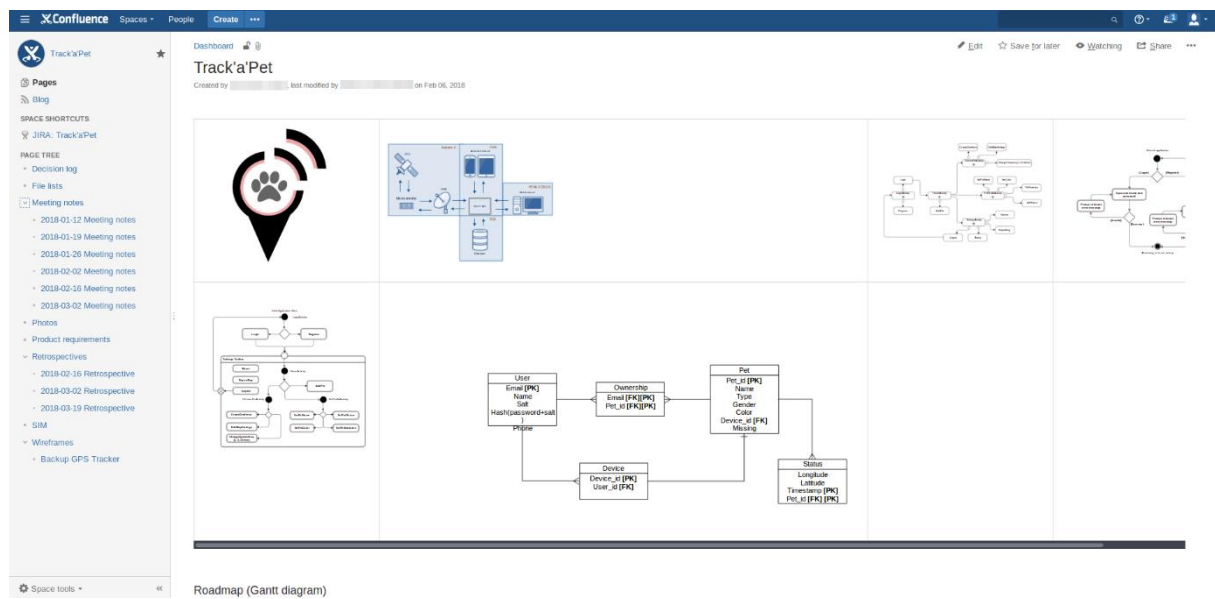


Figure 19: Screenshot of Confluence dashboard

3.15.2 JIRA

Jira is a collaborative agile management tool which offers a workspace to plan and perform agile software development. Jira also offers many support tools to plan and review the progress of the project, such as retrospective and estimated burndown charts. However, the main functionality of Jira is to provide a digitalized version of sticky notes, where each task can be broken down into separate issues and tracked in real time.

From Figure 20 we can see a section of the whole backlog, where all current tasks preside. From here the team can set up a sprint, shown in Figure 21. When a sprint is started, issues are assigned to a person and moved to 'in progress'. Once the issue is resolved they are moved to 'done'. This offers the team a straightforward way of keeping track of progress.

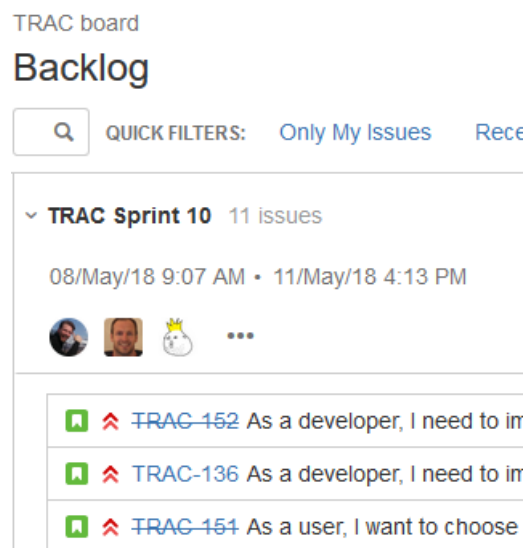


Figure 20: Jira – Backlog

TRAC Sprint 10

QUICK FILTERS: [Only My Issues](#) [Recently Updated](#)

The screenshot displays a Jira active sprint board with three columns: To Do, In Progress, and Done. The 'Done' column contains one issue, TRAC-58, which is marked as 'DONE' and has a sub-task description: 'As a user, I want to receive a notification when my pet leaves the geofence'. The 'To Do' column contains three issues:

- TRAC-136**: As a developer, I need to implement a serial-number on the microcontroller. Category: Hardware [tracker]. Priority: 16.
- TRAC-111**: As a developer, I need to implement functionality for pet graph fragmentation. Category: Mobile. Priority: 22.
- TRAC-152**: As a developer, I need to implement picture functionality on register pet activity. Category: Mobile. Priority: 12.

Figure 21: Jira – Active sprint

3.15.3 MINUTES OF MEETING

For the weekly meeting with our supervisor, meeting notes were written in Confluence to help plan the meeting and share our notes for team members not present.

3.15.4 RETROSPECTIVE

A sprint retrospective meeting is held after finishing a sprint. The team discusses what went well and what could be improved. The goal is to improve product quality by improving work processes, and at the end of a retrospective the scrum team should have identified what to improve in the next sprint (Schwaber og Sutherland 2017).

4 RESULTS

This chapter will give an in-depth view of our process in the whole project, rooted in the theoretical part of the report.

4.1 ARCHITECTURE

In these sections, we will describe how the system is constructed, and why we chose to construct it the way we did. Figure 22 shows a very general overview of the system as a whole.

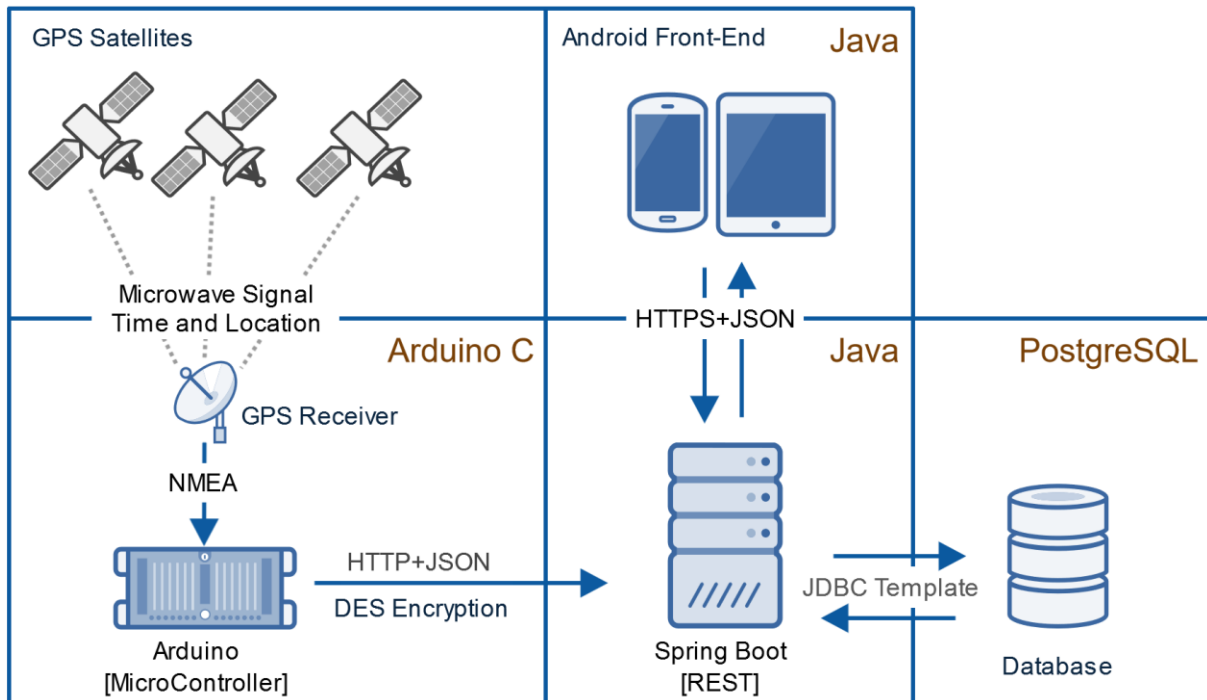


Figure 22: General system architecture

Firstly, the GPS module acquires a location using data received by GPS satellites. The module parses the data into NMEA sentences and encrypts and sends the location data to the REST service. The REST service receives HTTP requests over an encrypted connection from Android devices, and responds in the same fashion. The backend server communicates with the production database over JDBC Template.

4.2 BACKEND ARCHITECTURE

The backend consists of a Spring Boot application, with a PostgreSQL database, deployed on an Ubuntu server located in the basement of NTNU in Ålesund. Figure 23 illustrates the general architecture of the backend. Custom components are marked with purple.

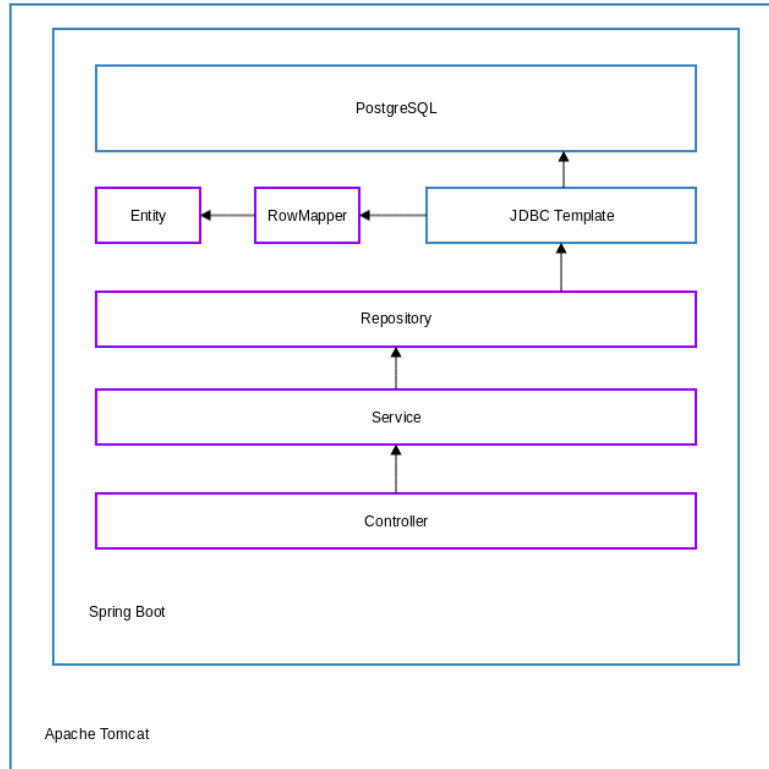


Figure 23: Backend architecture

This section will further detail the backend architecture and explain the choices that were made.

4.2.1 PROGRAMMING LANGUAGE AND PLATFORM

We chose to implement the backend in Java, as every team member were relatively proficient in the language. Java is the language that has been used in most of the programming courses, therefore everyone on the team knew how to utilize it for several use cases, such as networking, app development, and more.

When deciding on which framework to use, we considered both Java EE and Spring Boot. Our decision to use Spring Boot was rooted in the fact that it has increased in popularity over the last few years, compared to Java EE. Figure 24 shows the popularity of both search terms in Norway over the last five years. However, the decision was also impacted by Spring Boot's ability to easily let you get your environment set up and start developing your project instead of spending a lot of time configuring the framework, as explained in section 3.4.1.

BACHELOR THESIS

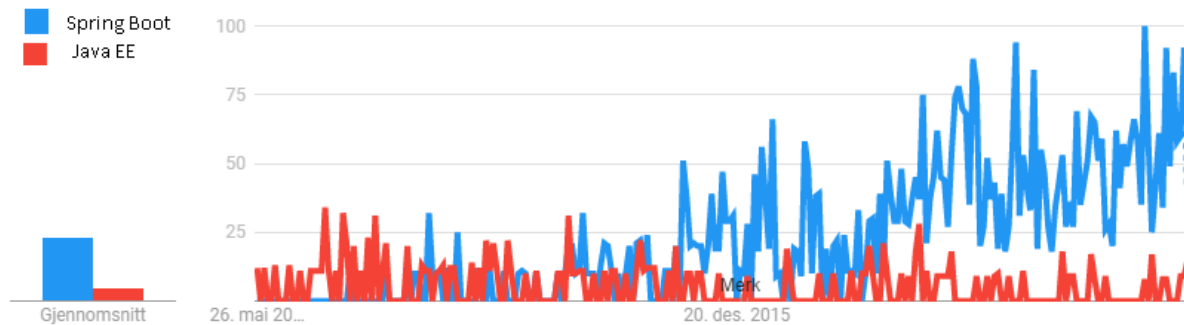


Figure 24: Popularity of search terms 'Spring Boot' and 'Java EE' represented by blue and red line, respectively (Google Trends u.d.)

4.2.2 OVERALL STRUCTURE

This section will document the overall structure that make up the backend.

4.2.2.1 Model-view-controller

The backend logic is implemented using the model-view-controller pattern. By doing this, we ended up with flexible code where we can easily change the implementation of almost any part of the system. The model part is made up of the repository classes that handle direct interaction with the database, the entity classes that represent data, and the service classes that handle other potential business logic. The view part is the response that is returned to the user. Finally, the controller part is made up of the controller classes. They receive and process user input.

4.2.2.2 REST API

The backend is implemented as a RESTful web service. As explained in section 0, this gives us a great deal of flexibility. It allows us to easier scale the system if it were to gain a large number of users, and it reduces complexity since we do not have to hold any state about connections or users. However, the primary reason we decided to implement the backend as a RESTful service is that we can easily implement other user interfaces without having to change or add a line of code to the backend logic. If we at some point decide to create a web interface for the application, we can simply have it take advantage of the end-points we created.

4.2.3 COMPONENTS

This section will document the main components of the backend.

4.2.3.1 RestController

The Controller classes are the end-points a client interacts with to send and retrieve data Code Snippet 17 shows the class declaration for UserController.

```
@RestController
@RequestMapping("/user")
public class UserController {
    // Methods omitted
}
```

Code Snippet 17: User Controller declaration

The `@RestController` annotation marks the class as a controller. Spring Boot will then treat it as a controller where each method returns a response body. The `@RequestMapping` annotation maps the URI `"/user"` to methods in this class. A specific method in this controller class is shown in Code Snippet 18.


```
@RequestMapping(value = "/register", method = RequestMethod.POST, consumes = "application/json")
public void createUser(@RequestBody User user) {
    userService.createUser(user);
}
```

Code Snippet 18: Method create User in class UserController

This method is mapped to the URI “user/register”, and we have specified that it only accepts the HTTP method POST. Additionally, we have specified that the method consumes a request body in the form of a JSON string. The parameter indicated that the JSON string will be read as a User object, meaning its field names must match the ones in the entity class User.

4.2.3.2 Service

Between the Controller and Repository classes, we have an added layer of business logic. In addition to providing some degree of looser coupling by not binding the repository classes directly to the controller classes, the service classes are there to perform any extra logic needed for the data to be fully usable by the controller classes. This way, we make sure the repository classes only have to worry about database interaction, and the controller classes only have to worry about user interaction. Code Snippet 19 gives an example of a service class declaration.

```
@Service
public class UserService {
    // Methods omitted
}
```

Code Snippet 19: UserService declaration

The @Service annotation tells Spring and others that will read the code that this is a component that holds business logic and interacts with the repository layer Code Snippet 20 shows how a method in a service class might look.

```
public void createUser(User user) {
    String hashedPassword = passwordEncoder.encode(user.getPassword());
    user.setPasswordHash(hashedPassword);
    userRepository.createUser(user);
}
```

Code Snippet 20: Method create User in class UserService

Firstly, the method accepts a user object, in this case always sent by a controller class. Then it performs some sort of operation on the data, in this case it hashes the password. Lastly, it sends the data down to the repository layer for it to be saved to the database.

4.2.3.3 Repository

The repository layer is responsible for handling interaction with the database. Code Snippet 21 shows the declaration for the class UserRepository.

```
@Transactional
@Repository
public class UserRepository {
    // Methods omitted
}
```

Code Snippet 21: UserRepository declaration

BACHELOR THESIS

The `@Transactional` annotation ensures that changes in the database are rolled back if an error occurs when interacting with it. The `@Repository` annotation tells Spring that this is a repository class, meaning it handles data storage and retrieval.

To interact with the database, we first used a tool built into Spring called JPA (Java Persistence API). However, this proved to be ineffective and produced code that was harder to read. Using it included defining all the database attributes and relations through annotations in the entity classes, which is challenging when you have mildly complex keys in the database. Additionally, the cluster of annotations does not immediately make it clear how entities are related to each other. Beyond that, communication with the database is abstracted to an interface, making it especially challenging if you want to perform queries beyond standard insertion and retrieval. We then decided to use JDBC Template instead.

JDBC Template lets you create your own SQL queries. These are often more effective, since you can make them as specific as you want. The code also became cleaner, since all the annotations from the entity classes could be removed. We also found the code became easier to understand, since you could see how the database was interacted with. A typical repository method makes use of JDBC Template, which is demonstrated in section 3.4.3.

To convert the data we retrieve from the database to a Java object, we use a Spring interface called `RowMapper`. To use it with JDBC Template, you pass a custom `RowMapper` object as an argument to JDBC Template's query method, which calls the overridden `mapRow()` method internally.

Code Snippet 22 demonstrates how it is used for a `User` object.

```
@Override
public User mapRow(ResultSet rs, int rowNum) throws SQLException {
    User user = new User();
    user.setEmail(rs.getString("email"));
    user.setPasswordHash(rs.getString("password_hash"));
    user.setName(rs.getString("name"));
    user.setPhone(rs.getString("phone"));
    return user;
}
```

Code Snippet 22: Method `mapRow` in `UserRowMapper`

4.2.4 DATABASE VERSIONING

A database in a system often changes to support new features or developments. To easily track and manage changes, we used a library called *Liquibase*. As explained in section 3.8.2, *Liquibase* automatically applies the changes defined in a changefile to the databases. We first wrote the initial database tables and attributes in a single changeset, so that when you first run the program, you do not have to worry about setting up the database yourself. All further changes were written in subsequent changesets that would automatically be applied when run for the first time. By doing this, neither the production database nor any local database would accidentally be left out of date, and team members did not have to run any changes themselves that others had defined.

4.2.5 HOSTING AND DEPLOYMENT

We chose to host our solution on a virtual machine running on NTNU's servers. Spring Boot was wired up to start our REST application as a servlet and build a WAR file as output.

When a new version of application is ready for deployment, the WAR file is uploaded to the server and placed in our servlet container `tomcat`. When a new file is deployed to a container, the container automatically unpacks it to access the files and launch the application. An important aspect of the WAR file is that the filename is used to map HTTP requests to our application.

BACHELOR THESIS

4.3 MICROCONTROLLER

In this project we wanted to assemble a microcontroller prototype, which would act as a tracker unit. The long-term plan was to develop a light-weight microcontroller embedded in a pet collar. This section will describe how this microcontroller was assembled and programmed.

4.3.1 HARDWARE

The hardware requirements for the prototype was that it should be able to receive its own GPS position, and communicate its position to our REST-service.

4.3.1.1 Components

The requirements we had for the components for this project was they had to be light-weight, readily available, robust, and the components needed to be compatible with Arduino Uno Rev3. So that if we ever went to production, the components could be sourced from reliable suppliers at a predictable price range. That is why we landed on the following components for the prototype:

- Arduino Uno Rev 3 as our microcontroller, which facilitates our software and logic.
- Adafruit Ultimate GPS breakout, to receive the GPS position.
- Arduino GSM shield 2, as our means of communication.
- SIM card.
- 8000mA battery pack.

4.3.1.2 Wiring Diagram

Arduino Uno Rev 3	GSM Shield 2	GPS breakout
5V	VCC	VCC
GND	GND	GND
PIN 2	GSM RX	(standard Adafruit RX)
PIN 3	GSM TX	(standard Adafruit TX)
PIN7	Modem Reset	
PIN 8		GPS TX
PIN 9		GPS RX

Table 5: Wiring Diagram - Arduino, GSM and GPS

4.3.1.3 Battery consumption

We estimate the tracker prototype to use roughly 133mAh, see Table 6. This is due to the huge power consumption of the GSM module, which is based on 4 position loggings each hour and an estimated initialization process- and send time of 30 seconds. The power drainage is probably less than the estimate, but to be on the safe side we use this estimate to calculate battery lifetime.

Component	Max	Amount	Rate	Total
Arduino	50 mA	1	0.4	20 mAh
GPS Breaker	20 mA	1		20 mAh
GSM Shield	700 mA	4	0.033	93 mAh
Consumption				133 mAh

Table 6: Estimated Power Consumption

Using a battery life calculator (Digikey 2018), we estimate that the lifetime of the tracker device surmounts to approximately 42 hours, with the 8000mA battery package.

BACHELOR THESIS

The Arduino implementation can most certainly be optimized, and an approach to this problem will be discussed in section 5.1.1.

4.3.1.4 Prototype

Figure 25 shows how our Arduino prototype was assembled.

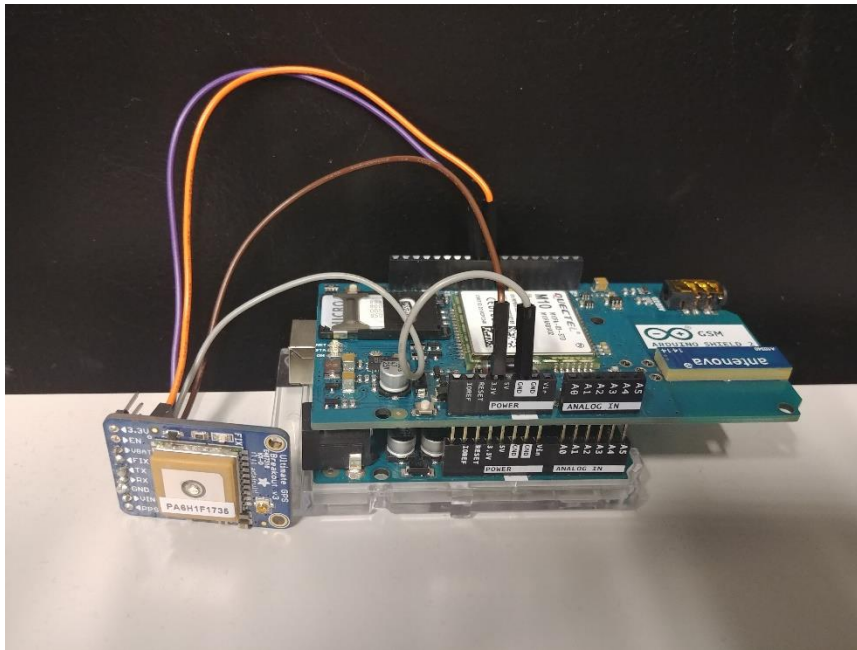


Figure 25: Track'a'Pet prototype of tracking device

4.3.2 MOBILE SUBSCRIPTION

To establish communication from our tracking device, we needed a suitable mobile subscription. After some research we found a subscription type called M2M, which is a machine-to-machine communication subscription. This subscription type is specialized for communication from IoT devices and was ideal for our use.

Since NTNU has a framework agreement with Telenor for such services, we could choose between two subscriptions delivered from Telenor. A comparison is shown in Table 7.

	Telenor Total (NOK)	Telenor Go Norge (NOK)
Establishment	15	150
Monthly price	20	299
Price pr. MB	6	0.10
Price pr. SMS	0.59	0.59

Table 7: Prices for Telenor M2M-subscriptions and additional services (Telenor u.d.)

To calculate the total cost of different subscriptions, we had to analyze the data usage for a connection from the microcontroller to the REST application. Using TShark as described in section 3.11.1, we analyzed several HTTP requests that consists of an initial three-way-handshake and a post of a new status to the server. We found that approximately 650 bytes of data was quite accurate for calculating one connection from the microcontroller. Together with the prices from Table 7, this is the base of calculation in Table 8.

	Frequency	# connections	MB	Telenor Total (NOK)	Telenor Go Norge (NOK)
1 Day	15 min	96	0.062	0.37	0.01
	5 min	288	0.187	1.12	0.02
	1 min	1440	0.936	5.62	0.09
1 Month	15 min	2880	1.872	31.23	302.12
	5 min	8640	5.616	53.70	304.37
	1 min	43200	28.080	188.48	317.85
1 Year	15 min	34560	22.464	374.78	3625.48
	5 min	103680	67.392	644.35	3652.44
	1 min	518400	336.960	2261.76	3814.18

Table 8: Data usage (650 bytes pr. connection) and the corresponding subscription cost

The calculations in Table 8 show that even with updates from the microcontroller every 1 min, Telenor Total would be the cheapest choice. However, with the uncertainty of packet size at the time of order we subscribed to Telenor Go Norge. With this solution we had flexibility for testing without any concern of data usage.

4.3.3 SOFTWARE

In the programming part of the Arduino, we used Adafruit alternative GPS, alternative Software Serial, Arduino GSM, as well as MemoryFree library. These are described in more detail in sections 3.7.2, 3.7.3, 3.7.1, and 3.7.4, respectively. Further on, we explain how our software implementation used those libraries to facilitate our desired result, as well we touch on some problems we encountered along the way.

4.3.3.1 JSON

At the starting point of this project, we hoped to use our own custom JSON parser to wrap our data to the REST service in a nice and memory efficient JSON string. Later in the process this parser did not give us the desired results, so we implemented ArduinoJson to parse our data. This later showed to be a bad idea.

As we developed the program for Arduino, the JSON parser was updated several times to accommodate the REST requirements. This refactoring led to so much extra work that we found it beneficial to implement the ArduinoJson library instead.

The ArduinoJson library originally worked as expected, and was simple to implement. However, a side effect of the library resulted in the microcontroller frequently restarting itself. We later discovered this was due to the ArduinoJson library using too much memory with our usage. Because of this, we had to implement our own light-weight JSON parser. This solved the memory leak problem, and our microcontroller could again communicate with our REST service.

4.3.3.2 GSM

To accommodate communication with our REST service, we implemented the built-in Arduino GSM library. This offered a way for our GSM component to communicate with the mobile phone network and use GPRS. By setting up a light-weight web client on the microcontroller, it enabled us to use the microcontroller to send data to our REST service. The GSM component uses the built-in antenna to first connect to the cellular provider using data from the SIM card. Once the connection is established, it can setup a light-weight web client to communicate over the internet. In our implementation, we created our own POST method to ensure memory usage would not exceed the microcontroller's limit, and comply with our requirements.

BACHELOR THESIS

4.3.3.3 GPS

Since the Arduino GSM library makes use of the integrated Software Serial library, we needed to find another way for the Arduino to communicate with the GPS module. This led to a series of conflicts where the alternative Adafruit GPS library did not contain a default return method for latitude and longitude. This was solved by making our own converter method, where we could pass degree-minutes as a parameter, to which we got decimal-degrees back.

4.3.3.4 Alternative Software Serial

Since we needed to both have the GSM and GPS module, we needed to implement a way for the GPS module to communicate with the Arduino. By default, the Adafruit GPS library uses the same software serial implementation the GSM shield relies on. By using an alternative Software Serial implementation in conjunction with the integrated Software Serial, we could specify which pins the alternative software serial implementation should use. This solved the conflict between the Arduino GSM shield and Adafruit Ultimate GPS breaker, allowing us to use both components simultaneously. This offered us the functionality we started out with as a requirement for the prototype.

4.3.3.5 MemoryFree

To help us debug and find potential memory leaks, we used a library called MemoryFree, which offered us an accessible way to read the memory usage on the microcontroller. This made it possible for us to debug and find issues before they became too complicated to deal with.

4.3.3.6 Software Flow

By following Figure 26, the microcontroller initiates the GPS module, then starts to receive GPS data. Once GPS data is received, the microcontroller connects to the GSM network and encrypts sensitive data before parsing the data into JSON, which it then sends to the REST service.

Thereafter, the microcontroller sends a new location once a timer has exceeded 15 minutes, bypassing the GPS initiation because the GPS maintains its fix on the satellites with low battery consumption.

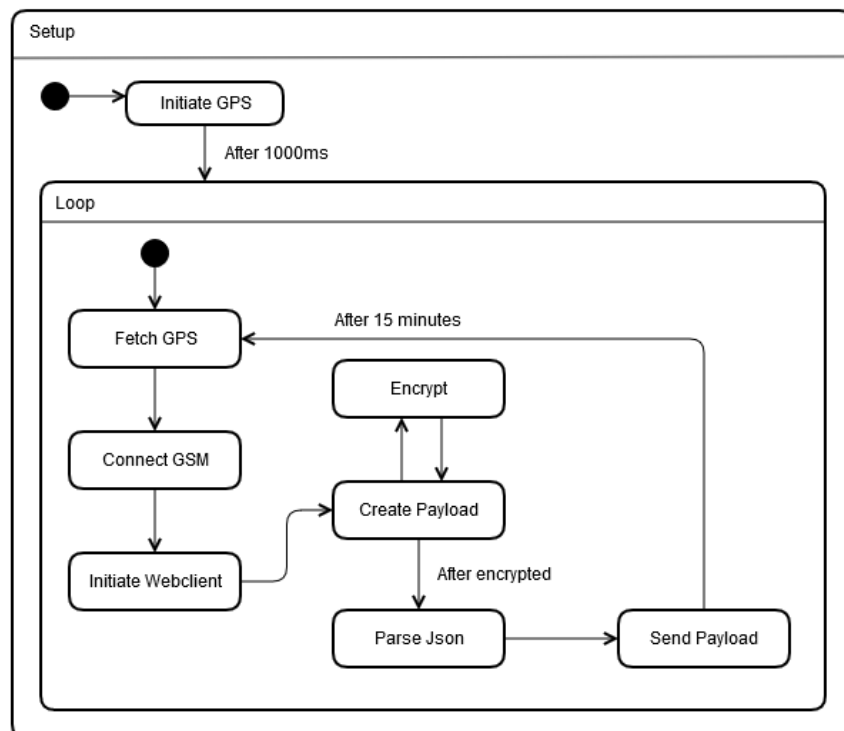


Figure 26: Microcontroller State Diagram

4.4 FRONTEND ARCHITECTURE

The frontend of the system is a native Android application. Because of our backend architecture, our system can handle any type of frontend client. As a proof-of-concept, we chose to develop an Android application, because we received feedback from potential users that they would prefer the simplicity and convenience of being able to check on their pets at any time, on their mobile device.

4.4.1 PROGRAMMING LANGUAGE

When deciding which programming language to use in the development the Android application, we had several options. We ultimately decided on Java, because of the amount of online support and resources available. Kotlin is relatively new language, with a smaller community, which means it may be harder to find solutions to common problems.

4.4.2 OVERALL STRUCTURE

We decided to structure the application using the Model-View-Controller pattern. The reasoning for this was increased flexibility, higher cohesion, and looser coupling. Table 9 Gives an overview of the files and classes that make up the Model-View-Controller structure in the application.

	File/class	Explanation
View	Layout	Files defining the layout of UI elements
	String resources	Files containing all the UI text
	Drawable	Figures and symbols
	Style	File containing format and look of layout
Controller	Activity	Represents a single screen in the application, controls user input
	Fragment	Represents a portion of a user interface, controls user input
Model	AsyncTask	Represents an operation that runs in the background, updates view
	HttpClient	Handles direct network communication
	Entity	Classes that represents domain objects

Table 9: Explanations of the components that make up the MVC

4.4.3 COMPONENTS

This section will describe the major components that make up the application.

4.4.3.1 Activity and fragment

An activity represents a single screen of an Android application. Activities interact directly with the user by handling input and updating the view (Android, Activity 2018). All the different screens in our application extend the Activity superclass, and have their own layout file that define how the UI is presented.

In our application, we also make use of a custom toolbar and navigation drawer. To avoid duplicating the code that controls these elements, we created a superclass that handles all the functionality pertaining to them. All the activities that want to use the navigation drawer and toolbar then only need to extend this class, called *BaseActivity*.

Using activities also allows our app to have more than one entry point. For example, the user can start the app normally and be directed to the main activity. However, if the user opens the app

BACHELOR THESIS

through a notification, they are directed to the map activity. This means the app adapts to the user, and not the other way around.

A fragment represents a more specific portion of the UI. Fragments are used for modular pieces of the application that can be reused several times. They have their own lifecycles, and handle their own user interaction (Android, Fragment 2018).

We used fragments in our application where we had a portion of a UI that we know we could reuse. For example, the list of pets in MainActivity is implemented as a fragment. If we were to adapt the app for tablets, we could show that fragment on one side, and the map on the other. Figure 27 demonstrates how this could look. This makes the code more modular, and allows for greater flexibility.

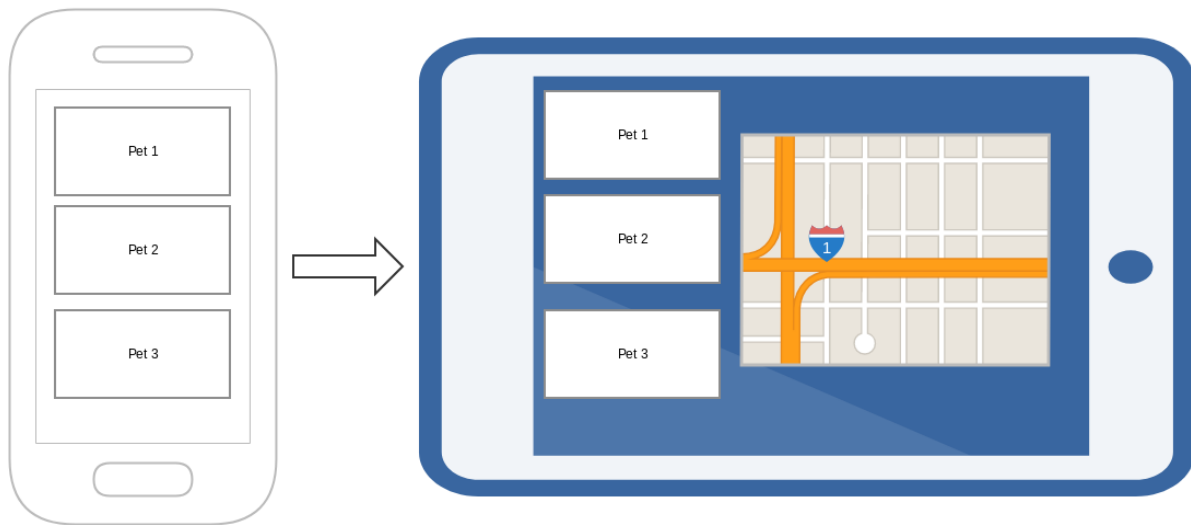


Figure 27: Usage of fragment

4.4.3.2 Async Task

AsyncTask is an Android-specific abstract class for performing background operations, and update the UI thread with the results (Android, AsyncTask 2018).

We make use of AsyncTasks when performing network operations, such as fetching data from the REST API. This has several advantages:

- Since tasks are executed asynchronously, the UI thread will not freeze while waiting for some network operation to finish.
- Tasks have only one responsibility, leading to higher cohesion.
- Tasks have no knowledge of activities or fragments, leading to looser coupling.

Code Snippet 23 demonstrates the usage of an AsyncTask:


```

public class LoginTask extends AsyncTask<String, Void, String> {
    OnPostExecute callback;

    public interface OnPostExecute {
        void onPostExecute(String token);
    }

    public LoginTask(OnPostExecute callback) {
        this.callback = callback;
    }

    @Override
    protected String doInBackground(String... strings) {
        String username = strings[0];
        String password = strings[1];
        return new LoginHttpClient().login(username, password);
    }

    @Override
    protected void onPostExecute(String token) {
        if (callback != null) {
            callback.onPostExecute(token);
        }
    }
}

```

Code Snippet 23: Implementation of an AsyncTask for user login

The three parameters, `<String, Void, String>`, represents the type of parameters sent to the task, the type of progress returned during execution, and the type the task returns when execution finishes, respectively. This class expects username and password as arguments, does not publish any progress, and returns an access token when it finishes executing. Here we have also defined an interface for `onPostExecute` to make it easier for the activity classes that override the method.

An example of how an `AsyncTask` is used is shown in Code Snippet 24.

```

private void login(final String username, String password) {
    new LoginTask(new LoginTask.OnPostExecute() {
        @Override
        public void onPostExecute(String token) {
            if (token != null) {
                JwtService.saveToken(getBaseContext(), token);
                navigateMainActivity();
            } else {
                showProgress(false);
                //Error handling omitted
            }
        }
    }).execute(username, password);
}

```

Code Snippet 24: Usage of LoginTask in the class LoginActivity

Here we instantiate a new `LoginTask` and pass an overridden `onPostExecute` as an argument. When the task finishes execution, it should return a new access token, which we then save to disk.

4.4.3.3 HTTP client

Our AsyncTasks do not handle HTTP communication directly. Instead, they call on a set of classes that are responsible for sending and retrieving data over the network. The application has a set of classes that handle their own specific HTTP event. For example, LoginHttpClient is responsible for sending login credentials to the correct URL, and return a token on successful login, and UserHttpClient is responsible for registering users, fetching users, updating, etc. Code Snippet 25 shows a typical method in an HttpClient class.

```
public class UserHttpClient extends HttpClient {

    public User getUser(String token) {
        String userJson = getJson(URL, token);
        return JsonParser.jsonToObject(userJson, User.class);
    }

    // Other methods and fields omitted
}
```

Code Snippet 25: Method `getUser` in the class `UserHttpClient`

Network communication often requires several steps. For example setting up a connection, opening a connection stream, exception handling, and more. For this reason, all HTTP classes extend from a single superclass that handles general HTTP events. This class, `HttpClient`, has general methods for posting JSON strings, fetching data, and more. This way, we avoid code duplication, and the classes that want to do network communication can work with the friendlier methods as seen above.

To convert the string of data returned from the network, we have a class that maps JSON to Java objects. This class contains static methods that are general enough to be used with any entity class. This class makes use of the Gson library, detailed in section 3.9.1.

4.5 USE CASE DIAGRAM

Figure 28 shows the most essential user interaction in the application.

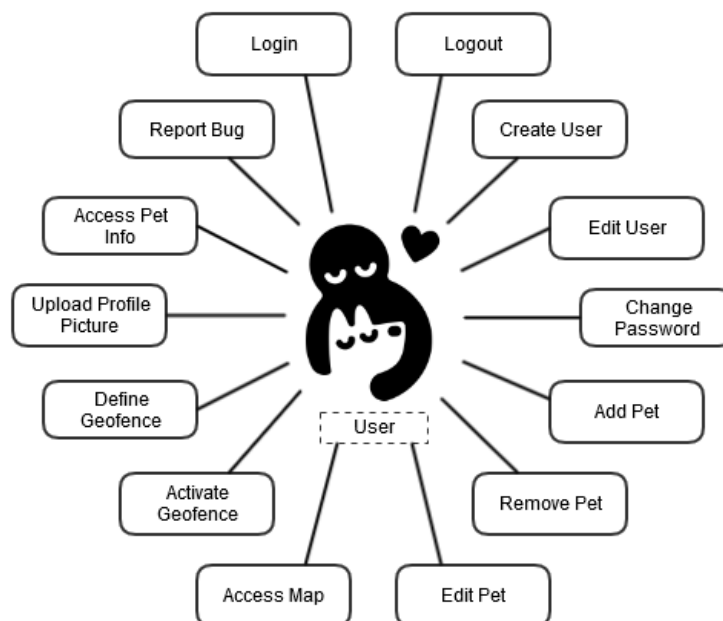


Figure 28: Use case diagram

4.6 CLASS DIAGRAM

To highlight the core implementation, we chose to create a couple of class diagrams to represent the dependencies in our project.

4.6.1 FRONTEND

Figure 29 shows a class diagram, focused around MainActivity. It highlights all the project specific dependencies, while all the android and java specific dependencies are omitted to increase readability. This demonstrates a general approach to dependencies, so we have chosen to omit class diagrams for the whole application.

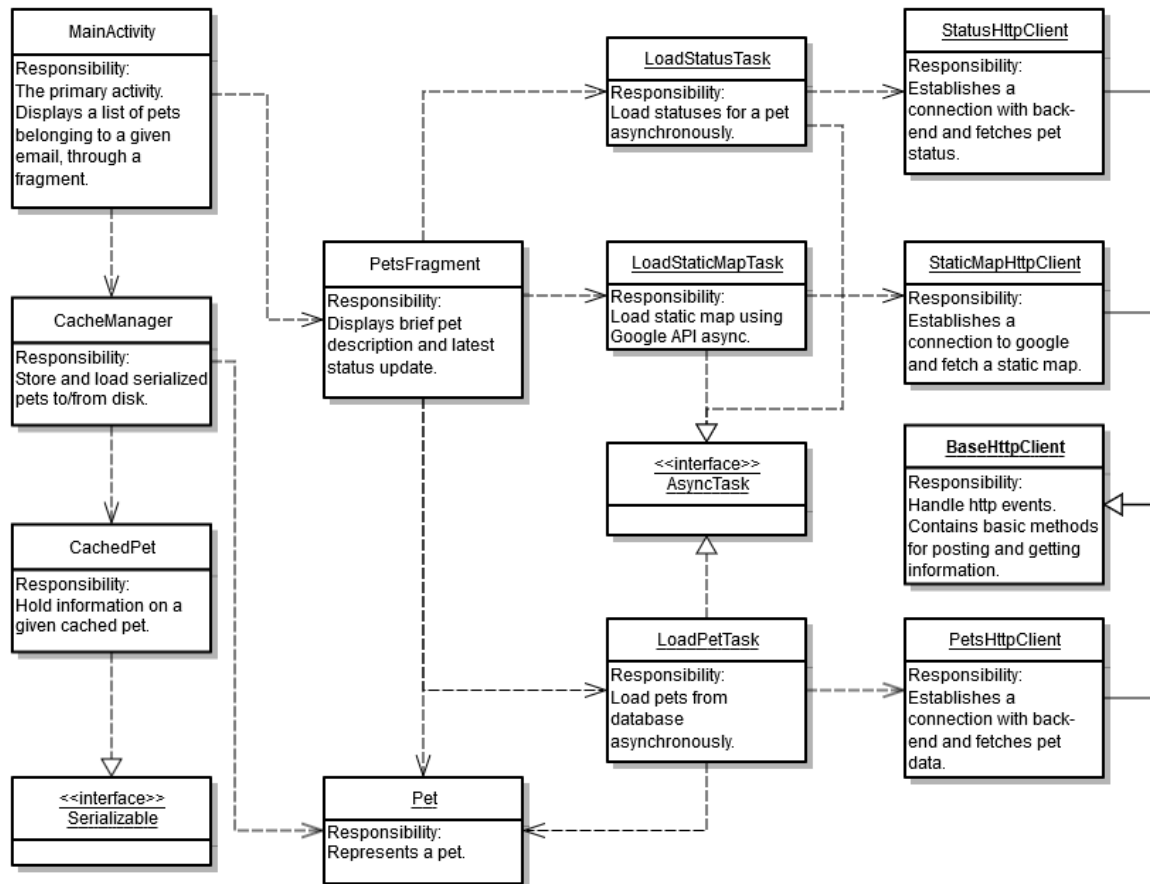


Figure 29: Class diagram around main activity

4.6.2 BACKEND

Figure 30 shows a simplified class diagram centered around UserController. It shows the general program flow and dependencies. We have chosen to omit the other controllers, services, repositories and entity classes, as the structure is generally the same for all of them. We have also chosen to omit all external dependencies and interfaces for improved readability. However, we have chosen to emphasize the flow of the security classes, because of their importance in the system.

The diagram also demonstrates loose coupling. Classes that are “low” in the hierarchy have no knowledge of the classes that call on them. For example, the entity class does not know of any other class in the system. UserRowMapper knows about the User class, but has no knowledge about the classes above itself. The diagram also shows how we have chosen to program against interfaces, and not implementations, which again results in looser coupling. A full class diagram is enclosed as an appendix.

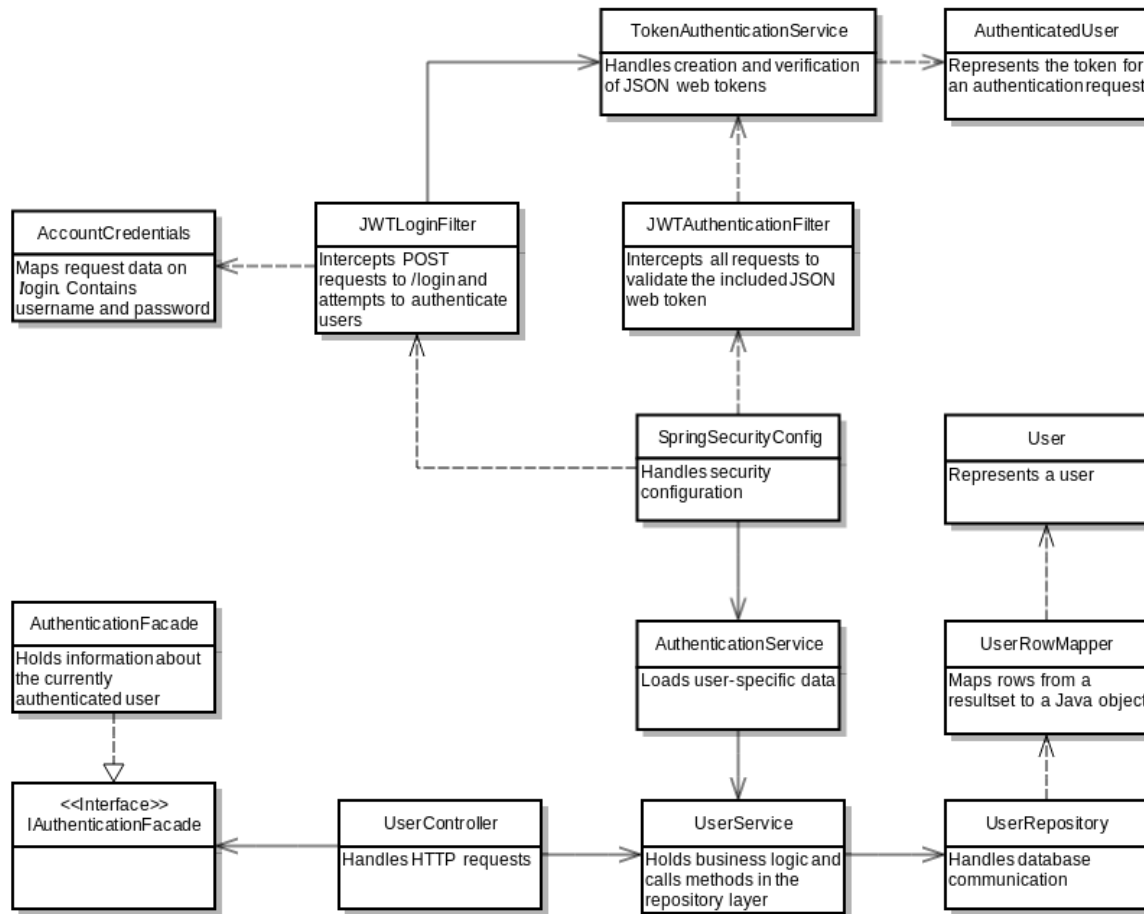


Figure 30: Simplified class diagram centered around UserController

4.7 DATABASE DESIGN

This section will describe how the database was designed, and which choices were made during the process. The Entity Relationship Diagram is shown in Figure 31.

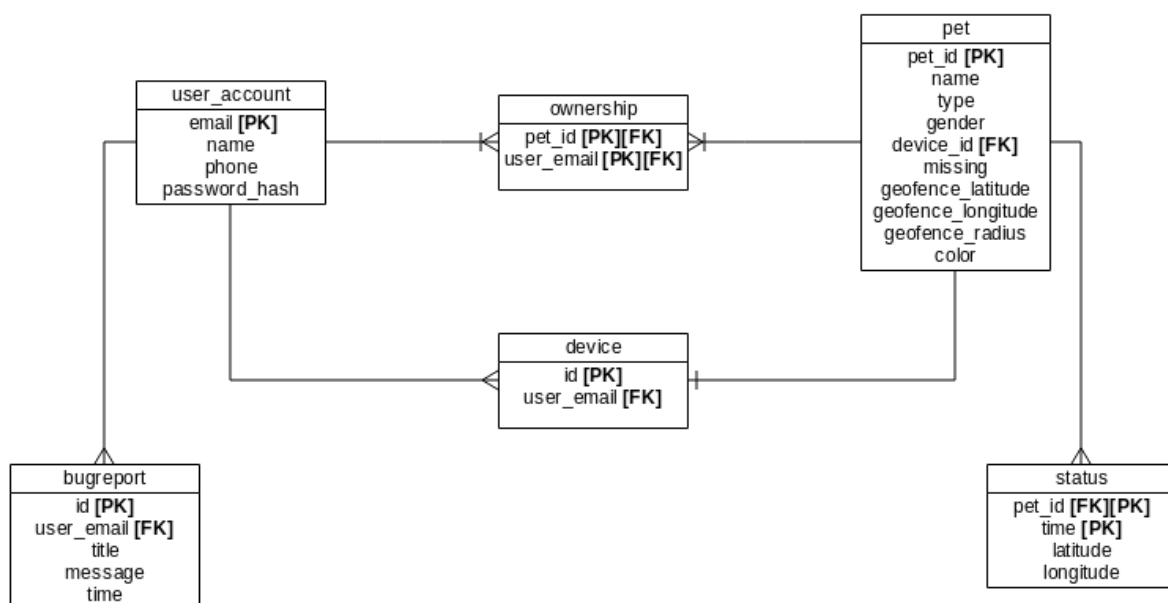


Figure 31: Entity Relationship Diagram for Track'a'Pet database

BACHELOR THESIS

The primary keys are marked with **[PK]**, and the foreign keys are marked with **[FK]**. Some tables, such as ownership, have composite keys. Composite keys are primary keys that consist of more than one attribute. In ownership and status, foreign keys are also used as primary keys. Not shown in the diagram is the database changelog table, which is generated and maintained by Liquibase.

By having a table for tracking devices, we gained some benefits. The table would contain all tracking devices in circulation. When a user registers a device with the app, the device is then locked to that user until it is deregistered. If someone tries to register an already registered device to themselves, they will be unable to until the original owner deregisters it. This means a device would be useless to a would-be thief.

The database design evolved over time. The bugreport table was added later, as we did not originally have a bug report feature planned. The user_account table originally had a *salt* attribute, but was later deleted since Bcrypt stores the salt with the hash.

4.7.1 ENTITY RELATIONSHIPS

- A user can own several pets
- A pet can be owned by several users
- A user can own several trackers (device)
- A pet can only wear one tracker at a time
- A pet can have several status updates associated with it
- A user can submit several bug reports

4.7.2 OVERVIEW OF ATTRIBUTES

This section details all the attributes in the database.

4.7.2.1 User_account

Attribute	Data type	Comment
Email	VARCHAR(100)	
Name	VARCHAR(100)	
Phone	VARCHAR(50)	Stored as string value since number operations will never be performed on phone numbers. Its length gives us better support for longer numbers, such as for other countries.
Password_hash	CHAR(60)	Hash is always 60 characters

Table 10: DB Table – User_account

BACHELOR THESIS

4.7.2.2 Pet

Attribute	Data type	Comment
Pet_id	Integer	
Name	VARCHAR(50)	
Type	VARCHAR(50)	For example 'Dachshund' or 'Egyptian Mau'
Gender	CHAR(1)	'M' or 'F'
Device_id	Integer	
Missing	Boolean	Defaults to false
Geofence_latitude	Double precision	
Geofence_longitude	Double precision	
Geofence_radius	Double precision	
Color	Integer	Google maps uses integers to represent colors

Table 11: DB Table - Pet

4.7.2.3 Status

Attribute	Data type	Comment
Pet_id	Integer	
Time	Timestamp	Timestamp is without time zone. Not including it simplifies some business logic. Can be added later to support more countries
Latitude	Double precision	
Longitude	Double precision	

Table 12: DB Table - Status

4.7.2.4 Ownership

Attribute	Data type	Comment
Pet_id	Integer	
User_email	VARCHAR(100)	

Table 13: DB Table - Ownership

4.7.2.5 Device

Attribute	Data type	Comment
Id	Integer	
User_email	VARCHAR(100)	

Table 14: DB Table - Device

4.7.2.6 Bugreport

Attribute	Data type	Comment
Id	Integer	
User_email	VARCHAR(100)	
Title	VARCHAR(100)	
Message	VARCHAR(500)	
Time	Timestamp	

Table 15: DB Table - Bugreport

4.8 SECURITY

This section will describe the measures we have taken to implement sufficient information security in the system. In the development of the system, we have emphasized that our solution will be as safe as possible.

4.8.1 PROTECTION FROM SQL INJECTION

Injection was the most critical security vulnerability in 2017 (OWASP 2017). To protect the system from malicious SQL injections detailed in section 2.2.5.2, we use prepared statements, as provided by JDBC Template. By structuring queries as shown in section 3.4.3, with question marks in place of variables, we can be sure that the user input is treated as a variable, and never as a valid SQL statement.

4.8.2 PROTECTION FROM BROKEN AUTHENTICATION

The second most critical security vulnerability in 2017 was broken authentication, such as functions related to authentication and session management (OWASP 2017).

4.8.2.1 Tokens

To authorize users, we decided to use self-contained access tokens for several reasons. Firstly, it would keep the backend stateless, since we would not have to store any information about the tokens. Secondly, we would not have to store passwords on the client devices. As explained in section 3.8.3, the JSON web token we are using only contains an identifier and expiration time.

The token itself is saved on the client device on a successful login. It is saved to the shared preferences, in private mode. This ensures that the token can only be accessed by the Track'a'Pet application. When the user logs out or uninstalls the app, the token is deleted.

The expiration time on the tokens were set to be ten days. When the token expires, the user must reauthenticate and receive a new token. This adds some additional security in case a user's token is somehow stolen, since they are only valid for a short time period.

4.8.2.2 End-point security

Almost all end-points on the API are secured. If a valid access token is not included in the header, the server will return a HTTP code 403 FORBIDDEN. Additionally, users will only be given access to the resources that matches the identifier in the token. This prevents users from manipulating the information belonging to other users. The exceptions are the login and register end-points, which everyone must be able to access without authentication.

We have also added a JSON web token login filter to the login end-point. This filter will try to authenticate the submitted user credentials. If they match a registered user, a token is generated and returned.

BACHELOR THESIS

4.8.3 PROTECTION FROM SENSITIVE DATA EXPOSURE

Sensitive data exposure was the third most critical vulnerability in 2017 (OWASP 2017). In Norway, the personal data act described in section 2.1.2 protects the users from violation of their right to privacy through the processing of personal data. When protecting our sensitive data, we had to take into consideration that GDPR will come into force in Norway by 2018 as described in section 2.1.3. Failure to follow the personal data act or the GDPR can result in considerable fines.

4.8.3.1 Data encryption

According to OWASP Top 10 (OWASP 2017), the most common reason for sensitive data exposure is simply not encrypting sensitive data.

As explained in section 2.2.3, by using HTTPS we ensure that all communication between users and the server is secure. No one other than the two parties involved can decipher the messages sent back and forth, thus also preventing man-in-the-middle attacks, detailed in section 2.2.5.1.

To enable HTTPS on our server, a certificate (described in 2.2.3.2) from a Certificate Authority was needed. *Let's Encrypt* is a free, automated and open CA (Internet Security Research Group u.d.). To get a certificate we had to demonstrate control over the domain. This had to be done by using a software that use the ACME protocol. This was done using Certbot client. Installation instructions can be seen in Figure 32.

```
sudo add-apt-repository ppa:certbot/certbot
sudo apt-get update
sudo apt-get install certbot
```

Figure 32: Install Certbot ACME client on Ubuntu

Certbot supports different plugins that can be used to fetch and deploy certificates. Running the command in Figure 33 will obtain a single certificate for trackapet.uials.no using the *webroot* plugin. To prove control of the domain, Certbot will place files below */var/lib/tomcat8/webapps/trackapet*.

```
certbot certonly --webroot -w /var/lib/tomcat8/webapps/trackapet -d trackapet.uials.no
```

Figure 33: Obtain a certificate from Let's Encrypt using Certbot

Certificates issued by Let's Encrypt is only valid for 90 days, therefore a scheduled task was set up to run twice a day to check if the certificate is due for renewal.

4.8.3.2 Password storage

One should be careful when working with user passwords, as explained in section 2.2.1. If unauthorized people manage to gain access to the passwords of other users, they can maliciously use that information on all accounts where a user might use the same password. For that reason, we salt and hash all user passwords before storing them in the database. We decided to go with BCrypt, because of its reliability, and because it automatically generates and stores unique salts for each individual password. Furthermore, BCrypt can be made resistant to brute-force attacks by increasing the iteration count of the internal hash function. This increases the time needed to hash a password, making it too slow to try common brute-force methods, but is still fast enough for a regular user who logs in normally.

4.8.4 MICROCONTROLLER

As mentioned in section 4.8.3.1 missing encryption of the data is the most common reason for sensitive data exposure. This shows how important it is to have encryption throughout the system. On the Android app this was solved by using HTTPS for all communication with the REST application. However, implementing HTTPS on the microcontroller proved to be as good as impossible with our setup, therefore we looked at alternative approaches using symmetric encryption as we described in

BACHELOR THESIS

section 2.2.3.1. While using symmetric encryption and with physical access to the device, it would be possible to reverse engineer the microcontroller and gain access to the shared key.

First, we tried AES encryption. this worked fine when testing the library by itself, but when using the AES library with our setup the system as a whole started to behave unstable. Sometimes it seemed to work and other times it crashed without any response. Debugging showed that the library was too resource-intensive.

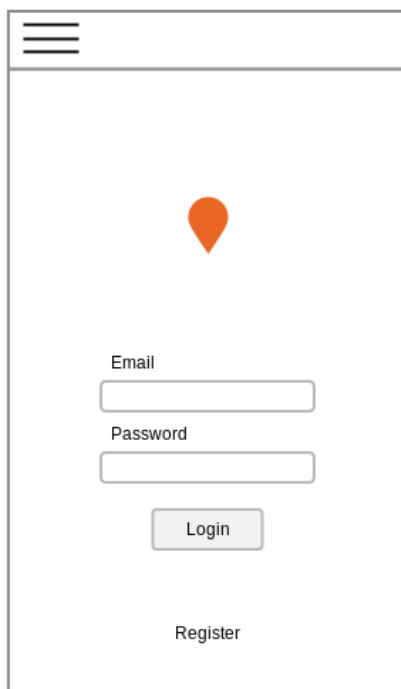
Next, we tried DES encryption, which was more stable with our setup. However, DES has some limitations. With specific knowledge and equipment, DES can in some cases be compromised. At the same time, it is the only stable solution possible on Arduino. Our implementation is generic enough to be easily replaced with any other symmetric encryption algorithm, or perhaps the final real device is powerful enough to run SSL. The information we send from the microcontroller is not considered sensitive by itself, so we concluded that stable encryption was more important than unstable, more secure encryption. The important thing to note is that we have shown that reasonable security is possible even on very resource-constrained devices.

4.9 DESIGN

In this section we will discuss the general design, with focus on the user interface.

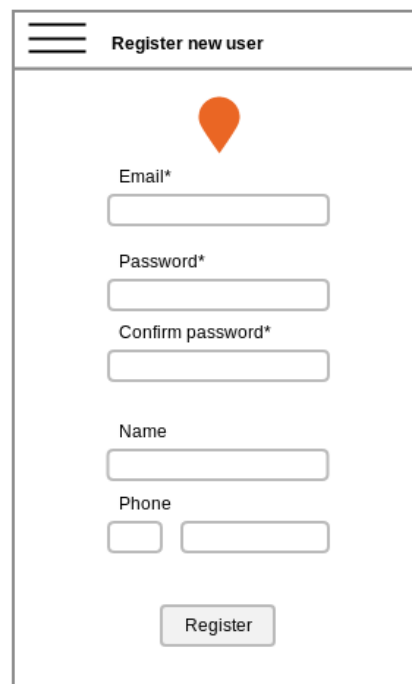
4.9.1 WIREFRAMES

Before we began writing code, we started planning how users would interact with the system. We began by drawing sketches on a whiteboard, and later we created more detailed wireframes. This saved us time long term, by having a general plan to stick to. A selection of wireframes that became the basis for our design is shown in Figure 34 through Figure 39.



The login wireframe shows a mobile application interface. At the top left, there is a hamburger menu icon. Below it, a large orange location pin icon is centered. Underneath the pin, there are two input fields: one labeled 'Email' and one labeled 'Password'. Below these fields is a 'Login' button. At the bottom of the screen, there is a 'Register' link.

Figure 34: Login wireframe



The register wireframe shows a mobile application interface. At the top left, there is a hamburger menu icon. To its right, the text 'Register new user' is displayed. Below this, a large orange location pin icon is centered. Underneath the pin, there are four input fields: 'Email*', 'Password*', 'Confirm password*', and 'Name'. Below these fields, there is a 'Phone' label followed by two input fields. At the bottom of the screen, there is a 'Register' button.

Figure 35: Register wireframe



Figure 36: Main activity wireframe

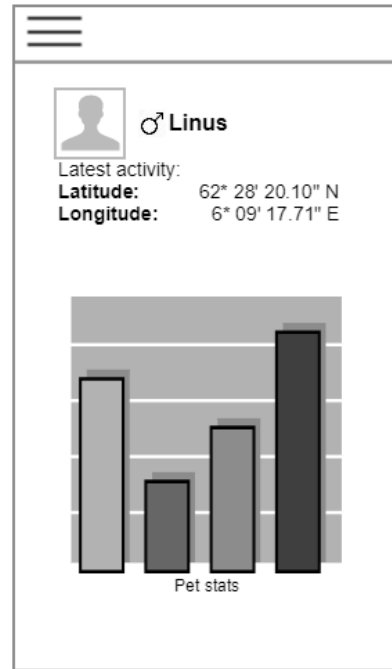


Figure 37: Pet profile wireframe

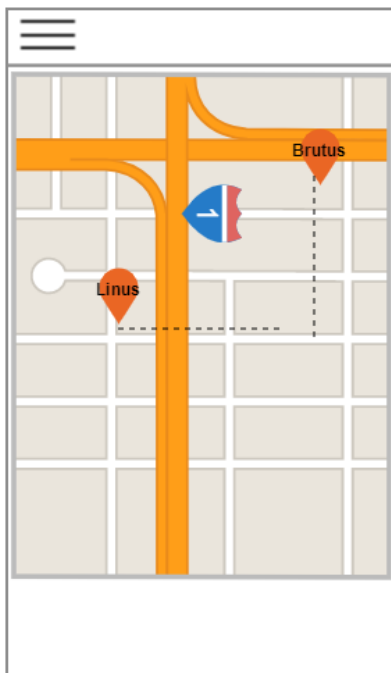


Figure 38: Maps activity wireframe

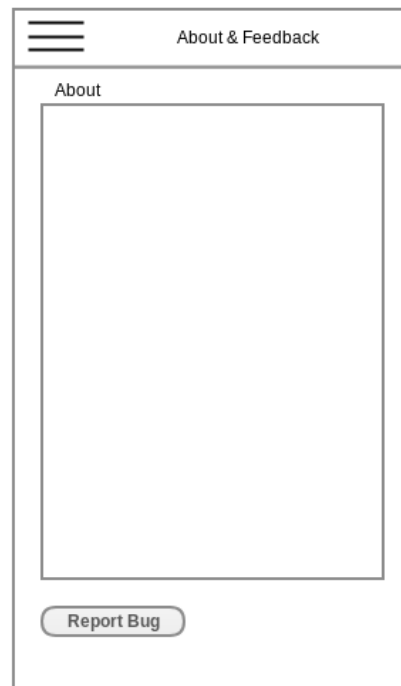


Figure 39: Feedback wireframe

The design was adjusted continuously while developing the application. If we found new or better ways to accomplish something, we would alter the design accordingly.

4.9.2 LAYOUT

4.9.2.1 Login and register

When a user starts the application for the first time, they are greeted with the login screen seen in Figure 40. If they do not have an account to log in with, they can click the text on the bottom to navigate to the register screen seen in Figure 41. After they successfully register, they are automatically logged in and redirected to the main activity.

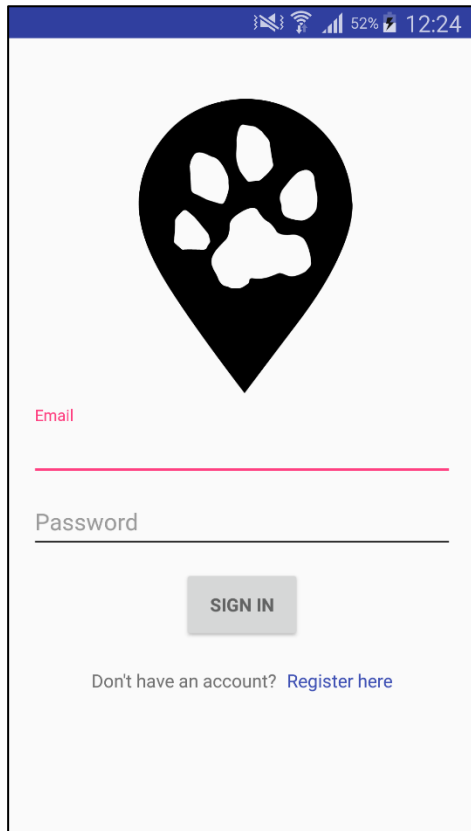


Figure 40: Login screen

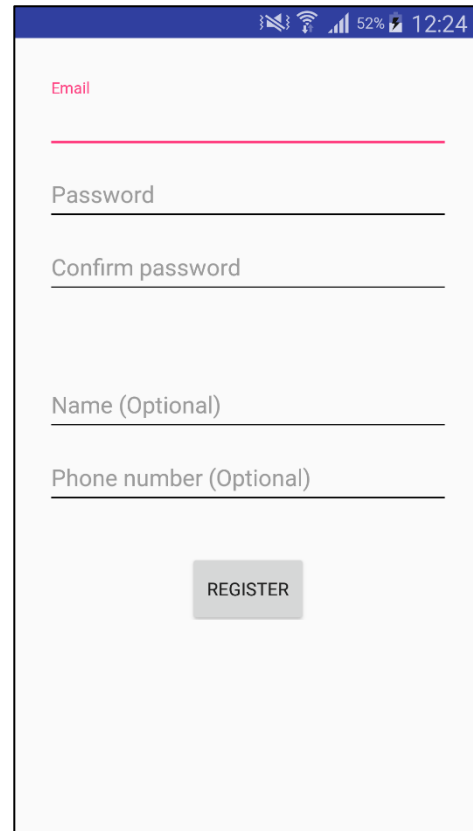


Figure 41: Register screen

The login and register screen have a multitude of error checks and communication with the user to ensure a smooth experience. For example, if the fields are empty when the user tries to sign up, messages appear telling the user exactly what is wrong. When the user tries to register with an email that already belongs to another account, an error message is displayed as well. Many other events are handled similarly. User feedback is explained in greater detail in section 4.9.3.

In Figure 41, we have utilized the gestalt principle proximity (2.7.1.1) by having a larger space between the obligatory and optional fields. This makes it easier for the user to immediately see which fields belong together.

The design is very similar to the wireframes, although we left out the toolbar and navigation drawer. The reason is that one should not be able to navigate to any other activity unless you are logged in.

4.9.2.2 Main activity

When users are logged in, they are directed to the main activity (Figure 42). It is worth noting that when the phone has a valid token saved, the user is automatically directed to the main activity. This way, the user does not have to log in every time they open the app.

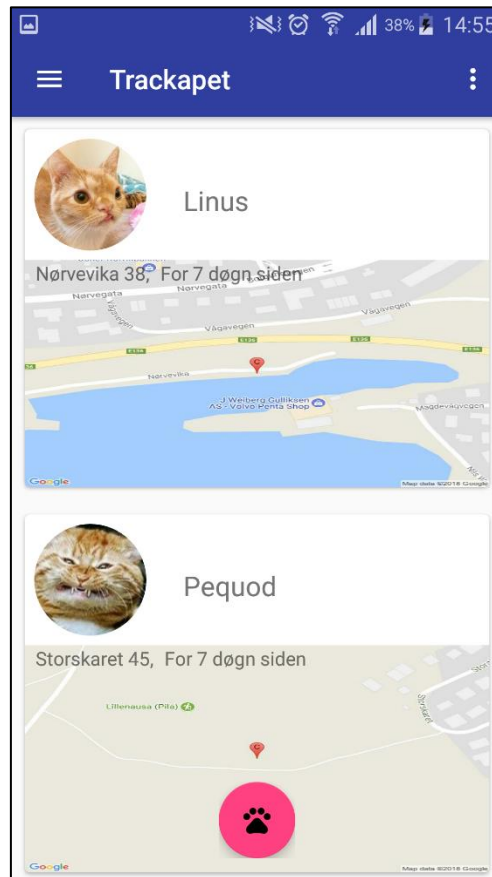


Figure 42: Main activity screen

The screen displays all the pets that the user is an owner of. Several people can own the same pets. An example is a family dog, which usually has at least two owners. Pet information is displayed as cards that consists of a profile image, name, static map image, last known position, and time of last update. The time of last update is displayed in a human-friendly format.

The gestalt principle figure/ground (2.7.1.3) is utilized by adding a slight drop shadow to each card to make them stand out from the background. This makes it clearer to the user what the focus on the screen should be. The principle of common fate (2.7.1.4) is also used, by having each card move as one unit when the user scrolls the screen up and down.

When at the top of the list, the user can drag the view down to update the list of pets, shown in Figure 43. Dragging down reveals an icon that begins spinning when let go, and the app then fetches the latest information from the server. If the user is somehow unable to perform this gesture, there is also a refresh button in the overflow menu on the far right on the toolbar.

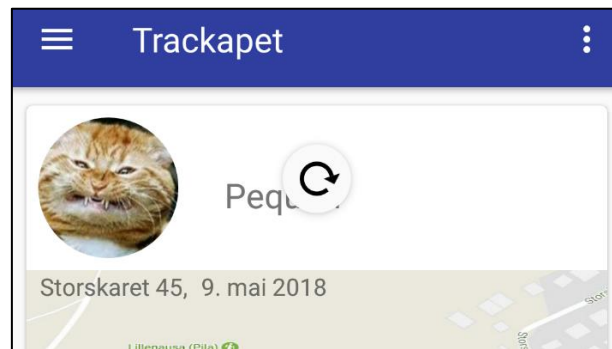


Figure 43: Refresh view by swiping down

The activity is designed to give users a quick overview of their pets, and easily letting them navigate to anything they would like more information on. Tapping on the upper part of the card directs the user to the pet profile screen. The map image is static, and only shows what the last registered position of the pet is. To navigate to the interactive tracking map, the user can tap on the map image. Tapping on the floating action button directs the user to the 'add new pet' screen.

4.9.2.3 Map activity

Once the user navigates to the map activity, they will be directed to a map specifically for that pet. This activity shows the latest position of the pet, as well as the last path the pet has traveled, indicated with a colored line. Figure 44 provides an example with blue lines. Furthermore, if the user has set up a geofence for the pet, it will be indicated on the map as a green circle with an eye as icon.

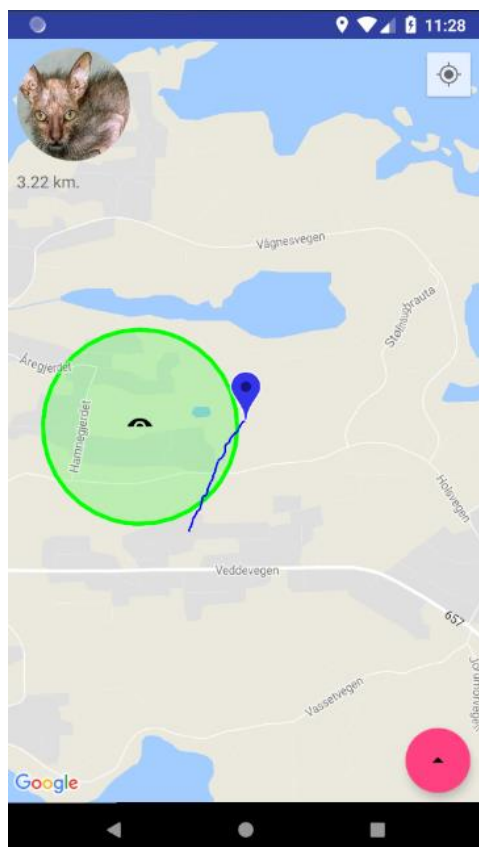


Figure 44: Actual map activity

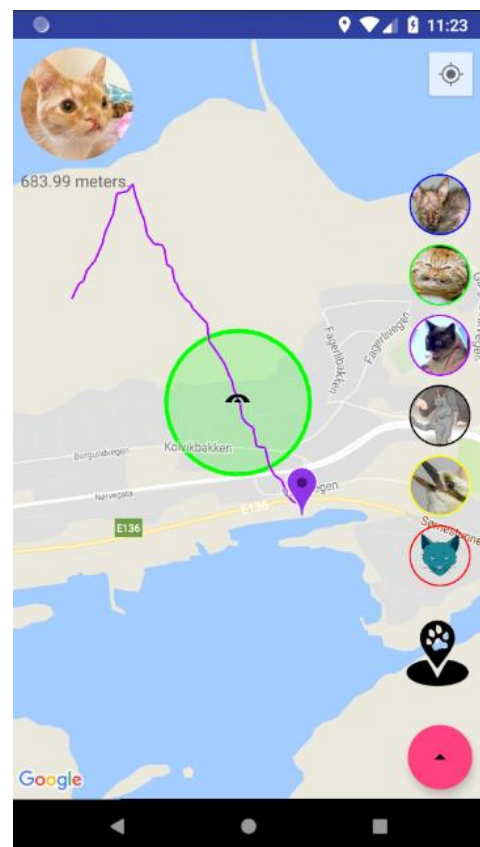


Figure 45: Map expanded options

The map also includes user location focus when the user presses the location icon in the upper right on the screen. The map activity periodically calculates the distance between the user and the current pet and displays it underneath the pet icon, on the upper left side of the screen. When the user

BACHELOR THESIS

presses the profile picture, the information of the pet is displayed besides the profile picture, as seen in Figure 45, in addition to the name of the latest position.

If the user needs the map to focus on the current pet, this is easily achieved with a press and hold gesture on the profile picture. This action animates the camera towards the pet and sets the marker in to focus. By using an animation on the camera panning, the user experiences a smooth transition from current position towards the pet. The gestalt principle proximity described in section 2.7.1.1 is utilized by grouping all current pet related features to the upper left of the screen. This lets the user use all the features associated with the current pet within a small section of the screen.

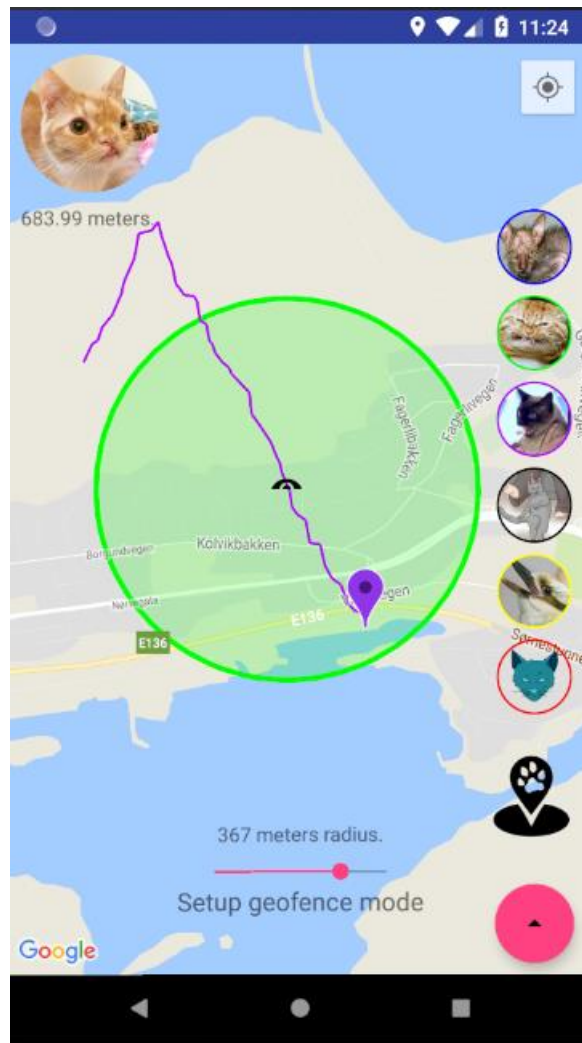


Figure 46: Defining geofence

Lastly, if the user presses the floating action button in the bottom right on the screen, the advanced options will be available. Here the user can activate markers for each of their pets in the same map.

The advanced options also give access to geofence options, seen as the icon above the floating action button in Figure 45. When this option is enabled, the user can define and manipulate a geofence for the pet. If the geofence is not defined, the user only clicks the center point of the geofence and uses the slider to define the radius of the geofence, as shown in Figure 46.

We use the proximity gestalt principle to achieve a sense of logical grouping, by association to buttons in advanced options in a to advanced options. In line with the gestalt principle similarity, the profile pictures have the same look, while the geofence has a distinct icon.

Table 16 contains a full list of events featured in the map activity.

Element	Press	Long Press
Profile Picture	Toggle name and location	Focus on pet location
User Location	Focus on user location	
Pet Marker(s)	Display name	
Geofence Marker	Display pet relationship	
Advanced Option	Toggle advanced options	
Geofence option	Toggle geofence placement and adjustment of geofence radius	
Non-primary pet icon	Toggle marker on map	Focus on pet location

Table 16: Full list of features in map activity

4.9.2.4 Pet profile activity

Once the user arrives at the pet profile activity, he or she will be presented with an overview of the current pet. On the top of the screen the user can see information on the pet in addition to the latest location in the form of a street address, with a timestamp, as seen in Figure 47.

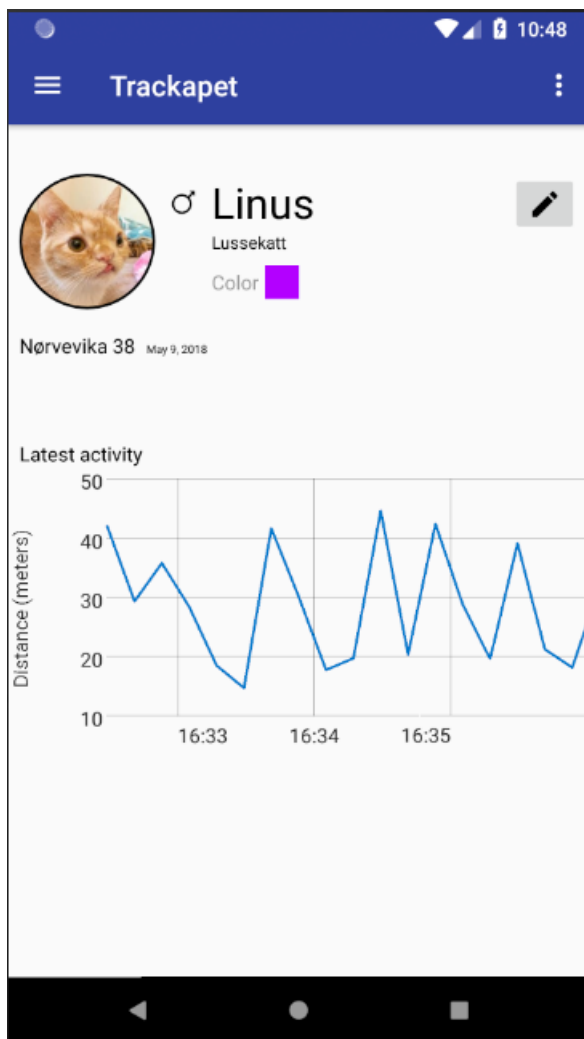


Figure 47: Pet profile activity



Figure 48: Pet profile edit

If the user wants to edit any of the information, such as name, color and profile picture, they must first enable editing by pressing the pencil icon. Thereafter, the elements which can be edited will be susceptible to click events.

If users want to take a new photo, they must first allow the application to use the camera, as shown in Figure 49. Since Android needs permission to use device-specific features, as described in section 2.8.1.1 and 2.8.1.2, this permission will be prompted the first time the user wants to use the camera. Once the permission is granted the application can use the camera to take a photo, or the user might want to select an existing photo from their device. See Figure 50 for an example. Both these options are in line with foreground/background gestalt principle by darkening the background and drawing the user's focus to the dialog box.

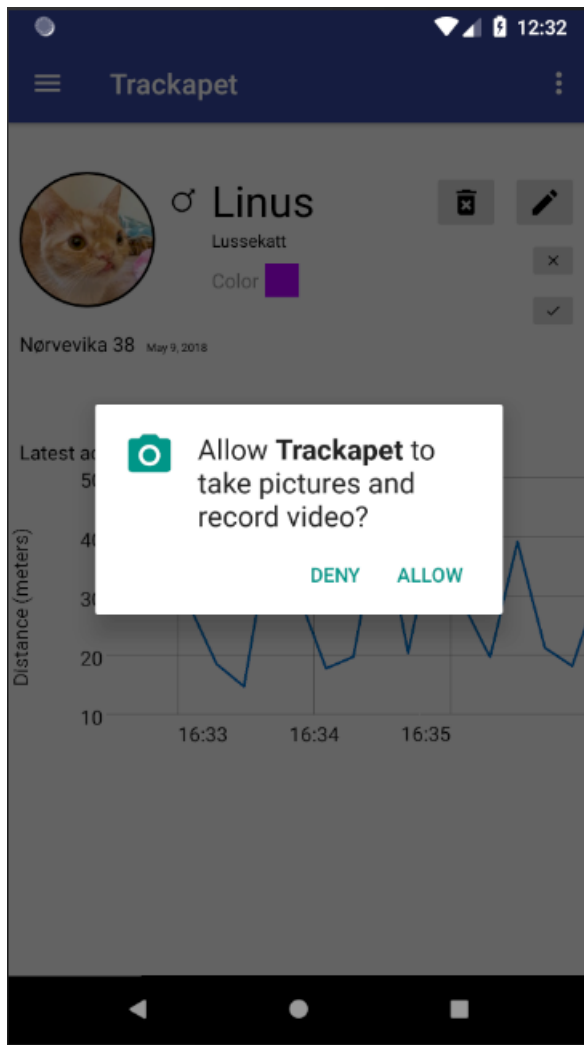


Figure 49: Permission to use camera

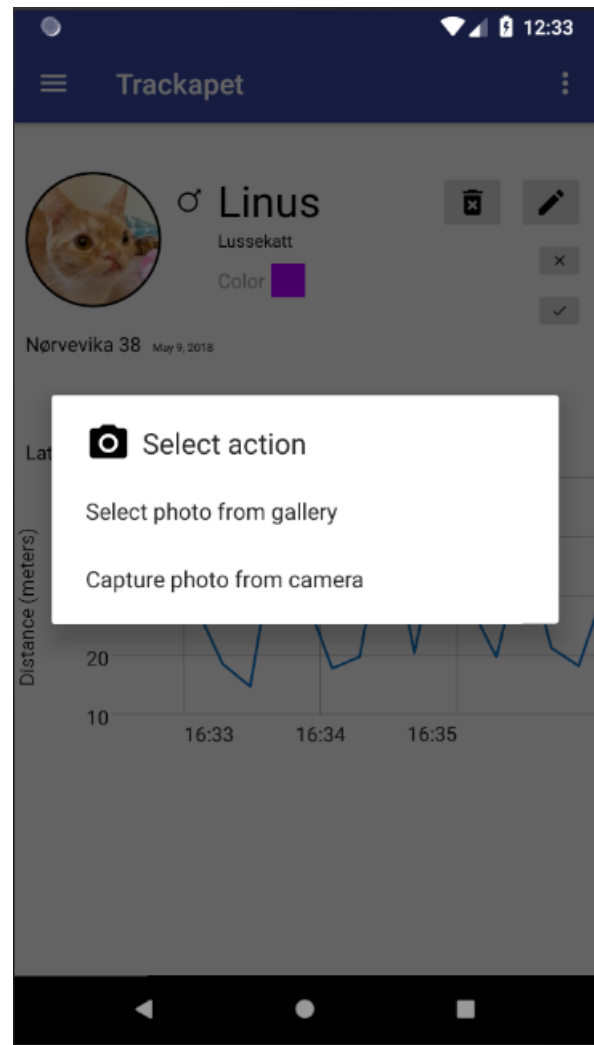


Figure 50: Select photo option

The pet profile activity also features a way to pick colors for the pet. Once the user clicks the color, a color pallet will appear in the foreground and prompt the user to choose a color, as illustrated in Figure 51. This color is later used in the map activity, 4.9.2.3, and displays the latest path of the pet. Additionally, it distinguishes the different pet markers on the map. To incorporate universal design, the colors are color blindness aware and selected from ColorBrewer, a color advice site for cartography (ColorBrewer 2018).

Lastly, the pet profile activity also displays a graph over the distance the pet has covered in the last 20 updates. The graph calculates the total distance between each status update, as seen in Figure 47.

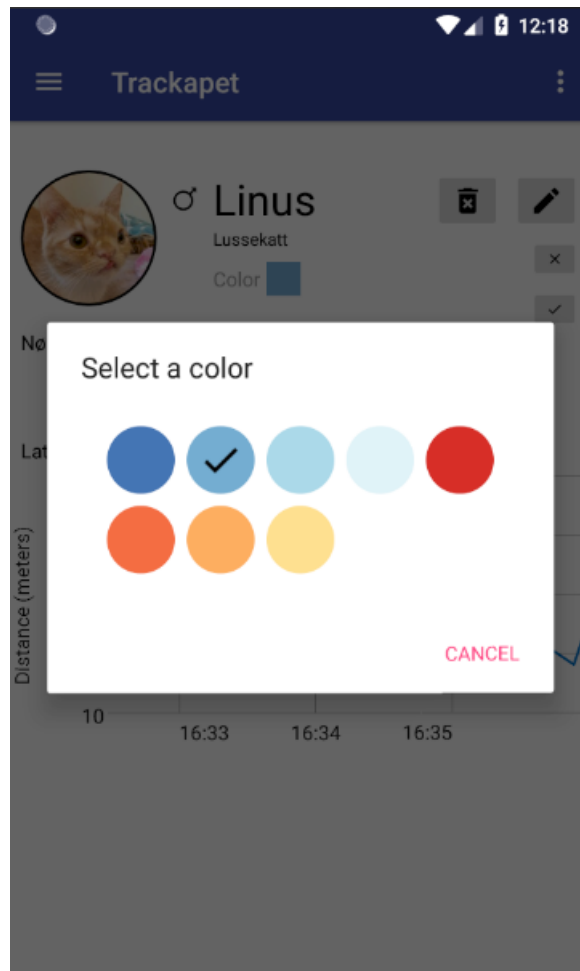


Figure 51: Color picker

4.9.2.5 Toolbar

Android toolbar gives the user a recognizable navigation bar at the top of the screen. This offers the user a persistent way to interact with the app, by giving them a short overview of what they are doing (see Figure 52).



Figure 52: Toolbar

4.9.2.6 Navigation drawer

The navigation drawer offers the user a way to navigate from anywhere in the application, see Figure 53. By using a persistent navigation drawer in the whole application, it offers the user a simpler way to navigate between activities, rather than chained navigation.

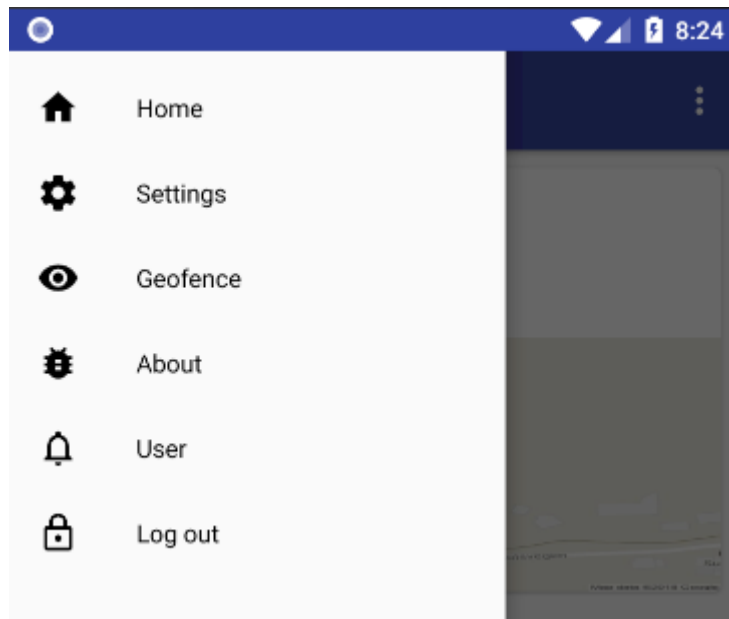


Figure 53: Navigation Drawer

To achieve this functionality, we have implemented a base activity class which contains the toolbar and navigation drawer, in addition to a fragment which can be manipulated. Activities that contain these features are all extended from this base activity, removing code duplication and giving us a codebase that is easier to maintain.

4.9.3 USER FEEDBACK AND ALERTS

It is important to give users concise feedback when they use the system. If an error occurs, they should not be left in the dark, even if the error is out of their control or not. Moreover, feedback should be consistent. For example, wrong information in input fields should be handled the same way for all fields.

An example of user feedback during data input is shown in Figure 54. When the user presses 'Sign in', an error message pops up giving a clear message on what is wrong. The field causing the error is marked with a red symbol, in line with the principles explained in section 2.7.3.

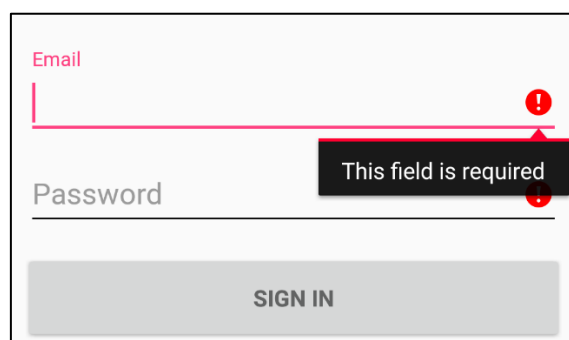


Figure 54: Error displayed when input fields are empty

BACHELOR THESIS

When informing the user of actions that require immediate attention, a dialog box is typically displayed. Usually, this means that an action of greater importance is about to be performed, or the system needs some action from the user before it can proceed. Figure 55 provides an example. The gestalt principle figure/ground (2.7.1.3) is used here. What was previously the user's focus becomes the background by darkening it, and displaying the dialog on top as the new figure. This lets users focus on this new event, while still providing them context through the background.

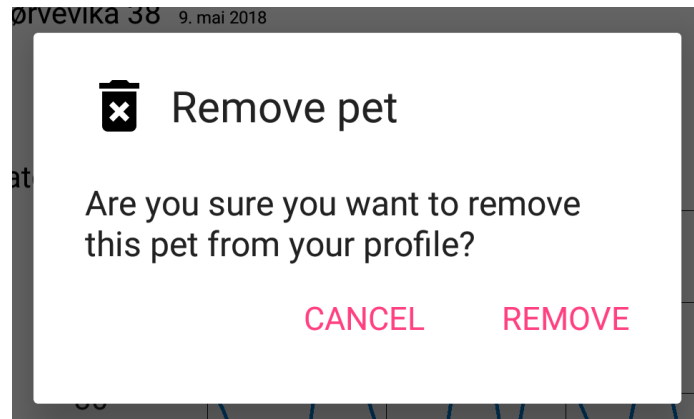


Figure 55: Alert on removal of pet

Often, non-important feedback is given to the user. This type of feedback does not require user intervention, and exists strictly to keep the user informed on what is happening. This increases the experienced responsiveness and enhances the user experience. These messages are usually displayed through a *toast*. A toast is a message that is shown briefly at the bottom of the screen. Figure 56 provides an example.



Figure 56: Toast saying a pet is successfully registered

Text fields have hints in them to let the user know what type of data is expected in them. When the user taps the field to begin typing, the hint becomes animated and moves up. This way, the user will always be reminded of what the expected data is, even if they are in the middle of typing. When text fields are in focus, they also become colored, so the user can easily see where they are supposed to be typing. Examples are shown in Figure 57 and Figure 58.



Figure 57: Unfocused input field



Figure 58: Focused input field

BACHELOR THESIS

4.9.4 DESIGN CONCERNS

4.9.4.1 Layout

To structure each activity, we used the Android Studio layout manager feature to drag and drop elements into our activity layouts. This featured worked great at building the UI elements, and to structure each layout.

4.9.4.2 Values

String

We have tried to keep our string values up-to-date as often as possible, and where it is logical we have translated the string values to Norwegian. By keeping a list of string values, instead of hardcoding in text in the application, porting the app to other languages becomes very easy.

Colors

To have persistent color throughout the whole application, we have defined all the secondary colors, which include red, green, blue, yellow, purple and orange, in addition to black and white. Furthermore, to reduce code duplication we only reference those colors in the code, so that the application does not depend on hidden hardcoded colors. We also have defined color-blind friendly colors to comply with universal design.

4.9.4.3 Drawable

When dealing with drawable, we preferred to use Android vector icons, which scaled nicely with our requirements. However, when handling profile pictures, we used image assets like Bitmap.

4.10 TESTING

This section will describe how we used testing in our project.

4.10.1 REGRESSION TESTING

Our main focus has been regression testing, to check if existing code has integrity after we implement new functionality. Most of these tests are concerned with computation and correct outputs, such as the distance calculation, time parsing and location. By testing these support methods, we can ensure that the application will keep integrity over time. Thus, reducing the opportunity for latent bugs later in development.

4.10.2 MANUAL TESTING

Other than regression testing we have used much time and effort on manual testing, because of the complexity of the system we focused on manual testing to test system-wide implications. We have also tried to test unexpected user inputs and navigation. By using this approach to test the application, we have uncovered many different scenarios which have helped us make the application more robust.

A typical approach we used was to try and break the application by overloading the UI and navigating to different parts of the application in an unnatural way. Lastly, we have handed the application over to users which have tried to break the application, to ensure integrity as well as highlighting improvements.

BACHELOR THESIS

4.11 BUSINESS ANGLE

In this section we will estimate the potential income and cost associated with the product. We do not account for indirect cost, and we make some assumptions surrounding cost and price.

4.11.1 MARKET

Initially the product would have been launched in Norway, and from Table 17 we can see that there are many potential pets in Norway. Note that this data is somewhat outdated, but it provides an idea of the potential market in Norway.

	Cat	Dog	Total potential pets
Pets	535'000	414'000	949'000
Pr. household	1.4	1.2	
Households	382'000	345'000	727'000

Table 17: Pet ownership Norway 2008, (Aftenposten 2008) validated with (Regjeringen.no 2004)

From Figure 59 we can see that Android has the largest market share internationally (roughly 80%), leaving our application highly compatible with the international market.

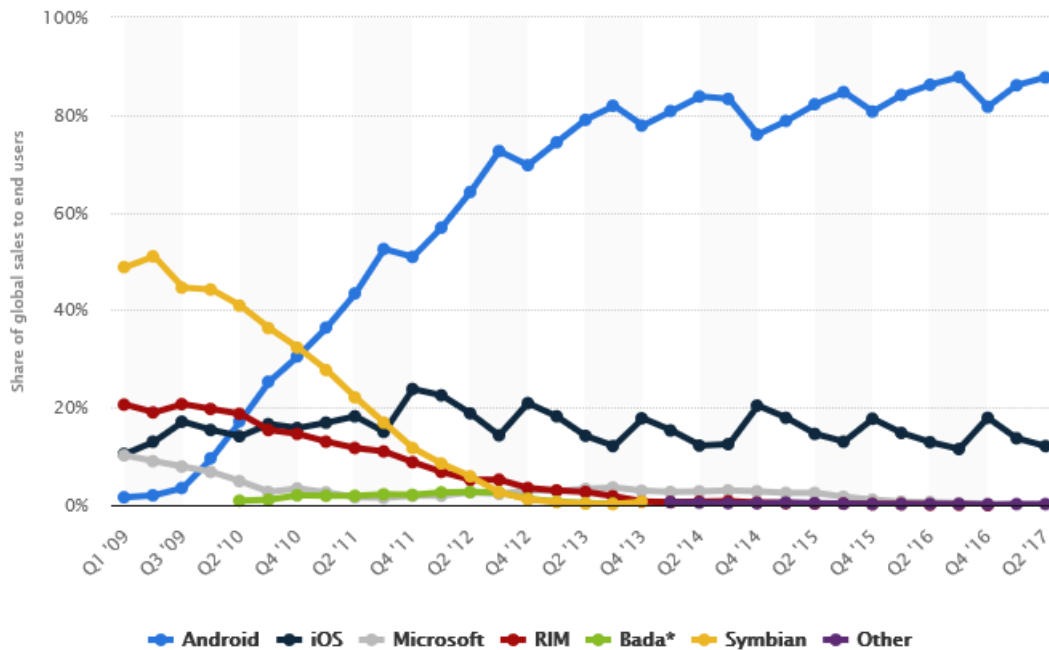


Figure 59: Smartphone market shares 2009-2017 (Statista.com 2016)

We intended to release the product to the Norwegian market, thus the Norwegian device market shares need to be accounted for. In the Norwegian market there is a different culture, where iOS devices are more abundant. Therefore, our application will be less sought after, leaving us roughly 50% compatibility towards Android.

BACHELOR THESIS

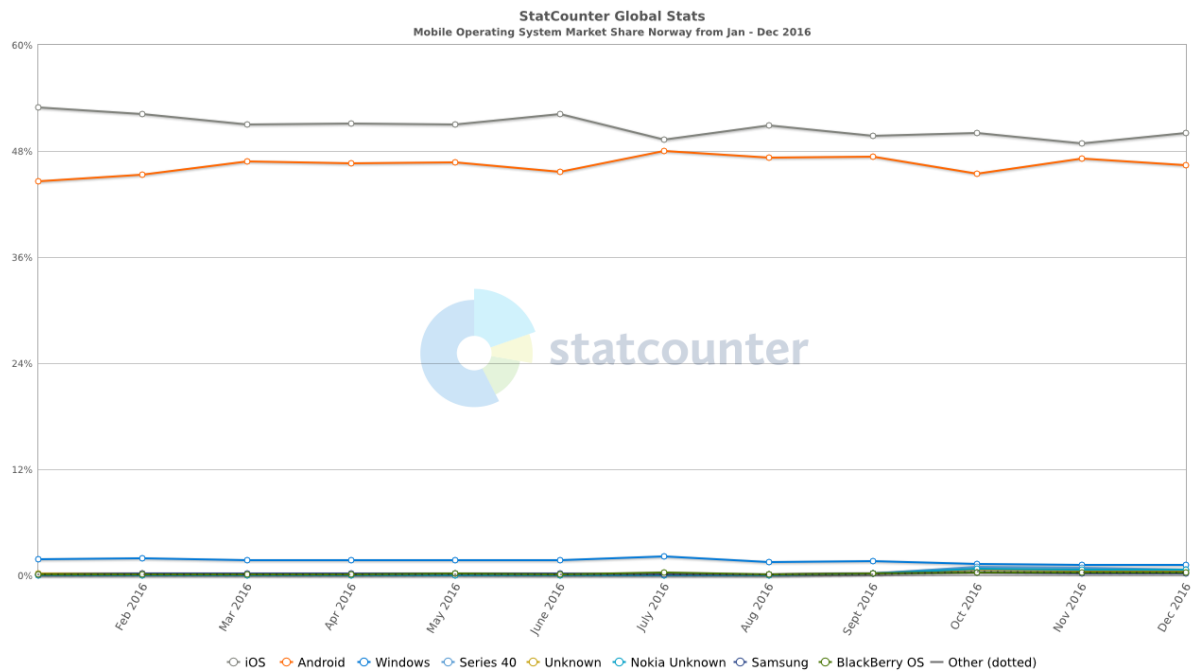


Figure 60: Smartphone market share Norway 2016 (gsStatcounter.com 2017)

If pet owners do not have any bias towards specific smartphone OS, this leaves us with approximately 475'000 potential pets and 363'000 households. We assume that we could potentially achieve to acquire roughly 0.1% - 2% of the estimated market within the first year of release. This results in 475 – 9'500 expected devices sold the first year.

4.11.2 HARDWARE COST

In this project the main cost has been to develop the backend and frontend solution, therefore most of the cost of development is already taken into account. Furthermore, the hardware components are a fixed cost, but as production scaled up we assume that we could produce cheaper components and arrange quantum discounts with suppliers. However, to be on the safe side we calculate with the initial cost of components, a pessimistic estimate and an optimistic estimate (Table 18).

Component	Full Cost (NOK)	Pessimistic Cost (90%) (NOK)	Optimistic Cost (50%) (NOK)
Arduino Rev 3	190	171	95
GPS Module	324	292	162
GSM Shield	618	556	309
Battery Solution	100	90	50
Miscellaneous	50	45	25
Total Cost	1282	1154	641

Table 18: Estimated cost

4.11.3 SOFTWARE COST

If this project were to be deployed, the team is confident that we would deploy the solution as a Heroku Cloud service (Heroku 2018). This would cost somewhere between 200 NOK to 4050 NOK, depending on the scale.

BACHELOR THESIS

4.11.4 INCOME

The team assumes the estimated market share, see 4.11.1, in Norway is achievable, thus setting the pet and household market to 475'000 and 363'000, respectively. In Table 19 we try to estimate device profit for the first year using the assumed price of 1'300 NOK per device.

% - Market Share	0.1% (475)		2% (9'500)	
	(NOK)		(NOK)	
Cost Modifier	Pessimistic	Optimistic	Pessimistic	Optimistic
Device Income	617'500	617'500	12'350'000	12'350'000
Device Cost	548'150	304'475	10'963'00	6'089'500
Device Profit	69'350	313'025	1'387'000	6'260'500

Table 19: Estimated Device Profit First Year

The running cost of the sim cards have huge impact on the profitability of the service, and to negotiate a reasonable price on that cost would benefit the business plan in the long run.

It is worth noting that Table 20 is based on full coverage of our estimated customer base from day one. This would likely not be the case, and a more reasonable approach would be that customer growth would be more linear. The first months probably would result in a deficit in this aspect.

% - Market Share	0.1% (363)		2% (7'260)	
	Pessimistic (NOK)	Optimistic (NOK)	Pessimistic (NOK)	Optimistic (NOK)
Subs. Income	30	30	30	30
M2M Cost	26	20	26	20
Service Profit	4	10	4	10
Agg. Service Profit	1'452	3'630	29'040	72'600
Heroku Cost	4'050	250	4'050	250
Monthly Profit	-2'598	3'380	24'990	72'350

Table 20: Continuous Monthly Profit

4.11.5 FINANCIAL FEASIBILITY

In comparison to competing products in the Norwegian market (see section 5.1.10), our pessimistic approach to hardware cost would place the product within the price range of competitors. By that estimate, it is possible to launch the product and break even. If we are more closely aligned with the optimistic cost, the product would produce quite a lot of revenue.

However, our main revenue stream would potentially be long-term subscriptions, given that we can negotiate better prices from cellular providers. Table 21 shows our estimate of the worst- and best-case situation one year after release. The conclusion being that this product is quite feasible to realize.

	Worst Case (NOK)	Best Case (NOK)
Profit	38'174	7'128'700

Table 21: First Year Profit Estimates

5 DISCUSSION

In this chapter, we will discuss the technical result of the project, and the execution of the project.

5.1 TECHNICAL RESULT

In this section, the technical result of the project will be discussed.

5.1.1 MICROCONTROLLER

Programming a microcontroller such as the Arduino proved to be a greater challenge than we first anticipated. The main reasons are the strict memory restrictions on the microcontroller, and the fact we had to use a low-level programming language we did not have much experience with.

We chose to work with the Arduino primarily because we had some experience with it beforehand, but we did not factor in the complexity of the tools we would have to use. However, if Track'a'Pet were to be realized as a full-fledged product, the Arduino prototype is closer to what a finished tracker would look like, compared to a higher-level board such as the Raspberry Pi.

We ended up using pointers and fixed size char arrays to handle strings, even though Arduino supports strings. The reason is that it is highly discouraged to use native strings in Arduino, since extra care must be taken to avoid memory leaks.

Following the advice from our supervisor, we did not focus on the prototype and as a result the GSM module is far from efficient. The GSM module implementation can certainly be optimized to expand the battery lifetime. For example, if we refactored the implementation to instantiate the GSM initiation to the setup, and only use the GSM module to post statuses when necessary, we would have less antenna drainage, resulting in longer battery lifetime. In addition, we also could have optimized the Arduino power drainage by slowing the clock frequency on the microcontroller, resulting in even less power drainage.

5.1.2 FRONTEND

5.1.2.1 Potential Web Frontend

We initially intended to implement a web-based administrative frontend, but chose to prioritize the Android user frontend. It is worth noting that since we have a truly independent REST API this could be done without much hassle, and it would function without depending on the Android frontend.

5.1.2.2 Android

Early in the process we discussed using Cordova and develop an application which could function on both Android and iOS using JavaScript. Cordova is an alternative approach to mobile device development, where the application is built in HTML, CSS and JavaScript. This results in a hybrid system, which is not a true native application nor a true Web-based application (Quasar 2018). We decided that this would introduce too much uncertainty in the project, because of lack of experience with JavaScript and similar solutions.

By using Reactive Extensions for JavaScript (Npmjs 2018) we could mimic the background tasks in Android, giving the application a smooth user interface.

In retrospect, Cordova could have solved some of the problems we have encountered along the way in Android Studio, but our feeling is that our progress and work in Android Studio have been of a high level. Additionally, we feel having native access to one type of device helped us in developing a more efficient application.

It is worth noting that the battery solution for the prototype is based on what was available to us at the mechatronic lab. This solution is far too heavy and bulky to ever be used on a cat. A more reasonable battery solution would be a 3.7V 2500mA Lithium Ion Polymer Battery (Adafruit 2018). By

BACHELOR THESIS

reducing the power supply, the importance to minimize the power consumption of the GSM module is even more apparent.

5.1.2.3 Geofence alternative approach

Android's memory allocation has meant that our geofence service is killed shortly after the application is closed. To solve this, we could have implemented a service on the REST backend that notified the application if a pet was outside the geofence. This could trigger the creation of a notification on the phone. The REST service would need to compare the location data as soon as it arrived in the database, and notify the user device if the pet was outside the allowed geofence.

To implement push notification, the application first needs to be registered at the OS push notification provider (OSPNS). Secondly, we would have needed to implement the Android specific push notification API on the REST service. Thereafter, we would have needed to store the app device IDs, which in turn would have mapped devices to the service. Using both the app-specific identifier and device identifier, we could have sent a push notification to the OSPNS, which in turn would have redirected the push notification toward the correct device (UrbanAirship.com 2018).

We originally tried to implement the Geofence client API (Android 2018), but it did not function for our use-case. It was dependent on the device position, and did not work with custom positions.

5.1.2.4 Architecture

Although we programmed the app in Java, Android has some specific intricacies that made it challenging to maintain good structure and code quality. For instance, activities and fragments should only contain code that handles user interface and operating system interaction. All other code should be moved to other classes. However, a lot of Android libraries and methods require the context of the activity that uses them. These methods sometimes belonged to classes that were so far removed from the activities, that using them meant sending the context as an argument several layers down.

Instead of creating our own HTTP client, it would have been more effective to use an existing solution, such as Retrofit (Retrofit u.d.). This could have taken care of everything related to network communication, which could have saved us a lot of time, and most likely would have resulted in a more stable client.

What we learned from this, is that we should have spent some time researching how to properly structure an Android app before we started writing code. We still did our best to write maintainable and modular code, but we can imagine that some parts can be more difficult to understand than others.

5.1.3 BACKEND

5.1.3.1 Framework

As discussed earlier, we decided to use the Spring Boot framework for the backend. The main reason for this choice was the minimal amount of configuration needed to get up and running. Spring also has good and easy support for security, among other things.

5.1.3.2 REST

We decided to implement a RESTful interface for several reasons. As we have mentioned earlier, RESTful services are much easier to scale, because of their stateless nature. We can effortlessly move our application to any device that can communicate with the REST interface. The backend does not have to know anything about the devices that it communicates with.

BACHELOR THESIS

5.1.3.3 Hosting and deployment

As mentioned earlier, we hosted our solution on a VM on NTNU's servers. With this solution we had more than enough resources available, and it was free of cost. For a full-scale release, we would rather go for a cloud solution like Heroku or Microsoft Azure. Doing so would give a scalable hosting solution and transfer the responsibility to the provider, and we could instead focus on further product development.

For deployment, we considered solutions for continuous delivery like Jenkins or Bitbucket Pipelines. Since our REST application will not have frequent updates, we decided to manually deploy the WAR file when a new version was ready for release.

5.1.4 DEVICE REGISTRATION

One of the technical aspects we did not completely finish, is the registration of tracker devices. Our goal was to have a database table of all devices in circulation, along with the user who owns it, as explained in section 4.7. The device ID would be a long string consisting of both numbers and letters. When the user buys a device, this ID would be visible on the device itself. Then the user could enter this in the app, and it would be locked to that user, until it is deregistered. Given a long enough ID, it would be almost impossible to brute-force it if there is no consistency in the way it is formed. This would prevent people from trying to register unowned devices to themselves.

Another solution to registering a device, would be to implement Bluetooth pairing. The user could press a button on the device to activate Bluetooth for a short period of time, and connect to it with their phone to permanently register it to themselves. This would prevent brute-force attacks altogether, but would increase the cost and complexity of the tracker devices.

5.1.5 SECURITY

This section will discuss the implemented security mechanisms.

5.1.5.1 Password hashing

Bcrypt was used for secure hashing of passwords. With alternative solutions, we would most likely have to salt passwords ourselves. This means we would need an extra column in the user table to store the salt, and we would need to extract it and append it to the password manually when doing comparisons. This has a larger margin for error because of the manual work needed. Bcrypt can be made more secure over time by increasing the number of times the internal hashing function is called. For the reasons mentioned, we believe Bcrypt to be the best solution for our needs.

5.1.5.2 Token

We chose to use JSON web tokens in our system. The benefits of JWT is that they are stateless and lightweight. We did not have to store any information about them on the server, and they were simple to work with. Changing the expiration on new tokens is simple if that is needed, and the expiration can be removed outright.

5.1.5.3 Microcontroller

The use of DES encryption on the microcontroller was far from optimal, but still better than sending the data as plaintext. 3DES, a technique to use DES encryption three times was also tested and considered. This approach would make it a little bit safer, but the time consumption would be tripled. We thought the small increase in security was not worth the extra resources.

Another solution to this problem was to buy an SSL chip that would take care of the encryption and give us satisfactory communication with HTTPS. This would be a much more robust solution, but unfortunately this was discovered so late in the process that we could not risk introducing new hardware into an already complex working solution. We decided to use the less secure DES encryption and suggest a proper encryption implementation as further development.

BACHELOR THESIS

5.1.6 DESIGN

We designed the app so that almost every interaction results in an immediate response. If the user needs to wait for something to happen, a loading icon is displayed to give the user feedback that the interaction is registered. The app also displays information to the user when the state changes, or events occur.

In the app, we used a color scheme that should be comfortable to look at. We deliberately avoided unfavorable color combinations, such as blue and yellow, or red and green.

We have tried to follow best practices for human-machine interaction. The gestalt principles are utilized to create a pleasant user experience, amongst other guidelines discussed in Johnson's 'Designing with the Mind in Mind' (J. Johnson 2014).

Although we have followed best practices, none of the team members are graphical designers. Knowing this, our priority was creating a system that was good on a technical level. If the application were to be released to the public, we would hire a professional to design the user interface for us.

5.1.7 TESTING

Testing on the microcontroller proved harder than expected, since we only used manual testing to test our Arduino programming. This approach to testing proved to be quite cumbersome and ineffective, and instead we should have researched and implemented a testing framework.

In the context of frontend testing we should have developed an initial testing plan. This could have led to more tests and better test coverage. In the end, this could have resulted in a recursive testing suite, that could have saved us time.

5.1.8 COMPLEXITY

Initially we felt that the system integration would predominate the complexity of the project, due to the different technological solutions. It has shown that this assessment was to a certain degree accurate. Along the way, we have also uncovered that implementing abstraction and best practices have proven quite complex, especially in the Android development phase.

Each individual piece is not incredibly complex in and of itself, but having it all work together in one coherent system is a different beast. The system consists of several different programming languages, frameworks, security considerations, storage engines, hardware components, interfaces for data exchange, and more. At the same time, we had to adhere to best practices and coding conventions, as well as develop the product with agile methods.

The prototype development proved to be much more complex than we initially thought. Since our supervisor warned us not to put too much effort into the hardware of the project, we managed to develop a proof of concept, rather than spending too much time developing a full-fledged production-ready prototype.

To have the opportunity to develop a full-stack system have proven beneficial for all team members, and we have acquired a great deal of experience during the project duration.

5.1.9 FUTURE DEVELOPMENT

There are several ways the product can be further developed and expanded upon in the future. One feature we wanted to implement was the ability to report pets as missing. The last position, pet information and owner contact information would show up on the map for nearby users, so that other pet owners could contact them if they saw their pets nearby.

Improved security on the microcontroller should be implemented, as described in section 5.1.5.3.

Another feature we considered was the possibility of Twitter integration. Each pet could have their own Twitter account, which automatically tweeted their location along with some context. For example, if the pet has been at home for the last few hours, it could automatically tweet a message

BACHELOR THESIS

along the lines of “Relaxing at home.” Alternatively, if the pet has been moving along a street, it could tweet “Having a blast walking down Karl Johan.”

Lastly, having Facebook integration with our system could remove the need for new users to register an account with us. This could make it more attractive for new users, since it would result in one less password to remember. At the same time, it would allow for pet status updates to automatically be posted on Facebook as well, and even information about missing pets.

5.1.10 EXISTING SOLUTIONS

There were not many solutions which covered pet tracking when we first discussed the problem domain. However, as time went on more and more tracker products hit the market, either in form of finished products or Kickstarter/pre-order products. The existing products will be discussed in comparison to Track’a’Pet.

5.1.10.1 Gibi Pet Locator

Gibi (Gibi 2018) is a GPS tracker collar using cellular providers to achieve maximum coverage. Gibi depends on an application and only operates within the United States of America. The estimated battery lifetime on the tracker is 2-3 days and costs \$99.99.

Compared to our system, Gibi offers some of the features we have developed, but currently only operates in America, which does not initially conflict with our potential market share. In addition, Track’a’Pet offers some extra features, such as pet profiles and native Norwegian translation.

5.1.10.2 The Paw Tracker

The Paw Tracker (The Paw Tracker 2018) is a light-weight tracking system which costs \$99.95, plus an additional \$10/month subscription fee. The paw tracker promotes a 10 days battery duration, with an update frequency of 12 hours.

The Paw only offers coverage in North America, which could be a selling point from our perspective. In addition to statistics over pet activity and a richer user experience.

5.1.10.3 Tractive

Tractive (Tracktive 2018) is a dog tracking system, which operates with international coverage. Due to their early mover advantage and coverage, the team views them as the biggest competitor. Their product costs \$49.99 (originally \$99.99), plus cellular provider fees.

Compared to Track’a’Pet, their cost alone is reason for concern. In addition, they have the most complete application in comparison to the other competitors. One approach to dealing with Tractive could be to highlight the security aspect of our system, in addition to promote nationally in Norway with native Norwegian translation.

5.1.10.4 Marcel Hundetracker

Marcel dog tracker (PetWorld 2018) costs 2199 NOK in addition to 99 a NOK/month subscription fee. It uses maps from Statens Kartverk and promotes a battery lifetime of 25 hours at an update frequency of 1 minute. Marcel dog tracker operates in Norway and is to be considered a close competitor.

Compared to Marcel, we can offer a substantial lower price and an application that puts the pet in focus.

5.1.10.5 Eyenimal

Eyenimal dog tracker (PetWorld 2018) is a light-weight tracker developed for hunting and hiking. It does not include an embedded sim card, and costs 2199 NOK. Battery life is estimated to roughly 24 hours. It features its own application. Eyenimal is to be viewed as a close competitor.

Compared to Eyenimal we can provide embedded sim card, and a lower cost. It is hard to provide a technical comparison because the application is proprietary.

5.2 PROJECT EXECUTION

During the whole project period the team has put in a lot of work and effort into making the system complete.

5.2.1 DEVELOPMENT METHODOLOGY

From the start we intended to use an agile approach to development, where we could adapt to changes as they presented themselves. By following the Scrum methodology, we were able to adapt our internal resources to the best of our effort. It also made the process more dynamic, where each member had the possibility to change their focus on a weekly basis. The team have tried to the best effort to follow the Scrum methodology with daily Scrum meetings, retrospectives, sprint planning and reorganizing the issue priorities along the way.

One drawback with using Scrum was our limited experience with estimating tasks. Estimating the time of software tasks is generally very hard, especially when no one on the team has had much professional software engineering experience. This resulted in some sprints where one or two tasks dominated by taking up much more time than anticipated. We got gradually better at adjusting our tasks and sprints as time went on, so it overall proved to be a valuable experience.

From our perspective, Scrum has been the right choice for the project and we had positive experiences using it throughout the project.

5.2.2 ORGANIZATION

Since the system consists of several main parts, the individual team members started working on separate pieces in parallel. That is, one member started working on the backend, another on the microcontroller, and the last on the server and hosting. This was beneficial, in that we could early on adapt our fundamental architecture to changes or limitations with other parts. However, it also meant that some parts mostly just had one person working on it, which made it harder for other team members to change the code later. With this in mind, we made sure that every team member contributed to the Android application early on. This allowed everyone to get familiar with the architecture and structure by getting hands-on experience with the development, so that it was easy to jump back in later to make changes.

5.2.3 VERSION CONTROL

From the start we set out to use full GIT flow, but as we time went on we only used the feature, master and develop branches. This resulted in a light-weight GIT flow. Consistent use of GIT flow made it easier to develop features, and avoid conflicts and partially implemented features. The team merged finished features into the develop branch, and only committed directly when implementing hot-fixes.

Additionally, the integrated VCS in Android Studio proved to be rather good and was used to solve many of the merge conflicts that emerged.

5.2.4 USE OF SOURCES

Throughout the project, we have focused on citing credible sources. Where possible, we have used text books from courses provided by NTNU, which are usually written by professionals who have established their credentials in the field. The fact that these textbooks have been chosen by lecturers can also establish some credibility, as it means other professionals agree to the quality of the texts.

When citing sources for technology used, we often ended up using the official documentation available online. They often contain more accurate information than tutorials on blogs or similar sites.

6 CONCLUSION

During the project, the group has acquired great experience and new insight into the development of full-stack applications. By working with a team-defined project, we have experienced freedom in how to design and implement decoupled systems.

The project resulted in a complete, usable product. Users can register themselves and their pets with the application and track them in real time. They can set up a geofence and will be notified if their pets leave it. Several pets can be tracked simultaneously and be distinguished by colored markers and paths. User and pet information can be modified at any time. Users can view statistics of their pets' movements, in form of a line graph.

The system implements best security practices. Passwords are salted and hashed before being stored. All traffic between the server and mobile devices are encrypted with HTTPS. The REST endpoints are properly secured in such a way that there is no risk of anyone other than the authenticated user gaining access to their information.

The microcontroller works as a proof-of-concept. It receives correct GPS information, and safely sends it to the remote server. It manages to perform a multitude of operations over long time periods, despite its limited memory and processing power.

Parts of the system are incomplete and need further development. Although pets can be owned by several people, there should be functionality for easily sharing pets with other users. Additionally, features regarding missing pets needs to be implemented, so that users can help other users find their pets. Facebook and Twitter functionality could also be implemented. This can be a good option for users that do not want to create a new account, and for sharing their pet's status and activity with their friends. Lastly, the microcontroller should implement HTTPS. This cannot be done natively, since it requires an external SSL chip.

In the project we put significant effort in writing good code. Even though this was time consuming, consistent refactoring resulted in better and more maintainable code. By following the principle of object-oriented programming and Clean Code, we managed to create a system that can easily be modified without much effort.

By depending on Spring Boot throughout the project, much of our backend services could be simplified and our focus diverted to implementing good solutions. Furthermore, by using Liquibase as version control for our database, it was easy to keep the project up-to-date across the team.

The project process has been challenging for all the team members, and we have grown significantly as full-stack developers. We have acquired valuable insight and experience in developing a product as a team, and have become more familiar with utilizing agile processes to develop software.

7 REFERENCES

- Adafruit. 2018. *3.7V 2500mA - Lithium Battery*. 1 January. Funnet May 28, 2018. <https://www.adafruit.com/product/328>.
- . 2018. «Adafruit.com.» *Ultimate GPS Breakout*. 1 January. Funnet April 13, 2018. <https://www.adafruit.com/product/746>.
- . 2018. «GPS Library.» *Adafruit.com*. 1 January. Funnet March 16, 2018. <https://learn.adafruit.com/adafruit-ultimate-gps-featherwing/arduino-library>.
- adafruit_support_rick. 2014. *Error w/ Adafruit GPS Library & GSM Library together*. 8163jb. 2 November. Funnet February 18, 2018. <https://forums.adafruit.com/viewtopic.php?f=25&t=38764&start=60>.
- Aftenposten. 2008. *Kjæledyr*. 25 May. Funnet May 24, 2018. <https://www.aftenposten.no/osloby/i/nwn55/Hvert-tredje-hjem-har-et-kjaledyr>.
- Android. 2018. *Activity*. 17 April. Funnet May 3, 2018. <https://developer.android.com/guide/components/activities/intro-activities>.
- . 2018. *Activity Life Cycle*. 17 April. Funnet March 3, 2018. <https://developer.android.com/guide/components/activities/activity-lifecycle>.
- . 2018. *AsyncTask*. 8 May. Funnet May 10, 2018. <https://developer.android.com/reference/android/os/AsyncTask>.
- . 2018. *Figure 1*. 8 May. Funnet May 10, 2018. <https://developer.android.com/guide/components/fragments>.
- . 2018. *Fragment*. 8 May. Funnet May 10, 2018. <https://developer.android.com/guide/components/fragments>.
- . 2018. *Geofence*. 5 May. Funnet May 30, 2018. <https://developer.android.com/training/location/geofencing#java>.
- . 2018. *Intent*. 27 April. Funnet May 10, 2018. <https://developer.android.com/guide/components/intents-filters>.
- . 2018. *Layout*. 27 April. Funnet May 14, 2018. <https://developer.android.com/guide/topics/ui/declaring-layout>.
- . 2018. *Permission*. 17 April. Funnet May 11, 2018. <https://developer.android.com/guide/topics/permissions/overview>.
- . 2018. *Request permission*. 25 April. Funnet May 11, 2018. <https://developer.android.com/training/permissions/requesting>.
- . 2018. *String resources*. 17 April. Funnet May 14, 2018. <https://developer.android.com/guide/topics/resources/string-resource>.
- . 2018. *Style*. 8 May. Funnet May 11, 2018. <https://developer.android.com/guide/topics/ui/look-and-feel/themes>.

BACHELOR THESIS

- Arduino. 2018. *Arduino*. 1 January. Funnet March 16, 2018. <https://www.arduino.cc/en/Guide/Introduction>.
- . 2018. «arduino.cc.» *arduino-uno-rev3*. 13 April. Funnet April 13, 2018. <https://store.arduino.cc/usa/arduino-uno-rev3>.
- . 2018. «Arduino.cc.» *Arduino GSM Shield 2*. 13 04. Funnet 04 13, 2018. <https://store.arduino.cc/arduino-gsm-shield-2-antenna-connector>.
- . 2018. *GSM*. 1 January. Funnet March 2, 2018. <https://www.arduino.cc/en/Reference/GSM>.
- . 2018. «Reference.» *Arduino.cc*. 1 January. Funnet March 16, 2018. <https://www.arduino.cc/reference/en/>.
- Atlassian. u.d. *Confluence*. Funnet March 23, 2018. <https://www.atlassian.com/software/confluence>.
- Auth0. u.d. *Introduction to JSON Web Tokens*. Funnet April 27, 2018. <https://jwt.io/introduction/>.
- Barnes, David J. 2012. *Objects first with Java: a practical introduction using BlueJ*. 5th. Boston: Pearson. Funnet March 1, 2018.
- Bitbucket. 2018. Funnet May 14, 2018. <https://bitbucket.org/product>.
- . 2018. *Gitflow*. Funnet May 14, 2018. <https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow>.
- Brain, Marshall. 2018. *Microcontroller*. 1 January. Funnet March 2, 2018. <https://electronics.howstuffworks.com/microcontroller1.htm>.
- Brainhub. u.d. «Scrum Process.» *brainhub*. Funnet May 3, 2018. <https://brainhub.eu/blog/wp-content/uploads/2018/04/differences-lean-agile-scrum-scrum-process.jpg>.
- Cohen, Eran. 2017. *35% of Users Have Weak Passwords; the Other 65% can be Cracked*. 13 March. Funnet March 1, 2018. <https://blog.preempt.com/weak-passwords>.
- ColorBrewer. 2018. *Color Selection*. Funnet May 26, 2018. <http://colorbrewer2.org/#type=diverging&scheme=RdYlBu&n=8>.
- Comodo. u.d. *What is HTTPS?* Funnet May 16, 2018. <https://www.instantssl.com/ssl-certificate-products/https.html>.
- Datatilsynet. 2017. *Personal Data Act*. 08 July. Funnet February 16, 2018. <https://www.datatilsynet.no/en/regulations-and-tools/regulations-and-decisions/norwegian-privacy-law/personal-data-act/>.
- DePriest, Dale. 2018. «NMEA data.» *gpsinformation.org*. 2 March. Funnet March 2, 2018. <http://www.gpsinformation.org/dale/nmea.htm>.
- . 2018. «NMEA data.» *Gpsinformation.org*. 1 January. Funnet March 16, 2018. <http://www.gpsinformation.org/dale/nmea.htm>.
- Difi. 2017. *Universal design of ICT*. 11 May. Funnet February 12, 2018. <https://uu.difi.no/om-oss/english>.

BACHELOR THESIS

- Digikey. 2018. *Battery Life Calculator*. 1 January. Funnet May 28, 2018. <https://www.digikey.no/en/resources/conversion-calculators/conversion-calculator-battery-life>.
- Dodenhof, Henning. 2018. *CircleImageView*. 2 April. Funnet April 28, 2018. <https://github.com/hdodenhof/CircleImageView>.
- Douglas, Barry. u.d. *Representational State Transfer*. Funnet April 20, 2018. https://www.service-architecture.com/articles/web-services/representational_state_transfer_rest.html.
- E. Whitman, Michael, og Herbert J. Mattord. 2014. *Principles of Information Security*. 5th. Boston: Cengage Learning. Funnet March 1, 2018.
- European Union. 2016. *Regulation (EU) 2016/679*. Official Journal of the European Union, European Union. <http://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=OJ:L:2016:119:FULL>.
- Fielding, Roy T, og Julian F Reschke. 2014. *RFC7230*. Internet standard, IETF. Funnet 04 20, 2018. <https://tools.ietf.org/html/rfc7230>.
- Fielding, Roy T, og Julian F Reschke. 2014. *RFC7231*. Internet standard, IETF. Funnet 04 20, 2018. <https://tools.ietf.org/html/rfc7231>.
- Fielding, Roy Thomas. 2000. *Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, Irvine: University of California. Funnet April 20, 2018. https://www.service-architecture.com/articles/web-services/representational_state_transfer_rest.html.
- Freeman, Eric, Elisabeth Robson, Bert Bates, og Kathy Sierra. 2010. *Head First: Design Patterns*. Cambridge: O'Reilly Media. Funnet May 3, 2018.
- Gehring, Jonas. 2018. *GraphView*. Funnet May 22, 2018. <http://www.android-graphview.org/>.
- Gibi. 2018. *Gibi Tracker*. 1 May. Funnet May 26, 2018. <https://getgibi.com/>.
- Google. u.d. *Features*. Funnet March 16, 2018. <https://developer.android.com/studio/features.html>.
- Google Trends. u.d. *Spring Boot, Java EE - Google search trends*. Funnet May 25, 2018. <https://trends.google.com/trends/explore?date=today%205-y&geo=NO&q=Spring%20Boot,Java%20EE>.
- Google. u.d. *User Guide*. Funnet March 16, 2018. <https://developer.android.com/studio/intro/index.html>.
- GoogleAPI. 2018. *LatLng*. 12 April. Funnet May 14, 2018. <https://developers.google.com/android/reference/com/google/android/gms/maps/model/LatLng>.
- Grenning, James W. 2002. «Planning Poker.» Paper. <https://wingman-sw.com/papers/PlanningPoker-v1.1.pdf>.
- Gson. 2017. *Gson user guide*. Funnet April 6, 2018. <https://github.com/google/gson/blob/master/UserGuide.md>.

BACHELOR THESIS

- gsStatcounter.com. 2017. *Smartphone Market Share Norway 2016*. Funnet May 24, 2018. <http://gs.statcounter.com/os-market-share/mobile/norway/2016>.
- Haerder, Theo, og Andreas Reuter. 1983. «Principles of transaction-oriented database recovery.» *ACM*. December. Funnet April 13, 2018. <https://dl.acm.org/citation.cfm?doid=289.291>.
- Heroku. 2018. *Heroku Cloud Service*. 1 January. Funnet May 30, 2018. <https://www.heroku.com/>.
- Igendel. 2014. «Arduino, programming.» *Idogendel*. 26 October. Funnet March 16, 2018. <http://www.idogendel.com/en/archives/19>.
- Internet Security Research Group. u.d. *Let's Encrypt*. Funnet May 7, 2018. <https://letsencrypt.org/>.
- Irvine, Calif. 2017. *Survey: Majority of Americans Reuse Passwords and Millennials Are the Biggest Culprits*. 19 July. Funnet March 1, 2018. <https://www.secureauth.com/company/newsroom/secureauth-survey-majority-reuse-passwords>.
- JetBrains. u.d. *Features*. Funnet March 16, 2018. <https://www.jetbrains.com/idea/features/>.
- . 2018. *IntelliJ IDEA*. Funnet March 16, 2018. <https://www.jetbrains.com/idea/>.
- Johnson, Jeff. 2014. *Designing with the mind in mind*. Waltham: Morgan Kaufmann.
- Johnson, Rod. 2005. *Introduction to the Spring Framework*. 1 May. Funnet April 13, 2018. <https://www.theserverside.com/news/1364527/Introduction-to-the-Spring-Framework>.
- Json.org. 2018. *Introduction to Json*. 1 January. Funnet May 30, 2018. <https://www.json.org/index.html>.
- JUnit. 2018. *JUnit*. Funnet May 21, 2018. <https://junit.org/junit4/>.
- Kimberlin, Michael. u.d. *Reducing Boilerplate Code with Project Lombok*. Funnet May 19, 2018. <http://jnb.ociweb.com/jnb/jnbJan2010.html#summary>.
- Kristoffersen, Bjørn. 2016. *Databasesystemer*. 4. utg. Oslo: Universitetsforlaget. Funnet March 1, 2018.
- Kurose, James F., og Keith W. Ross. 2013. *Computer networking: a top-down approach*. 6th ed., International ed. Harlow: Pearson Education. Funnet March 1, 2018.
- Liquibase. u.d. *Documentation*. Funnet April 6, 2018. <https://www.liquibase.org/documentation/index.html>.
- Lovdata. 2015. *Personal Data Act*. 01 October. Funnet February 16, 2018. <https://lovdata.no/dokument/NL/lov/2000-04-14-31>.
- . 2017. *Regulations for universal design of information and communication technology (ICT) solutions*. 22 May. Funnet February 16, 2018. <https://lovdata.no/dokument/SF/forskrift/2013-06-21-732>.
- Martin, Robert C. 2008. *Clean Code*. Prentice Hall.

BACHELOR THESIS

- Mediatek Labs. 2018. «Mediatek.com.» *MT3339*. 1 January. Funnet April 13, 2018. <https://labs.mediatek.com/en/chipset/MT3339>.
- Medium.com. 2016. *MVC*. 1 November. Funnet May 15, 2018. <https://medium.com/upday-devs/android-architecture-patterns-part-1-model-view-controller-3baecef5f2b6>.
- Microchip.com. 2018. «microchip.com.» *atmega328P*. 13 April. Funnet April 13, 2018. <http://www.microchip.com/wwwproducts/en/atmega328p>.
- Microsoft. 2018. 17 April. Funnet May 23, 2018. <https://support.microsoft.com/en-us/help/246071/description-of-symmetric-and-asymmetric-encryption>.
- Mockito. 2018. *Mockito*. Funnet May 21, 2018. <http://site.mockito.org/>.
- Mozilla. 2018. *developer.mozilla.org*. 29 January. Funnet April 13, 2018. <https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview>.
- MSDN. u.d. *Access Tokens*. Funnet April 27, 2018. <https://msdn.microsoft.com/en-us/library/Aa374909.aspx>.
- Muthu, Sudar. 2012. *MemoryFree*. 3 September. Funnet February 14, 2018. <https://github.com/sudar/MemoryFree/blob/master/README.md>.
- Niemeyer, Patrick, og Daniel Leuck. 2013. *Learning Java*. 4th. Sebastopol, California: O'Reilly Media. Funnet March 16, 2018.
- Npmjs. 2018. *Reactive Extension for JavaScript*. 22 May. Funnet May 28, 2018. <https://www.npmjs.com/package/rxjs>.
- Nuzzle. 2018. *Nuzzle Tracker*. 1 May. Funnet May 26, 2018. <https://hellonuzzle.com/>.
2016. *Oracle*. Funnet Mai 29, 2018. <https://docs.oracle.com/javaee/7/api/javax/servlet/Servlet.html>.
- Oracle. u.d. *Java SE Technologies - Database*. Funnet April 13, 2018. <http://www.oracle.com/technetwork/java/javase/jdbc/index.html>.
- OWASP. 2017. «OWASP Top 10 - 2017.»
- PassLib. u.d. *Bcrypt*. Funnet March 15, 2018. <https://passlib.readthedocs.io/en/stable/lib/passlib.hash.bcrypt.html>.
- PetWorld. 2018. *Eyenimal*. 1 January. Funnet May 28, 2018. <https://www.petworld.no/no/artiklar/gps-peiler-hund-for-mobiltelefon.html>.
- . 2018. *MarcelDogTracker*. 1 January. Funnet May 28, 2018. <https://www.petworld.no/no/artiklar/marcel-hundetracker-sort.html>.
- PJRC. 2018. *AltSoftSerial*. 1 January. Funnet February 26, 2018. https://www.pjrc.com/teensy/td_libs_AltSoftSerial.html.
- . 2017. *AltSoftSerial Repository*. 24 August. Funnet February 26, 2018. <https://github.com/PaulStoffregen/AltSoftSerial>.

BACHELOR THESIS

- u.d. *PostgreSQL*. Funnet April 13, 2018. <https://www.postgresql.org/about/>.
- Programcreek. 2013. April. Funnet Mai 29, 2018. <https://www.programcreek.com/wp-content/uploads/2013/04/web-server-servlet-container.jpg>.
- Project Lombok. u.d. *Lombok features*. Funnet May 22, 2018. <https://projectlombok.org/features/all>.
- u.d. *Putty*. Funnet May 4, 2018. <https://www.putty.org/>.
- QT. 2009. «qt.io.» *AT Commands*. 1 January. Funnet April 13, 2018. <https://doc.qt.io/archives/extended4.4/atcommands.html>.
- Quasar. 2018. *Cordova*. 21 February. Funnet May 16, 2018. <https://quasar-framework.org/guide/cordova-introduction.html>.
- Regjeringen.no. 2004. *Hundeloven*. Funnet January 24, 2018. <https://www.regjeringen.no/no/dokumenter/otprp-nr-48-2002-2003-/id172933/sec3>.
- Retrofit. u.d. *Retrofit: A type-safe HTTP client for Android and Java*. Funnet May 29, 2018. <http://square.github.io/retrofit/>.
- Rick, Adafruit Support. 2014. «Adafruit Forum.» *Adafruit*. 12 November. Funnet March 16, 2018. <https://forums.adafruit.com/viewtopic.php?f=31&t=63113&hilit=+gps+gsm+altsoftserial>.
- RosettaCode. 2018. *Haversine*. 9 May. Funnet May 14, 2018. https://rosettacode.org/wiki/Haversine_formula.
- Rouse, Margret. 2005. «www.TechTarget.com.» *CamelCase*. 01 September. Funnet April 27, 2018. <https://searchmicroservices.techtarget.com/definition/CamelCase>.
- Schwaber, Ken, og Jeff Sutherland. 2017. *The Scrum Guide™*. November. Funnet March 23, 2018. <https://www.scrumguides.org/scrum-guide.html>.
2012. *Servlet lifecycle*. October. Funnet Mai 29, 2018. https://www.ntu.edu.sg/home/ehchua/programming/java/images/Servlet_LifeCycle.png.
- Shmeltzer, Shay. 2017. *Managing Your Database Source Code*. 16 October. Funnet April 6, 2018. <https://blogs.oracle.com/shay/introduction-to-liquibase-and-managing-your-database-source-code>.
- Silberschatz, Abraham. 2014. *Operating system concepts: international student version*. 9th edition. Hoboken, N.J.: Wiley. Funnet March 1, 2018.
- Sommerville, Ian. 2016. *Software engineering*. 10th ed. (global ed.). Boston: Pearson. Funnet March 1, 2018.
- Spaniakos. 2015. *ArduinoDES*. February. Funnet Mai 29, 2018. <http://spaniakos.github.io/ArduinoDES/>.
- Spring. u.d. *Class JdbcTemplate*. Funnet April 13, 2018. <https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/jdbc/core/JdbcTemplate.html>.

BACHELOR THESIS

- Statista.com. 2016. *MarketShare_Smartphone*. Funnet May 24, 2018. <https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/>.
- TechTerms.com. 2018. *Vector Graphic*. 1 January. Funnet May 11, 2018. <https://techterms.com/definition/vectorgraphic>.
- Telenor. u.d. *M2M*. Funnet May 19, 2018. <https://www.telenor.no/bedrift/iot/m2m/>.
- The Paw Tracker. 2018. *The Paw Tracker*. 1 May. Funnet May 26, 2018. <https://www.thepawtracker.com/>.
- Tracktive. 2018. *Tractive GPS Tracker*. 1 May. Funnet May 26, 2018. https://tractive.com/noeu_en.
- Tutorialsteacher. u.d. *Inversion of Control*. Funnet April 27, 2018. <http://www.tutorialsteacher.com/ioc/inversion-of-control>.
- UrbanAirship.com. 2018. *Push Notification Explained*. 1 January. Funnet May 1, 2018. <https://www.urbanairship.com/push-notifications-explained>.
- US Environmental Protection Agency. 2017. «Latitude/Longitude Data Standard.» *United States Environmental Protection Agency*. 7 June. Funnet March 2, 2018. <https://www.epa.gov/geospatial/latitudelongitude-data-standard>.
- Waldby, James. 2017. «StackExchange - Arduino.» *icsp-pin-what-is-it-actually*. 30 June. Funnet April 13, 2018. <https://arduino.stackexchange.com/questions/40098/icsp-pin-what-is-it-actually>.
- Walters, Nathan. 2016. *Spectrum*. 23 December. Funnet May 16, 2018. <https://github.com/the-blue-alliance/spectrum>.
- Webb, Philip, Dave Syer, Josh Long, Stephane Nicoll, Rob Winch, Andy Wilkinson, Marcel Overdijk, et al. 2018. *Spring Boot Reference Guide*. Funnet April 12, 2018. <https://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/#getting-started-introducing-spring-boot>.
- Wireshark. u.d. *TShark*. Funnet April 27, 2018. https://www.wireshark.org/docs/wsug_html/#AppToolstshark.
- . u.d. *Wireshark*. Funnet April 27, 2018. <https://www.wireshark.org/>.
- Wiring. 2018. «Wiring.» *Wiring.org.co*. 1 January. Funnet March 16, 2018. <http://wiring.org.co/>.

TABLE OF FIGURES

Figure 1: Password hashing process	3
Figure 2: Typical flow of token exchange	4
Figure 3: Symmetric encryption	5
Figure 4: Public-key cryptography	5
Figure 5: HTTPS connection to trackapet.uials.no	6
Figure 6: HTTP request (Mozilla 2018)	7
Figure 7: HTTP response (Mozilla 2018)	8
Figure 8 – One type of NMEA sentence - GGA: Fix data (DePriest, NMEA data 2018)	9
Figure 9: Agile development	10
Figure 10: Overview of MVC interactions	12
Figure 11: Gestalt principle - Proximity	13
Figure 12: Gestalt principle – Similarity	13
Figure 13: Gestalt principle – Figure/ground	13
Figure 14: Scrum process (Brainhub u.d.)	16
Figure 15: How web server and servlet container process requests (Programcreek 2013)	20
Figure 16: Servlet lifecycle (Servlet lifecycle 2012)	20
Figure 17: Example of Gitflow in our project	28
Figure 18: Start a TShark capture	29
Figure 19: Screenshot of Confluence dashboard	31
Figure 20: Jira – Backlog	31
Figure 21: Jira – Active sprint	32
Figure 22: General system architecture	33
Figure 23: Backend architecture	34
Figure 24: Popularity of search terms ‘Spring Boot’ and ‘Java EE’ represented by blue and red line, respectively (Google Trends u.d.)	35
Figure 25: Track’a’Pet prototype of tracking device	39
Figure 26: Microcontroller State Diagram	41
Figure 27: Usage of fragment	43
Figure 28: Use case diagram	45
Figure 29: Class diagram around main activity	46
Figure 30: Simplified class diagram centered around UserController	47
Figure 31: Entity Relationship Diagram for Track’a’Pet database	47
Figure 32: Install Certbot ACME client on Ubuntu	51
Figure 33: Obtain a certificate from Let’s Encrypt using Certbot	51
Figure 34: Login wireframe	52
Figure 35: Register wireframe	52
Figure 36: Main activity wireframe	53
Figure 37: Pet profile wireframe	53
Figure 38: Maps activity wireframe	53
Figure 39: Feedback wireframe	53
Figure 40: Login screen	54
Figure 41: Register screen	54
Figure 42: Main activity screen	55
Figure 43: Refresh view by swiping down	56
Figure 44: Actual map activity	56
Figure 45: Map expanded options	56
Figure 46: Defining geofence	57
Figure 47: Pet profile activity	59
Figure 48: Pet profile edit	59

BACHELOR THESIS

Figure 49: Permission to use camera	60
Figure 50: Select photo option	60
Figure 51: Color picker	61
Figure 52: Toolbar	61
Figure 53: Navigation Drawer	62
Figure 54: Error displayed when input fields are empty	62
Figure 55: Alert on removal of pet	63
Figure 56: Toast saying a pet is successfully registered	63
Figure 57: Unfocused input field	63
Figure 58: Focused input field	63
Figure 59: Smartphone market shares 2009-2017 (Statista.com 2016)	65
Figure 60: Smartphone market share Norway 2016 (gsStatcounter.com 2017)	66
Figure 61: Retrospective - 2018.02.16	1
Figure 62: Retrospective - 2018.03.02	1
Figure 63: Retrospective - 2018.03.19	2
Figure 64: Retrospective - 2018.04.03	2
Figure 65: Retrospective - 2018.04.16	3
Figure 66: Retrospective - 2018.04.20	3
Figure 67: Retrospective - 2018.04.30	4

TABLE OF TABLES

Table 1: HTTP methods	7
Table 2: HTTP status codes	8
Table 3: How to use branches in Gitflow	28
Table 4: Hardware components	29
Table 5: Wiring Diagram - Arduino, GSM and GPS	38
Table 6: Estimated Power Consumption	38
Table 7: Prices for Telenor M2M-subscriptions and additional services (Telenor u.d.)	39
Table 8: Data usage (650 bytes pr. connection) and the corresponding subscription cost	40
Table 9: Explanations of the components that make up the MVC	42
Table 10: DB Table – User_account	48
Table 11: DB Table - Pet	49
Table 12: DB Table - Status	49
Table 13: DB Table - Ownership	49
Table 14: DB Table - Device	49
Table 15: DB Table - Bugreport	50
Table 16: Full list of features in map activity	58
Table 17: Pet ownership Norway 2008, (Aftenposten 2008) validated with (Regjeringen.no 2004)	65
Table 18: Estimated cost	66
Table 19: Estimated Device Profit First Year	67
Table 20: Continuous Monthly Profit	67
Table 21: First Year Profit Estimates	67

TABLE OF CODE SNIPPETS

Code Snippet 1: Json Example	Error! Bookmark not defined.
Code Snippet 2: Declaring required permission	14
Code Snippet 3: Runtime permission check for contacts (Android, Request permission 2018)	15
Code Snippet 4: Declaring feature	15
Code Snippet 5: Method updateUser in class UserRepository	18
Code Snippet 6: Method convention	21
Code Snippet 7: Nested methods convention	22
Code Snippet 8: Method separation convention	22
Code Snippet 9: X Bad code with comment. Should be refactored (Martin 2008)	22
Code Snippet 10: ✓ Better code, with no unnecessary comment (Martin 2008)	22
Code Snippet 11: Password hashed with Bcrypt	24
Code Snippet 12: Liquibase changeset	24
Code Snippet 13: Encoded JSON web token	25
Code Snippet 14: Decoded header	25
Code Snippet 15: Decoded payload	25
Code Snippet 16: Signature	26
Code Snippet 17: User Controller declaration	35
Code Snippet 18: Method create User in class UserController	36
Code Snippet 19: UserService declaration	36
Code Snippet 20: Method create User in class UserService	36
Code Snippet 21: UserRepository declaration	36
Code Snippet 22: Method mapRow in UserRowMapper	37
Code Snippet 23: Implementation of an AsyncTask for user login	44
Code Snippet 24: Usage of LoginTask in the class LoginActivity	44
Code Snippet 25: Method getUser in the class UserHttpClient	45

APPENDIX

Appendix 1	Preliminary report
Appendix 2	Retrospective reports
Appendix 3	Meeting notes
Appendix 4	Burndown charts
Appendix 5	Class Diagram (uploaded zip-file)
Appendix 6	Source code (uploaded zip-file)

SOURCE CODE

Backend	: https://bitbucket.org/trackapetgroup/rest/src/master/
Frontend	: https://bitbucket.org/trackapetgroup/android/src/master/
Tracker	: https://bitbucket.org/trackapetgroup/tracker_v3/src/master/

Appendix 1 – Preliminary Report

APPENDIX 1 – PRELIMINARY REPORT

TITTEL: Forprosjekt - Track'a'Pet

--

STUDENTNUMMER(E): 10021 10019 10050			
DATO: 24/01/18	EMNEKODE: IE303612	EMNE: Bacheloroppgave	DOKUMENT TILGANG Åpen
STUDIUM: Dataingeniør		ANT SIDER/VEDLEGG: 22/0	BIBL. NR.: Ikke i bruk -

OPPDRAGSGIVER(E)/VEILEDER(E): Egendefinert oppgave / Girts Strazdins
--

Postadresse
NTNU i Ålesund
N-6025 Ålesund
Norway

Besøksadresse
Larsgårdsvegen 2
Internett
www.ntnu.no

Telefon
70 16 12 00
Epostadresse
postmottak@ntnu.no

Telefax
70 16 13 00
Bankkonto
7694 05 00636
Foretaksregister
NO 971 72 140

Kjetil Yndestad
Anders Grytten Standal
Kay Sindre Lorgen

INNHOLDSFORTEGNELSE

I. PREFACE.....	I
II. TABLE OF CONTENTS.....	II
III. SUMMARY.....	VII
IV. TERMINOLOGY.....	VIII
CONCEPTS	VIII
ABBREVIATIONS	IX
1 INTRODUCTION.....	1
1.1 BACKGROUND.....	1
1.2 PURPOSE AND APPROACH.....	1
1.3 THESIS STRUCTURE	1
2 THEORETICAL BASIS	2
2.1 LAWS AND REGULATIONS	2
2.2 INFORMATION SECURITY.....	3
2.3 STANDARDS	7
2.4 AGILE DEVELOPMENT	10
2.5 OBJECT ORIENTED PROGRAMMING	10
2.6 DESIGN PATTERNS AND PRINCIPLES.....	11
2.7 MAN-MACHINE INTERACTION.....	12
2.8 ANDROID AND ANDROID STUDIO	14
3 MATERIALS AND METHODS	16
3.1 PROJECT ORGANIZATION.....	16
3.3 PRE-PLANNING.....	17
3.4 ARCHITECTURE	18
3.5 PROGRAMMING LANGUAGES	21
3.6 CODING CONVENTION	21
3.7 EXTERNAL LIBRARIES – MICROCONTROLLER	23
3.8 EXTERNAL LIBRARIES - BACKEND	24
3.9 EXTERNAL LIBRARIES – ANDROID	26
3.10 DEVELOPMENT TOOLS.....	27
3.11 SUPPORT TOOLS	28
3.12 TESTING METHODS	29
3.13 TESTING TOOLS	29
3.14 MATERIALS	29
3.15 DOCUMENTATION.....	30
4 RESULTS	33
4.1 ARCHITECTURE	33

Appendix 1 – Preliminary Report

4.2	BACKEND ARCHITECTURE	34
4.3	MICROCONTROLLER	38
4.4	FRONTEND ARCHITECTURE	42
4.5	USE CASE DIAGRAM	45
4.6	CLASS DIAGRAM	46
4.7	DATABASE DESIGN	47
4.8	SECURITY	50
4.9	DESIGN	52
4.10	TESTING	64
4.11	BUSINESS ANGLE	65
5	DISCUSSION	68
5.1	TECHNICAL RESULT	68
5.2	PROJECT EXECUTION	73
6	CONCLUSION.....	74
7	REFERENCES.....	75
	TABLE OF FIGURES	82
	TABLE OF TABLES	84
	TABLE OF CODE SNIPPETS	85
	APPENDIX	86
	APPENDIX 1 – PRELIMINARY REPORT.....	1
	APPENDIX 2 – RETROSPECTIVE.....	1
	APPENDIX 3 – MEETING NOTES.....	1
	APPENDIX 4 – BURNDOWN CHARTS	1

1 INNLEDNING

Savner du kjæledyret ditt? Ved hjelp av Track'a'Pet trenger du ikke bekymre deg lenger.

I en verden med stadig økt usikkerhet er behovet for dynamiske sikkerhetstiltak rundt kjæledyr mer framtreddende. Det finnes et behov for økt sikkerhet og kontroll på kjæledyr, som kan dekkes av dagens teknologi.

Oppgaven omfatter å utvikle et system som skal holde kontroll over GPS posisjonene til kjæledyr og rapportere det tilbake til bruker.

Valget falt på denne oppgaven siden den omfatter de fleste elementer og fagfelt fra studiet vårt, er meget omfattende og kompleks, samtidig som oppgaven har samfunnsmessig nytte utover læringsmål. Det vil også være i oppgavens natur å koble forskjellig teknologi sammen, som for eksempel mikrokontrollere og Android programmering. Videre vil sikkerhet stå sentralt, siden uautorisert tilgang kan få store konsekvenser for bruker og kjæledyr.

Oppdragsgiveren i dette tilfellet er oss selv.

2 BEGREPER

2.1 Begreper

Man-in-the-middle-angrep

Angrep hvor angriperen releer og potensielt endrer meldinger mellom to parter som tror de kommuniserer direkte. [Scrum](#)

Scrum er en smidig metode for å utvikle programvare. Denne metoden baserer seg på dynamiske prosesser som skal fremme kvaliteten på programvaren som blir utviklet. Metoden fungerer med at man jobber i mindre perioder, kalt en sprint, noe som gjør det enklere å gjøre kritiske endringer underveis i prosessen.

Sprint

En sprint i Scrum er en tidsperiode ofte definert fra en til tre uker. Innenfor denne perioden blir det foretatt utvikling, hvor ved enden av perioden den overordnet planen for prosjektet blir revidert for å ta høyde for nyvunnen kunnskap. [Product backlog](#)

I Scrum opererer vi med en backlog (funksjonalitetsbank) som rommer alle aktivitetene vi har og skal gjennomføre for at produktet vil bli ansett som fullført. Backloggen blir dynamisk lagt til etterhvert som nye features er ønsket, samtidig som man oppdager bugs som må fikses. Derfor vil prosjektet i utgangspunktet begynne med mange issues i backloggen, som blir behandlet med tid.

Issue

En issue i Scrum er en klart definert oppgave som skal gjennomføres for å legge til eller redigere funksjonalitet i programvaren. Dette vil være de overordnede oppgaven som må gjennomføres. Som for eksempel fjerne en bug, legge til funksjonalitet og lignende.

Planning Poker

Planning poker er en metode for å vekte arbeidsmengden hvert issue burde vektes. Det gjøres dynamisk i gruppen i begynnelsen av prosjektprosessen. Hvor hvert medlem i gruppen angir poeng på hvert issue, med en gitt skala. Estimater blir da diskutert og man kommer fram til en enighet. Slik unngår man en poeng smitteeffekt innad i gruppen, hvor første tallet blir gjeldende.

JIRA

JIRA er et issue tracking program som gjør det mulig å holde oversikt over alle issueene som står igjen og organisere prosessen på en lettfattelig måte. Fra JIRA sin backlog organiseres det sprinter og rapporter som burndown charts. [Confluence](#)

Confluence er et samarbeidsverktøy for dokumentasjon.

Bug

En bug er en uønsket hendelse i programvare. Dette kan være noe som ikke fungerer, eller noe som fungerer feil. En bug oppstår da en programvare inneholder mye forskjellig kode som kan være i konflikt, eller at man har konfigurert noe feil. Generelt er det tilnærmet umulig å ikke feilaktig innføre bugs i programmet.

Konfidensialitet

Når informasjon er beskyttet fra eksponering og formidling til uautoriserte personer eller systemer. Bare personer med riktig rettigheter skal kunne ha tilgang til sensitiv data.

Integritet

Når informasjon er hel, komplett, og ukorrupt.

Tilgjengelighet

Når autoriserte brukere eller systemer har tilgang til data uten obstruksjon eller forstyrrelser, og i riktig format.

GIT

Versjonskontroll program som sikrer forbedret team samspill ved å gi mulighet for mellomlagring og synkronisering innad i teamet.

Bitbucket

Distribuert hostingtjeneste for versjonskontroll.

Bruker

Sluttbruker av systemet, i dette tilfellet eier av kjæledyr.

Backend

Grunnleggende system sluttbruker ikke dirkete arbeider mot. Backend systemet står for den databærende funksjonalitet.

Frontend

Systemet sluttbrukeren jobber mot. Gjerne grafisk framstilling og interaksjon med systemet.

Android

Operativsystem som benyttes på mange mobiltelefoner.

2.2 Forkortelser

GPS - Global Positioning System

GSM - Global System of Mobile communication

UML - Unified Modeling

Language GUI - Graphical User

Interface

SQL – Structured Query Language

REST - REpresentational State Transfer

IDE – Integrated Development

Environment

3 PROSJEKTORGANISASJON

3.1 Prosjektgruppe

Studentnummer	Navn	Epost
060274	Kjetil Yndestad	kjetilyn@stud.ntnu.no
460039	Kay Sindre Lorgen	kaysl@stud.ntnu.no
269644	Anders Grytten Standal	andergs@stud.ntnu.no

Tabell 1: Studentnummer(e) og navn på gruppedlemmene

3.1.1 Oppgaver for prosjektgruppen - organisering

Dette prosjektet vil bli organisert delvis ved hjelp av Scrum utviklingsmetodikk. Derfor vil rollene for gruppedlemmene være dynamisk og tidvis skiftende. Siden vi er en liten gruppe har vi sett det som nødvendig at gruppedlemmene opererer med flere funksjoner.

Rolle	Navn
Scrum-master i Februar	Kay Sindre Lorgen
Scrum-master i Mars	Kjetil Yndestad
Scrum-master i April	Anders Grytten Standal
Utvikler	Kay Sindre Lorgen
Utvikler	Kjetil Yndestad
Utvikler	Anders Grytten Standal
Sekretær	Kjetil Yndestad
Beta tester	Linus (the cat)

Tabell 2: Rollefordeling

Utover dette forventes det også at gruppen innfrir de følgende arbeidsoppgavene.

- Overholde frister for møter
- Sette opp systemet
- Utvikle programvaren
- Rapporteringsarbeid

3.1.2 Oppgaver for Scrum-master

Scrum-masteren er ansvarlig for at Scrum-rammeverket blir fulgt. Sørge for at produkteier forstår sin rolle og at samspillet mellom disse fungerer godt. Scrum-master hjelper utviklingsteamet til å bli selvorganisert og om nødvendig skjermer utviklingsteamet mot ytre forstyrrelser.

3.1.3 Oppgaver for utviklingsteamet

Scrum-teamet tar kollektivt ansvaret for å skape mest mulig verdier for interessentene.

3.1.4 Oppgaver for sekretær

Overordnet ansvar for å planlegge møter med veileder og løpende loggføring av møter. Ved fravær vil det falle på de andre gruppemedlemmene å utføre denne rollen.

3.2 Styringsgruppe

Navn	Epost	Rolle
Girts Strazdins	gist@ntnu.no	Veileder

Tabell 3: Kontaktinformasjon for styringsgruppen

4 AVTALER

4.1 Arbeidssted

Gruppen disponerer arbeidsplass på NTNU Ålesund sine lokaler, hvor L167 er sentralt. Videre har vi gruppen også tilgang til visualiserings-laboratoriet og diverse grupperom på bygget. Ved mangelfull arbeidsplass kan eget bosted brukes som hjemmekontor.

4.2 Ressurser

Funksjonalitet	Navn
Utviklingsverktøy til REST tjener	Netbeans
Utviklingsverktøy til Android	Android Studio
Oppgavehåndtering	JIRA
Prosjektdokumentasjon	Confluence
Rapportskriving	MS Word (delt dokument)
Versjonskontroll distributør	Bitbucket
Database	PostGRES
SSL sertifikat	Let's Encrypt
Webserver	Apache

Tabell 5: Ressurser

4.2.1 Tilgang på personer

Veileder er løpende tilgjengelig for spørsmål på epost, samtidig som det blir foretatt ukentlige møter med veileder.

Gruppedeltakerne er tilgjengelig på NTNU sine lokaler i ordinær arbeidstid. Ut over dette er alle i utgangspunktet tilgjengelig på epost.

4.2.2 Datasikkerhet

I henhold til personopplysningsloven skal vi gjennom planlagte og systematiske tiltak sørge for tilfredsstillende informasjonssikkerhet med hensyn til konfidensialitet, integritet og tilgjengelighet ved behandling av personopplysninger.

4.2.3 Avtalt rapportering

Det er avtalt å rapporteres løpende under hele utviklingsprosessen, da dokumentasjonen i denne prosessen vil være dynamisk og derfor krever konstant revidering. I tillegg skal det rapporteres på engelsk.

Det skal utarbeides meeting notes i confluence til ukens møte, som skal fungere som agenda for ukens møte. Disse skal i tillegg fungere som møtoreferater ved at sekretær fyller ut stikkord fra møtet underveis.

4.3 Gruppenormer – samarbeidsregler – holdninger

4.3.1 Gruppenormer og samarbeidsregler

- Møte opp i god tid til ukentlige møter med veileder
- Gi beskjed i god tid hvis man ikke kan møte opp på skolen
- Samtlige medlemmer i prosjektgruppen har like mye innflytelse over avgjørelser relatert til prosjektet
- Programvaren som blir utviklet skal ha tiltak for å sikre informasjonssikkerhet, i henhold til konfidensialitet, integritet og tilgjengelighet

4.3.2 Holdninger til dataingeniørfaget som profesjon

- Utføre oppgaver på en profesjonell måte
- Hjelp andre prosjektmedlemmer når det er behov for det
- Skrive ren og opprettholdbar kode
- Koden skal bestå alle tester før den går i produksjon
- Gi ærlige tidsestimat
- Ikke overta arbeidsoppgaver fra andre prosjektmedlemmer uten vedkommendes samtykke

5 PROSJEKTBEKRIVELSE

5.1 Problemstilling - målsetting - hensikt

Problemstillingen i dette prosjektet er å utvikle en sporer som kan kommunisere med en sentral server, samt utvikle et grafisk grensesnitt for sluttbruker.

Hensikten og målsettingen er å utvikle et system som innfrir alle de implisitte kravene som ligger i problemstillingen. Vi skal utvikle en databaseløsning, REST-basert serveroppsett, Android-applikasjonen og mikrokontroller med GPS som kommuniserer over GSM.

Hovedmålet er å utvikle en applikasjon som viser posisjon og posisjonshistorikk til et valgt kjæledyr. Applikasjonen skal fungere på en hensiktsmessig og tilfredsstillende måte.

Målnivå	Målformulering
Effekt mål	<ul style="list-style-type: none"> • Det hovedsakelige effekt målet er å tilby kjæledyreiere en enkel og effektiv måte å finne ut hvor dyrene deres befinner seg, og hvor de har vært. • Kutte ned tiden det tar en dyreeier å lete etter kjæledyr.
Ressurs mål	<ul style="list-style-type: none"> • Det skal utvikles et halsbånd til kjæledyr som loggfører GPS lokasjonene til dyret, som maskinvaren kommuniserer til en sentral database. • Det skal settes opp en database som tar imot GPS lokasjonene til flere kjæledyr. • Lokasjonsdata skal vises for brukeren via et GUI i form av en Android applikasjon. • Det skal i tillegg utvikles en web GUI som gjør det mulig for bruker å administrere abonnement sitt via en nettside knyttet til prosjektet. • Systemet skal innfri de moderne kravene til datasikkerhet.
Samfunns mål	<ul style="list-style-type: none"> • Systemet kan ha en avskrekkende effekt på pøbler og forkastelige personer som ønsker å skade eller bortføre kjæledyr.

Tabell 6: Målformulering

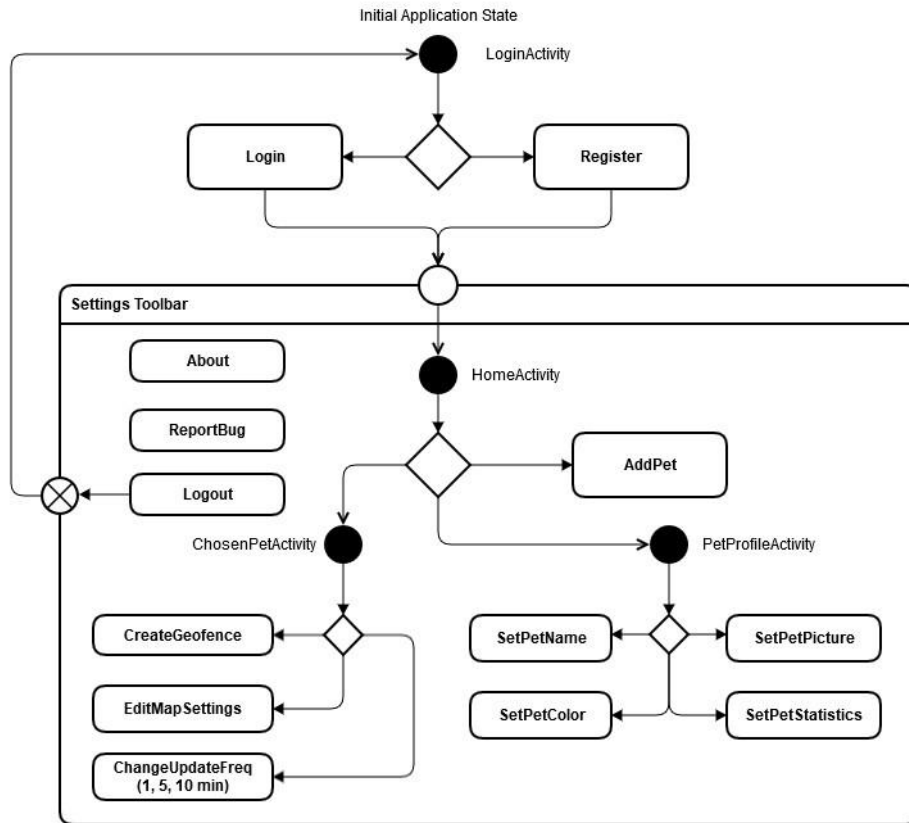
5.2 Krav til løsning eller prosjektresultat – spesifikasjon

5.2.1 Kravspesifikasjon

Track'a'Pet skal ha følgende funksjonalitet:

Handlinger fra brukergrensesnitt

- Bruker registrerer seg i applikasjonen
 - Fornavn/etternavn
 - Passord
 - Epost
- Bruker logger inn i applikasjonen
- Bruker administrer konto
 - Bruker endrer/velger passord
 - Bruker endrer personinstillinger
- Bruker registrerer kjæledyr i applikasjonen
 - Navn
 - Type dyr
 - Hjemmeadresse
 - Bilde/Avatar
 - Endre markørfarge
- Bruker sporer kjæledyr ved å se lokasjonen på et kart
- Bruker aktiverer GPS på telefon for å se sin egen lokasjon i forhold til kjæledyr
- Bruker kan spore flere kjæledyr samtidig
- Bruker kan melde kjæledyr savnet



Figur 1: Aktivitetsdiagram

5.2.2 Leveranse fra prosjektet

For at prosjektet skal bli ansett som fullført, kreves følgende:

- Grafisk brukergrensesnitt med alle kravene spesifisert i 5.2.1
- Backend-tjener som håndterer lagring av data og kommunikasjon mellom brukergrensesnitt og hardwaremodul
- Generell dokumentasjon
- God kodelstil på samtlige systemer
- God GUI utforming

5.3 Planlagt framgangsmåte for utviklingsarbeidet

Prosjektstyringen vil hovedsakelig foregå gjennom JIRA, hvor vi setter ukentlige sprints for prosjektet. Utover dette benyttes planning poker for å vekte arbeidsmengden hvert issue omfatter.

I prosjektet vil vi benytte oss av en smidig utviklingsmetode som heter Scrum. I Scrum deler man opp arbeidet inn i mindre oppgaver som kan bli løst i løpet av en gitt tidsramme; en *sprint*. Hovedideen i Scrum er at kravet til produktet endrer seg underveis, så planen skal holde seg fleksibel. Arbeidsflyten er følgende:

- Produkteieren lager en prioritert ønskeliste av funksjoner som blir kalt for en produktkø
- Når teamet planlegger neste sprint velger de seg et sett med oppgaver fra produktkøen
- Teamet jobber med å implementere oppgavene i løpet av sprintperioden, samtidig som de har daglige stand-up møter hvor de diskuterer fremgangen
- Scrum-masteren jobber konstant med å holde teamet fokusert på målet
- På slutten av en sprint skal teamet sitte igjen med et produkt som er klar til bruk, selv om ikke alle oppgavene nødvendigvis er ferdige
- Oppgaver som ikke ble gjort denne sprinten blir lagt på toppen av sprintkøen
- Sprinten avsluttes med en sprintgjennomgang og retrospektiv
- Teamet velger seg ut oppgaver for neste sprint fra produktkøen

I Scrum jobber man i *sprinter*, altså iterasjoner med en fast lengde. Når en sprint starter, skal teamet helst levere det omfanget fra produktkøen de tror de kan klare å gjennomføre. Når en sprint er ferdig skal teamet demonstrere det de har laget for interessentene og produkteier. Til slutt skal teamet reflektere over hva som gikk bra, og hva som burde forbedres til neste sprint.

I vårt tilfelle vil en sprintene først vare 14 dager. Etter hvert som vi blir ferdig med det obligatoriske faget *Ingeniørfaglig systemteknikk og systemutvikling*, vil vi vurdere å begynne med 7-dagers sprinter. Vi vil ha cirka like mange arbeidsdager siden vi ikke har andre fag å jobbe med.

Produkteieren representerer prosjektets interessenter. I tillegg skal vedkommende sikre at teamet til enhver tid jobber med de rette oppgavene sett fra et forretningsperspektiv. Eieren er ansvarlig for at produktkøen er prioritert og estimert for interessentene og teamet.

Scrum-masteren skal være ansvarlig for at teamet lærer seg å selv organisere, at produkteier forstår sin rolle, og at samspillet fungerer som det skal. Scrum-master kan også være ansvarlig for at teamet ikke blir distraheret av utvendige forstyrrelser.

Utviklingsteamet er ansvarlig for å utvikle produktet. Det er som regel et relativt lite team på 3-9 personer. Teamet har ansvar for å skape mest mulig verdier for interessentene.

Det er flere fordeler ved å benytte seg av Scrum. For det første kan man fort ta i bruk produktet selv om det ikke er ferdig. Ved å ha en ferdig versjon på slutten av hver sprint sørger man for tidlig verdiskaping.

En annen fordel er at det er mindre kostbart å gjøre endringer. Ved å bruke vanlige fossefallsmetoder vil det ofte være dyrt og tidkrevende å endre funksjonalitet i prosjektet før det er ferdig. Siden Scrum er såpass dynamisk, kan man lett endre betydelige deler av produktet uten for mye ekstra arbeid.

Scrum har også noen svakheter. I prosjekter der man fra starten av har full kontroll på kravene og løsningen vil det ikke gi noen økt gevinst ved å bruke Scrum, med tanke på at man har liten mulighet til å fastsette krav på forhånd.

Scrum fungerer heller ikke optimalt der team er geografisk delt eller noen medlemmer jobber på deltid. Det er fordi Scrum er lagt opp til at utviklere skal arbeide tett, med pågående samarbeid.

5.4 Informasjonsinnsamling – utført og planlagt

Så langt har vi funnet flere løsninger som ligner på det vi skal oppnå i dette prosjektet. Disse varierer veldig i funksjonalitet og pris. Vi kan bruke disse som utgangspunkt når vi for eksempel skal finne ut hvor lenge batteriet på halsbåndet skal kunne vare mellom hver ladning.

Videre gjenstår det å finne ut hvordan vi kan gjøre hardwaren, altså modulen i halsbåndet, så lite som mulig.

5.5 Vurdering – analyse av risiko

Med tanke på at hardware spiller en vesentlig rolle i prosjektet, er det en liten mulighet for at prosjektet ikke blir realisert innenfor den gitte rammen. Det er mange kilder til feil når man jobber med slik hardware. Vi må ta hensyn til batteritid, at kommunikasjonen over GSM fungerer, at GPS gir riktig plassering, og at selve modulen er liten nok og tåler å bli utsatt for regn og annet vær.

Et annet risikoelement er sikker kommunikasjon mellom sporer modulen og den sentrale serveren. Kommunikasjon må krypteres for å forhindre man-in-the-middle angrep. Slike operasjoner på mikrokontrollere kan ofte være kostbart, med tanke på prosesseringsevne og batteritid.

Leveransetid på hardware kan også være en fallgrop. Slike varer må som regel bestilles fra utlandet, der vi må sette av tid til frakt og eventuelt toll. Vi kan begynne å planlegge og programmere før vi har den nødvendige hardwaren, men har da ikke mulighet for å teste om det fungerer.

Når det kommer til softwaredelen av prosjektet, er det en liten fare for at vi får for lite tid. Det er mange elementer vi ønsker å ha med, og i flere av tilfellene kommer vi til å bruke teknologi vi ikke har jobbet med tidligere. Oppsiden er da at oppgaven er såpass modulær at vi kan skalere prosjektet etter hvert som vi jobber.

ID	Beskrivelse
A	Sjukdom
B	Problemer rundt mikrokontroller prototype.
C	Arbeidsmengden blir for stor

Tabell 7: Identifisering av risiko

Mulighets-/ Risiko- matrise	Konsekvens							
	Mulighet				Risiko			
Sann- synlighet	> 7 dager	2- 7 dager	1 - 2 dager	< 1 dag	< 1 dag	1 -2 dager	2 - 7 dager	> 7 dager
Veldig stor > 50 %							B	
Stor 25-50 %								
Middels 5-25 %							A	C
Liten 1-5 %								
Usynlig < 1 %								

Figur 2: Mulighet og risiko analyse

5.5.1 Handlingsplan

ID	Tiltak
A	God kommunikasjon og dokumentasjon internt i gruppen. Om et av medlemmene blir sjuk har dem andre i gruppen kunnskap og forståelse nok til å fortsette arbeidet.
B	Simulere data som prototypen sender ved hjelp av software.
C	En av fordelene med smidige metoder er at vi kan tilpasse arbeidsmengden underveis og vil prioritere funksjonaliteten. På denne måten kan vi skalere ned om arbeidsmengden blir for stor og levere et fungerende produkt.

Tabell 8: Handlingsplan

5.6 Hovedaktiviteter i videre arbeid

Hovedaktivitetene har en logisk sammenheng, hvor hovedaktivitetene må forekomme i kronologisk rekkefølge. Videre kan noen av delaktivitetene bli omrokkert ved behov.

Nr	Hovedaktivitet	Omfang
A1	Utvikle prototype	3 uker
A1.1	Identifisere alternativer for prototype	
A1.2	Programmere hardware	
A1.3	Test GSM kommunikasjon	
A1.4	Test GPS kommunikasjon	
A2	Backend	3 uker
A2.1	Design database	
A2.2	Implementasjon av databaseløsning	
A2.3	RESTful design	
A2.4	RESTful implementasjon	
A2.5	Backend konfigurasjon	
A2.6	Kommunikasjon mellom prototype og API	
A3	Frontend	10 uker
A3.1	Design av android applikasjon	
A3.2	Utvikle android applikasjon	
A3.3	Teste android applikasjon	
A3.4	Design av web frontend	
A3.5	Utvikle web frontend	
A3.6	Teste web frontend	
A3.7	Kommunikasjon mellom applikasjon og API	
A4	Systemteknisk validering	2 uker
A4.1	Android systemtesting	
A4.2	Web frontend systemtesting	

Tabell 9: Hovedaktivitetene i prosjektet

5.7 Framdriftsplan – styring av prosjektet

5.7.1 Hovedplan

Gruppen kommer til å benytte seg av smidige metoder. Vi planlegger å ha sprints som varer 14 dager ut februar for så gå over til 7 dagers sprint fra Mars. I tillegg skal vi ha daglige Scrum-møter for å kartlegge fremgang og videre planlegge hva som skal gjøres i nærmeste fremtid.

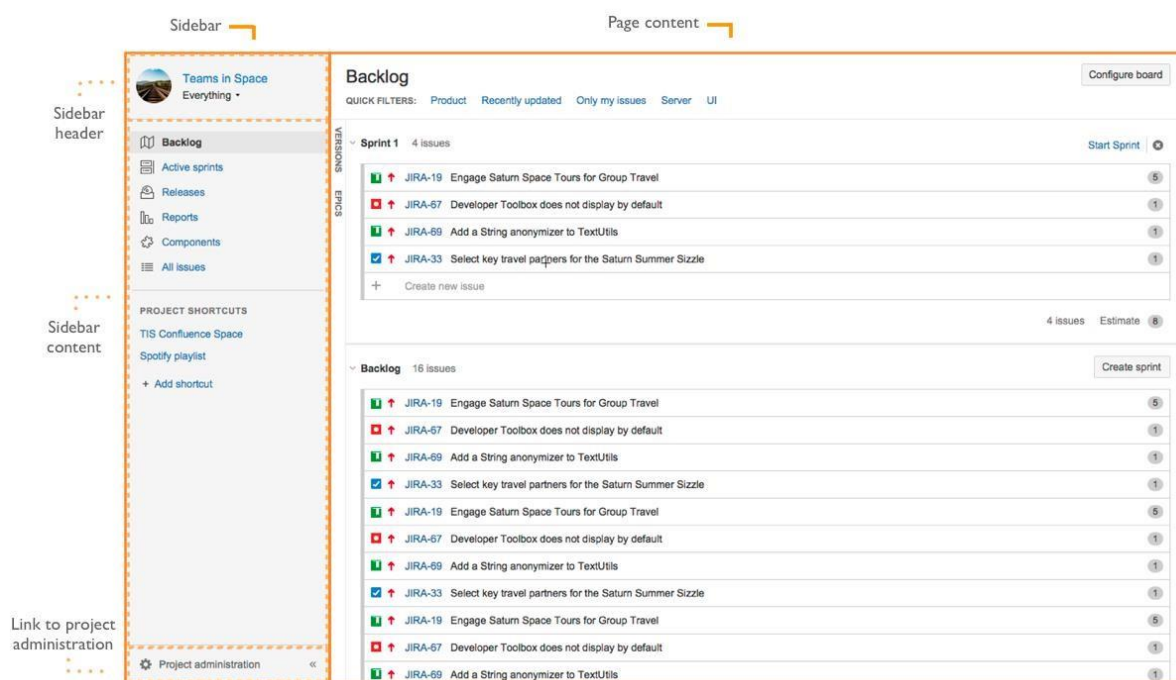
Gruppen vil benytte seg av oppgavehåndteringsverktøyet JIRA for å planlegge sprintene, samtidig som gruppen planlegger hvilke oppgaver hvert medlem arbeider med i begynnelsen av hver sprint. Ved uenigheter har Scrum-master ansvar for å komme fram til en fornuftig løsning.

Rapporten skal skrives underveis før siste uken settes av til å sy sammen rapporten og avsluttende finpuss.

5.7.2 Styringshjelpemidler

JIRA

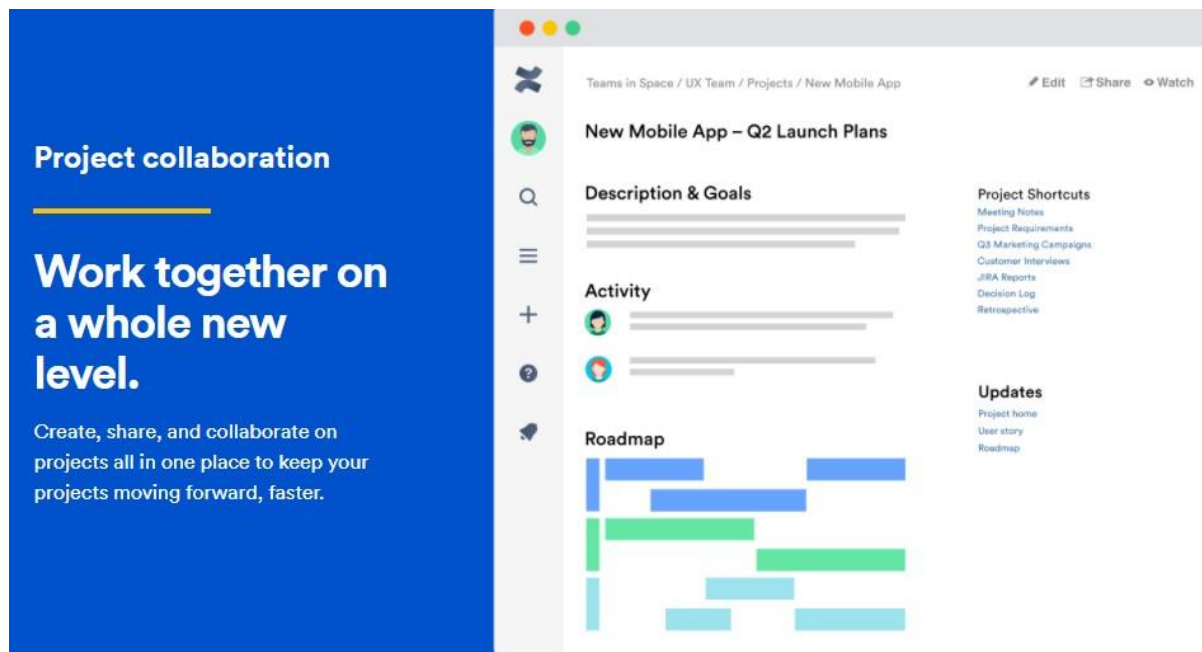
JIRA er et issue tracking program som gjør det mulig å holde oversikt over alle issueene som står igjen og organisere prosessen på en lettfattelig måte. Fra JIRA sin backlog organiseres det sprinter og rapporter som burndown charts.



Figur 3: Eksempelbilde av JIRA

Confluence

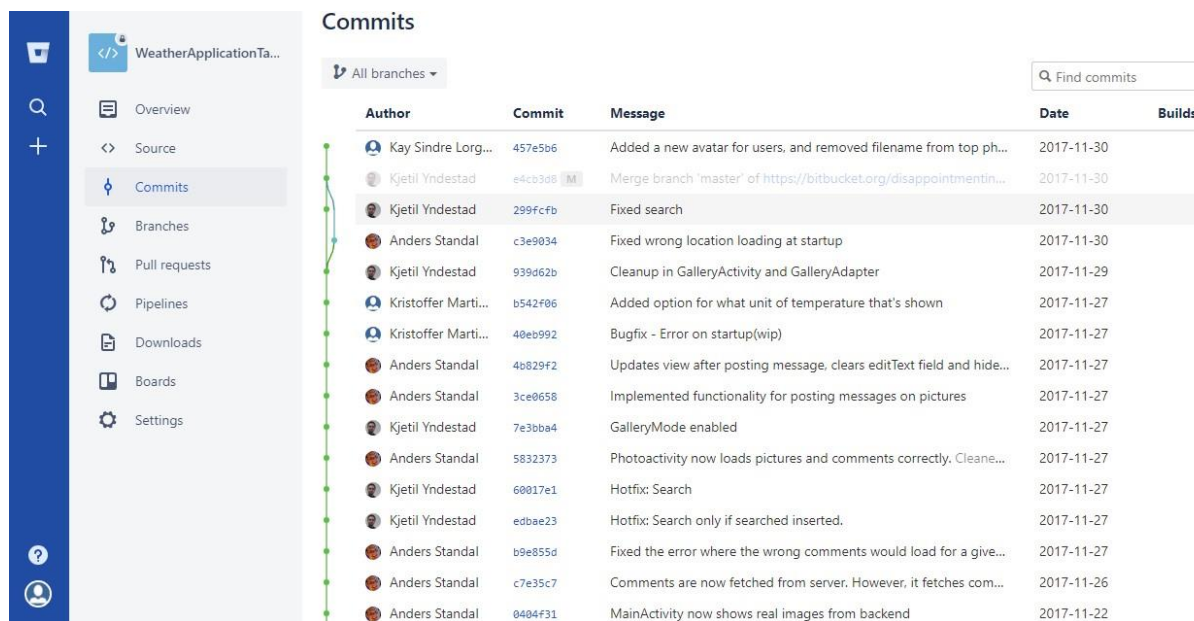
Confluence er programvare for innholdssamarbeid og gir ditt team et sentralt sted å holde prosjektets arbeid organisert og tilgjengelig.



Figur 4: Eksempelbilde av Confluence

Bitbucket

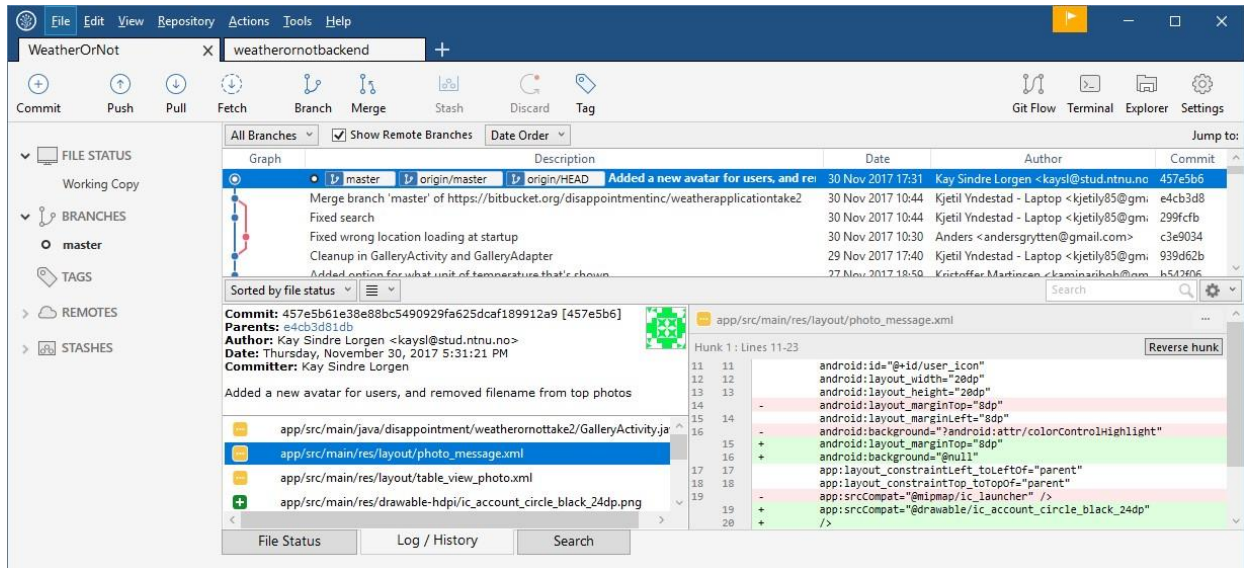
Bitbucket er et skalerbart distribuert versjonskontrollsystem som gjør det enkelt å samarbeide med teamet ditt.



Figur 5: Eksempelbilde av Bitbucket

Sourcetree

Sourcetree forenkler hvordan du samhandler med GIT-repoene dine slik at du kan fokusere på koding. Visualiser og administrer GIT-repoene via GUI.



Figur 6: Eksempelbilde fra Sourcetree

5.7.3 Utviklingshjelpemidler

Netbeans

Netbeans er et utviklingsverktøy som primært er utviklet med hensyn på Java. Det gir brukeren en god mengde ekstra funksjonalitet og hjelp når man skal programmere. Dette fører til at programmeringsprosessen blir effektivisert.

Android Studio

Android Studio er et utviklingsmiljø for programmering av applikasjoner til Android telefoner. Android studio gjør det også mulig å teste applikasjonene på i et virtuelt kjøremiljø, hvor dette effektiviserer noen av debuggingsprosessen.

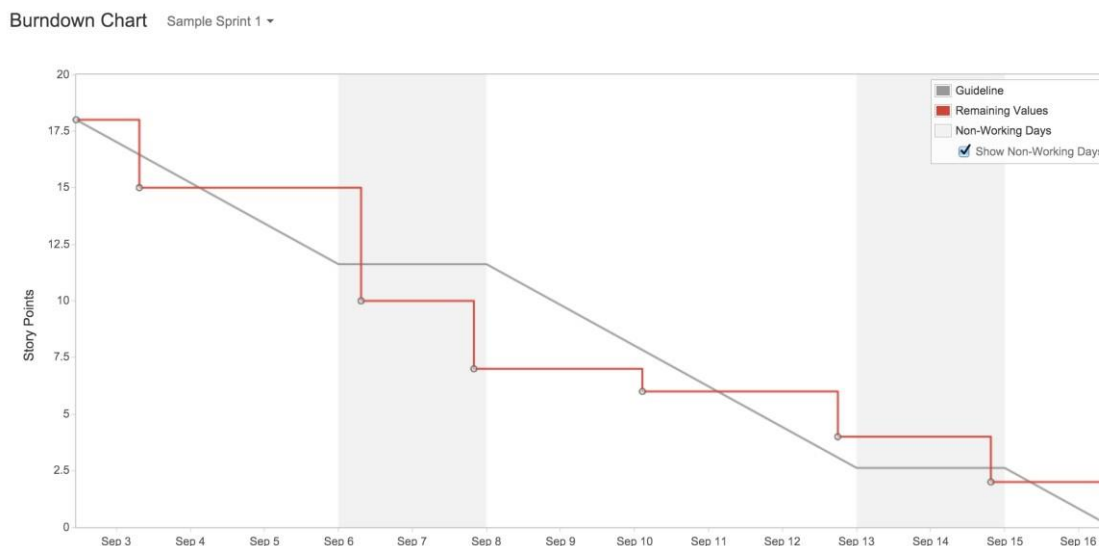
Arduino

Arduino er en IDE som er utviklet for å gjøre det lett å programmere en Arduino mikrokontroller. Det støtter i tillegg seriellport overføring, noe som gjør det mulig å overvåke mikrokontrolleren i sanntid via utviklingsmiljøet.

5.7.4 Intern kontroll – evaluering

Burndown chart

For hver sprint sjekker vi burndown chart og får en pekepinn på hvordan arbeidet har gått utover sprinten. Ser vi at vi ikke har kommet i mål eller blitt for tidlig ferdig med sprinten kan vi justere arbeidsmengden vi planlegger for neste sprint.



Figur 7: Eksempel på burndown chart

5.8 Beslutninger – beslutningsprosess

5.8.1 Avgrensning

Vi bestemte oss tidlig for hva vi ville oppnå i prosjektet. Vi tegnet opp en overfladisk arkitektur (figur 8) som viser hva slags komponenter vi skal ha med i systemet, hvordan de interagerer, og hvilke programmeringsspråk som skal tas i bruk. Ved hjelp av veileder kom vi frem til hva som er et realistisk mål for prosjektet.

5.8.2 Kjøremiljø

Vi valgte å bruke Spring Boot som rammeverk i backend-systemet. Dette valget baserte vi på at:

- Det er enkelt å sette opp og raskt få i gang applikasjoner ● Det tilbyr enkel og kraftig databasestøtte
- Det reduserer mengden boilerplate kode man må skrive ● Det tilbyr enkel dependencyhåndtering

For databasesystemet valgte vi å bruke PostgreSQL. Noen av fordelene med det er at:

- Det er gratis og åpen kildekode
- Det tilbyr mye funksjonalitet, samtidig som den ikke er veldig ressurskrevende
- Det er pålitelig, kjent for å ikke bryte sammen
- Det er tilgjengelig på nesten alle plattformer

5.8.3 Hardwaremodul

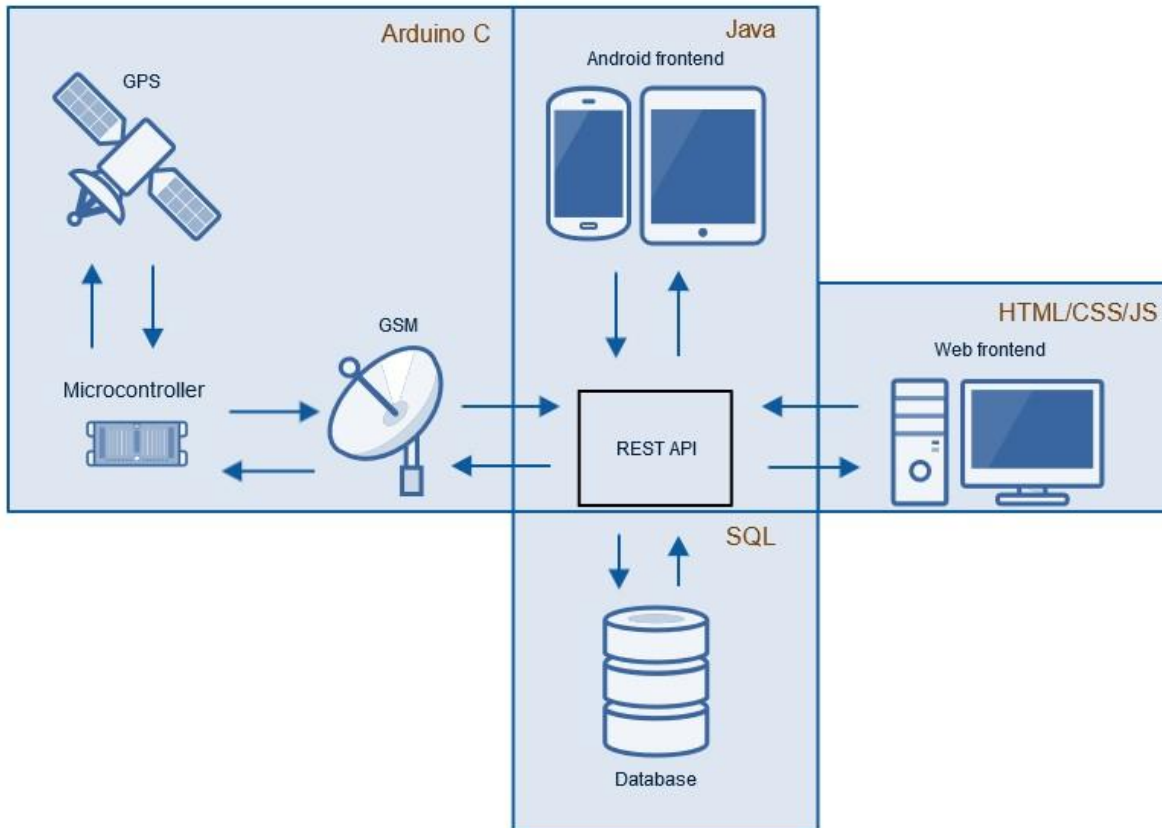
For selve sporeren velger vi å utvikle første prototypen med Arduino Uno og nødvendige ekstramoduler.

Det er hovedsakelig fordi Arduino er lett å jobbe med, og det tar lite tid å sette sammen noe som fungerer. Programmeringsspråket, Arduino C, er en forenklet versjon av C som er simplifisert slik at det skal være lettere og raskere å skrive kode, men det tilbyr fortsatt mye fleksibilitet.

5.8.4 Videre beslutninger

Store avgjørelser og beslutninger vil bli gjort internt i utviklingsteamet, siden medlemmer av teamet også er produkteier og Scrum-master.

5.8.5 Systemarkitektur



Figur 8: Overordnet systemarkitektur

6 DOKUMENTASJON

6.1 Rapporter og tekniske dokumenter

Gruppen ønsker at kildekoden skal følge beste praksis slik at den blir lettleselig nok til at dokumentasjon blir overflødig. Noe dokumentasjon av kode vil uansett være nødvendig i noen tilfeller.

Det vil føres møtereferat av sekretær i Confluence etter møte med veileder. Møtereferat skal leses av alle gruppemedlemmer og tillegg/feil skal rapporteres til sekretær.

JIRA vil også fungere som dokumentasjon. Der vil det være en logg over hvilke oppgaver som er gjort av hvem og når, i tillegg til å vise hvilke oppgaver som gjenstår.

Vi vil skrive retrospektiv i Confluence for avsluttet sprint hvor vi kartlegger hva som gikk bra og hva som gikk dårlig. På denne måten kan vi forbedre prosessen underveis i prosjektet.

7 PLANLAGTE MØTER OG RAPPORTER

7.1 Møter med styringsgruppen

Dato	Klokke	Formål	Lokasjon
19.01-2018	09:00	Møte med veileder	Hovedbygg B313
26.01-2018	09:00	Møte med veileder	Hovedbygg B313
02.02-2018	09:00	Møte med veileder	Hovedbygg B313
09.02-2018	09:00	Møte med veileder	Hovedbygg B313
16.02-2018	09:00	Møte med veileder	Hovedbygg B313
23.02-2018	09:00	Møte med veileder	Hovedbygg B313
02.03-2018	09:00	Møte med veileder	Hovedbygg B313
09.03-2018	09:00	Møte med veileder	Hovedbygg B313
16.03-2018	09:00	Møte med veileder	Hovedbygg B313
23.03-2018	09:00	Møte med veileder	Hovedbygg B313
06.04-2018	09:00	Møte med veileder	Hovedbygg B313
13.04-2018	09:00	Møte med veileder	Hovedbygg B313
20.04-2018	09:00	Møte med veileder	Hovedbygg B313
27.04-2018	09:00	Møte med veileder	Hovedbygg B313
04.05-2018	09:00	Møte med veileder	Hovedbygg B313
11.05-2018	09:00	Møte med veileder	Hovedbygg B313
18.05-2018	09:00	Møte med veileder	Hovedbygg B313

Tabell 10: Planlagte møter

7.2 Prosjektmøter

Siden vi kjører Scrum som arbeidsmetodikk holder vi daglig standup møte 08:15 hvor alle gruppemedlemmene deler informasjon, beskriver framgangen siden forrige møte, problemer som har oppstått og planlagt arbeid påfølgende dag.

På denne måten er alle gruppemedlemmene oppdatert på arbeidet og kan hjelpe hverandre for å løse enkle utfordringer på et tidlig tidspunkt før det eskaleres til et problem.

7.2 Periodiske rapporter

7.2.1 Framdriftsrapporter (inkl. milepæl)



Figur 9: Framdriftsplan

8 PLANLAGT AVVIKSBEHANDLING

Hvis vi ikke klarer å lage en fungerende hardwareenhet i tide, må vi vurdere å benytte oss av mulige ferdige løsninger. Vi må da finne en modul som er såpass åpen at vi enkelt kan hente ut den nødvendige informasjonen om lokasjon. Alternativt bruke en mobiltelefon som sporsenhet, slik at vi kan fokusere på software utviklingen.

Hvis vi ser at vi ikke får tid til å lage ferdig alle softwarekomponentene, så kan vi underveis vurdere om vi skal skalere ned prosjektet til å bare fokusere på enkelte deler, og droppe de delene som ikke er kritiske for systemet.

Ved feilestimert tidsforløp angående utviklingsprosessen er web fronten noe som kan settes som andreprioritert, samtidig som prosjektet bruker Android som primær GUI.

9 UTSTYR

For å kunne lage sporingsenheten trenger vi følgende:

- Mikrokontroller
- GPS modul
- GSM modul
- Mobilabonnement
- SIM-kort

10 REFERANSER

Sommerville, I. (2015). *Software Engineering*. Pearson.

Sutherland, J., & Schwaber, K. (2016). *The Definitive Guide to Scrum: The Rules of the Game*. Henta frå [scrumguides.org](http://www.scrumguides.org): <http://www.scrumguides.org/docs/scrumguide/v2016/2016-Scrum-GuideUS.pdf>

APPENDIX 2 – RETROSPECTIVE

2018-02-16 Retrospective

Date	16 Feb 2018
Participants	Kay Sindre Lorgen Anders Grytten Standal Kjetil Yndestad

Retrospective

What did we do well?

- Use of planning poker was a good technique to estimate story points
- Daily scrum

What should we have done better?

- Move issues to "In progress" and "Done" in JIRA
- More specific issues, now some of the issues was to "big"
- Some issues should be in the sprint because of dependencies

Figure 61: Retrospective - 2018.02.16

2018-03-02 Retrospective

Date	02 Mar 2018
Participants	Anders Grytten Standal Kay Sindre Lorgen Kjetil Yndestad

Retrospective

What did we do well?

- Effectively solved most issues with the microcontroller
- Almost completed all issues in sprint, in spite of absence from several team members
- Better at using Jira

What should we have done better?

- Remember to notify team members if you can't show up
- Write more consistently on thesis

Figure 62: Retrospective - 2018.03.02

2018-03-19 Retrospective

Date	19 Mar 2018
Participants	Kay Sindre Lorgen Anders Grytten Standal Kjetil Yndestad

Retrospective

What did we do well?

- Daily scrum
- Good report workflow

What should we have done better?

- Should have more precise issues
- Added sub-tasks after main issue was completed (should have been added as a new issue)
- Miscalculated story points

Figure 63: Retrospective - 2018.03.19

2018-04-03 Retrospective

Date	03 Apr 2018
Participants	Kay Sindre Lorgen Kjetil Yndestad Anders Grytten Standal

Retrospective

What did we do well?

- Charged our batteries during the Easter holidays

What should we have done better?

- Should finish the sprint on time
- Too many story points allocated
- Easter has too many vacation days
- Write report

Figure 64: Retrospective - 2018.04.03

2018-04-16 Retrospective

Date	16 Apr 2018
Participants	Kay Sindre Lorgen Anders Grytten Standal Kjetil Yndestad

Retrospective

What did we do well?

- Good progress until Thursday

What should we have done better?

- Remember to complete sprint.
- Because of sickness we could not add to many story points.
- .gitignore made problems, should work now.

Figure 65: Retrospective - 2018.04.16

2018-04-20 Retrospective

Date	20 Apr 2018
Participants	Kjetil Yndestad Kay Sindre Lorgen Anders Grytten Standal

Retrospective

What did we do well?

- Good progress on app development.
- Implemented security on Android.
- Good progress on estimated issues.
- Mapped out the road ahead.

What should we have done better?

- More focus on report. Write report consecutively.

Figure 66: Retrospective - 2018.04.20

2018-04-30 Retrospective

Date	30 Apr 2018
Participants	Anders Grytten Standal Kjetil Yndestad Kay Sindre Lorgen

Retrospective

What did we do well?

- Worked relatively productively
- Follow SCRUM process closely

What should we have done better?

- More focus on bug fixing
- Should have finished sprint on friday, instead of after the weekend

Figure 67: Retrospective - 2018.04.30

2018-01-12 Meeting notes

Date

12 Jan 2018

Attendees

- [Kjetil Yndestad](#)
- [Kay Sindre Lorgen](#)
- [Anders Grytten Standal](#)
- [Girts Strazdins](#)

Goals

- Preliminary meeting

Discussion items

Time	Item	Who	Notes
15	Misc		<ul style="list-style-type: none">• Loose discussion around the bachelor.

Action items

- Setup JIRA and Confluence [Kay Sindre Lorgen](#) 18 Jan 2018
- Research technology surrounding GPS and GSM tracking

2018-01-19 Meeting notes

Date

19 Jan 2018

Attendees

- [Kjetil Yndestad](#), [Kay Sindre Lorgen](#), [Anders Grytten Standal](#) and [Girts Strazdins](#)

Goals

- Select technology
 - Which hardware has the most promise?
 - First prototype?
- Specify group organization, product owner and SCRUM master.
- What should we focus on with the preliminary project?

Discussion items

Time	Item	Who	Notes
10min	Technology		<ul style="list-style-type: none"> • REST server (Begrunn) • Begrunnelse: Hvorfor rest? → Android og Web mulig. • Begrensning på hardware: • Eksterne bibilotek. • Prototype: Greit med stor arduino. • Finns det noen åpne systemer?

2018-01-26 Meeting notes

Date

26 Jan 2018

Attendees

- [Anders Grytten Standal](#)
- [Kay Sindre Lorgen](#)
- [Kjetil Yndestad](#)
- [Girts Strazdins](#)

Discussion items

Time	Item	Who	Notes
10min	Communication between arduino and server. Keeping packet size small over SSL		<ul style="list-style-type: none"> • UDP? • TLS SSL overhead på toppen av TCP. • 100bits pr pakke?,
5-10min	Our project compared to tractive, to similar? https://tractive.com/noeu_en		Apps på toppen som bruker tjenesten.

2018-02-02 Meeting notes

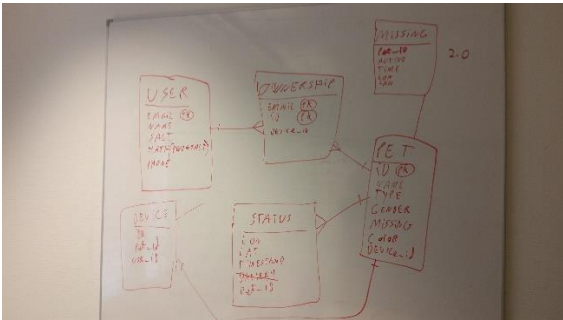
Date

02 Feb 2018

Attendees

- [Kay Sindre Lorgen](#)
- [Anders Grytten Standal](#)
- [Girts Strazdins](#)

Discussion items

Time	Item	Notes
5 min	Who will Judge the final report? Max Mørk said in a lecture that all english words should be in Norwegian so anyone could read it and understand it.	<ul style="list-style-type: none"> • Write English • Persons with technical experience
5 min	Should we write daily worklog?	<ul style="list-style-type: none"> • No need
10 min	JIRA structure	<ul style="list-style-type: none"> • Works good • Condition for when a task is done.
	<p>Database structure</p> 	<ul style="list-style-type: none"> • Can modify database later in development • Versioning • Don't make it more complex than necessary <ul style="list-style-type: none"> ○ Don't have several tables that aren't used in the finals system

2018-02-16 Meeting notes

Date

16 Feb 2018

Attendees

- [Kay Sindre Lorgen](#)
- [Anders Grytten Standal](#)
- [Kjetil Yndestad](#)
- [Girts Strazdins](#)

Discussion items

Time	Item	Notes
5min	Hardware status	<ul style="list-style-type: none"> ▪ Test GPS next week.
5min	Backend status	<ul style="list-style-type: none"> ▪ Autodeploy solutions (continuous integration) ▪ Database version systems, see Robert. Scheme version system. SQL version ▪ (Binding layer, retrieve result, convert to list.)
▪	Rapport	<ul style="list-style-type: none"> ▪ Send final preliminary rapport to Girts.z

2018-03-02 Meeting notes

Date

02 Mar 2018

Attendees

- [Kjetil Yndestad](#)
- [Girts Strazdins](#)
- [Anders Grytten Standal](#)
- [Kay Sindre Lorgen](#)

Goals

- Road ahead

Discussion items

Time	Item	Who	Notes
			<ul style="list-style-type: none">• Liquidbase
			<ul style="list-style-type: none">• Setup new sprint, start week after exam.• JWT json webtoken (one group last year). important https (letsencrypt)

2018-04-13 Meeting notes

Date

13 Apr 2018

Attendees

- [Kjetil Yndestad](#)
- [Girts Strazdins](#)
- [Anders Grytten Standal](#)
- [Kay Sindre Lorgen](#)

Discussion items

Time	Item	Who	Notes
	Generic update	Anders Grytten Standal Kay Sindre Lorgen Kjetil Yndestad	<ul style="list-style-type: none"> • Problem with git, mostly fixed. • Not SSL hardware, implement symmetric key. AIS broken? Try RC4, weakness decompile sourcecode, spoof pet location. Write in report could fix security with built-in SSL chip, and other security measures. • Third solution implement noise, binary key. • Challenges with spring implementation. Send message, send json which is encrypted, latitude, longitude and id. • Json webtoken. Hard to keep track of concerns. If somebody steals your mobile somebody can use the token from the phone. Last year, logout all users from phones.
	Next week		<p>Anders: Pretty much the same.</p> <p>Kay: Arduino security and app dev.</p> <p>Kjetil: Activitylog and Map.</p>
	Admin		Send bachelor draft to girts. (track changes, find solution on track changes)

2018-04-20 Meeting notes

Date

20 Apr 2018

Attendees

- [Kjetil Yndestad](#)
- [Anders Grytten Standal](#)
- [Kay Sindre Lorgen](#)
- [Girts Strazdins](#)

Discussion items

Time	Item	Who	Notes
	show and tell		<ul style="list-style-type: none"> • What is the wow factor • Example: Security in whole app (focus security in thesis), version control in DB, • Girts sees: SQL version, real security arduino as wow factor. • Jumper serial. Once connected send random key. LED serial. • APP: Geofencing, log when cat leaves house, and back again. • TODO: Estimate battery time, on production full product, duty cycle, warm up • Maybe business model? • Focus finished product, does not need to implement all functionality, show previous products, could not use closed API.
•	Report		Remember to also write the solutions that did not work.
	Next week		Microcontroller security.

2018-04-27 Meeting notes

Date

27 Apr 2018

Attendees

- [Kjetil Yndestad](#)
- [Kay Sindre Lorgen](#)
- [Anders Grytten Standal](#)
- [Girts Strazdins](#)

Goals

- How to make the report better?

Discussion items

Notes
<ul style="list-style-type: none"> • Comment, if method does thing that are not intuitive. • If can not be null. • Not intuitive boolean returns. • Algorithms.
<p>Next:</p> <ul style="list-style-type: none"> ▪ Implement geofence data backend ▪ Fix bug with images ▪ Bugfixing ▪ Geofence: Unit testing, more test, different direction, different lengths.
<p>Report:</p> <ul style="list-style-type: none"> ▪ Inversion of control: As a read I don't understand after I read it. IOC → If dependency is going the wrong way, can revert it with implementing an interface. Look at what is dependency injection, write something about it. ▪ A section describing micro services, backend, frontend, why is it useful. Stateless → makes it easier for backend, server does not need state to operate. ▪ Why have we chosen this solution.

2018-05-08 Meeting notes

Date

11 May 2018

Attendees

- [Anders Grytten Standal](#)

Discussion items

Time	Item	Who	Notes
5min	Do we need to write theory about solutions that didn't end up in the final product?		
	Where to put sections in the thesis? Should we write about Android-specific technology under theory, methods, or results?		

2018-05-11 Meeting notes

Date

11 May 2018

Goals

- The road ahead, what need to be done and advice on the report.

Discussion items

Time	Item	Who	Notes
5min	Where are we?	Anders Grytten Standal Kjetil Yndestad	<ul style="list-style-type: none"> • Thought about responsibility (async task, http client, etc) • Lombok → Write short about it (materials and method) • From system, point id (does not matter where it comes from). • Create some marketing point (why is it excellent points, innovation). • Market → Arduino remote device, security on device, analysed ssl not working, • Market → Safety and personal data, token, no critical data in DB, secure endpoints
10 min	Feedback	Girts Strazdins	<ul style="list-style-type: none"> • Report → Focus on what is done, specifically on security on the whole system. <ul style="list-style-type: none"> ◦ Detailed → Login and authorization ◦ No sql-injection ◦ Describe other alternatives ◦ Even though Robert has Liquibase, do describe liquibase indepth. • Presentation → Device → Packaged in separate device (not connected to computer) • Geofence → could have used websocket. • Maybe mention hackaton if relevant. (maybe possible to make very tiny devices, piggyback on disruptive technologies network) • Written good achitecture, componenets independent, change out and add app, web, tracking device, etc.

2018-05-19 Meeting notes

Date

22 May 2018

Attendees

- [Kjetil Yndestad](#)
- [Anders Grytten Standal](#)
- [Kay Sindre Lorgen](#)
- [Girts Strazdins](#)

Discussion items

Time	Item	Who	Notes
	Should we have some code implementation such as: <ul style="list-style-type: none">▪ HTTP client abstraction▪ Geofence Service▪ Distance calculation▪ Example of async task	Girts Strazdins	

2018-05-25 Meeting notes

Date

25 May 2018

Attendees

- [Anders Grytten Standal](#)
- [Girts Strazdins](#)

Discussion items

Time	Item	Notes
	Notes on thesis	<ul style="list-style-type: none"> • Generally don't say "we didn't have time" when explaining why we did not implement some functionality • Where to write about technical problems and how we solved it <ul style="list-style-type: none"> ◦ Can create a new section on Discussion
	Presentation	<ul style="list-style-type: none"> • Should not spend more than 5 minutes on introduction <ul style="list-style-type: none"> ◦ Who we are, what the project is, what the goal is, etc • Can talk about technical details, but focus should be on results <ul style="list-style-type: none"> ◦ Can mention technical challenges, architecture, how we solved certain problems, etc

2018-05-29 Meeting notes

Date

30 May 2018

Attendees

- [Kjetil Yndestad](#)
- [Kay Sindre Lorgen](#)
- [Anders Grytten Standal](#)
- [Girts Strazdins](#)

Goals

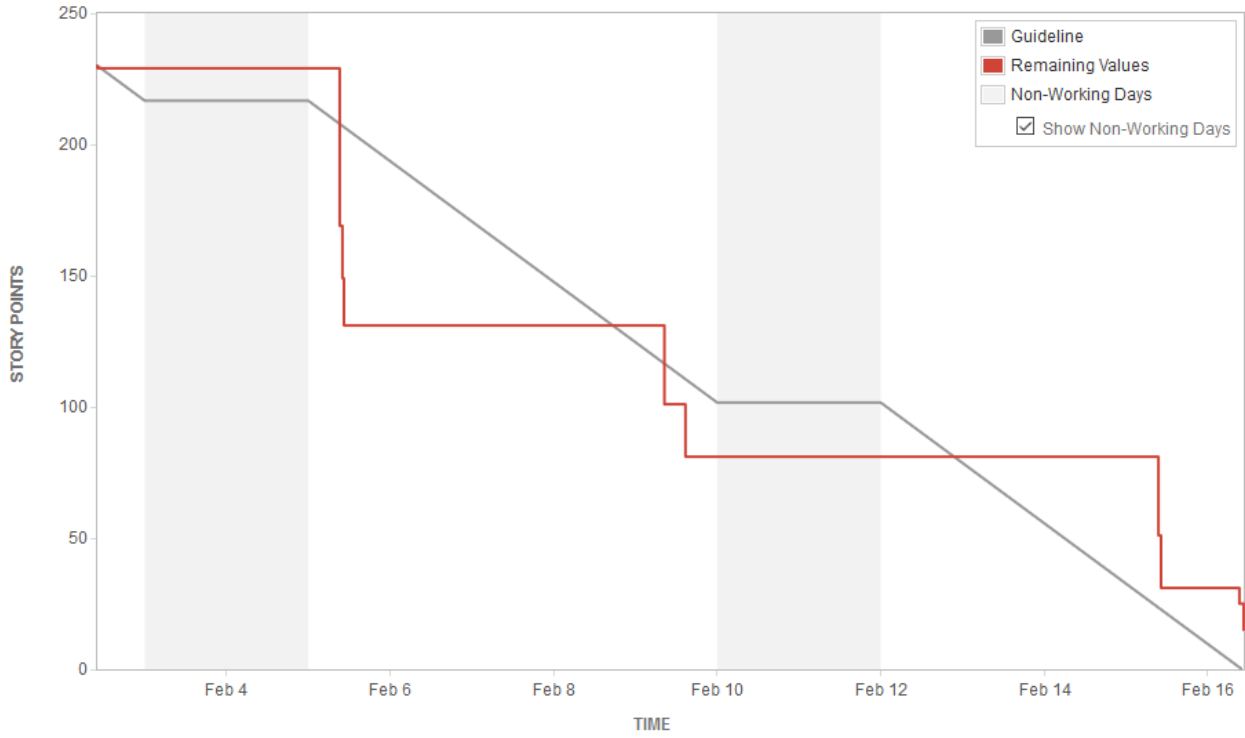
- Final meeting, clarify last uncertainties.

Discussion items

Time	Item	Who	Notes
5-10min	Battery calculations	Kjetil	<ul style="list-style-type: none"> • Rough estimate
5min	Package size	Kay	<ul style="list-style-type: none"> • ok
	Which appendixes?		Have: <ul style="list-style-type: none"> • Meeting notes • Preliminary Report • Retrospectives Maybe: <ul style="list-style-type: none"> • Burndown charts? • other?
	Report		<ul style="list-style-type: none"> • Candidate number or Student number and name? • Include word count? Neh, MC: Duty cycle, persistent connection, clock speed, etc.

APPENDIX 4 – BURNDOWN CHARTS

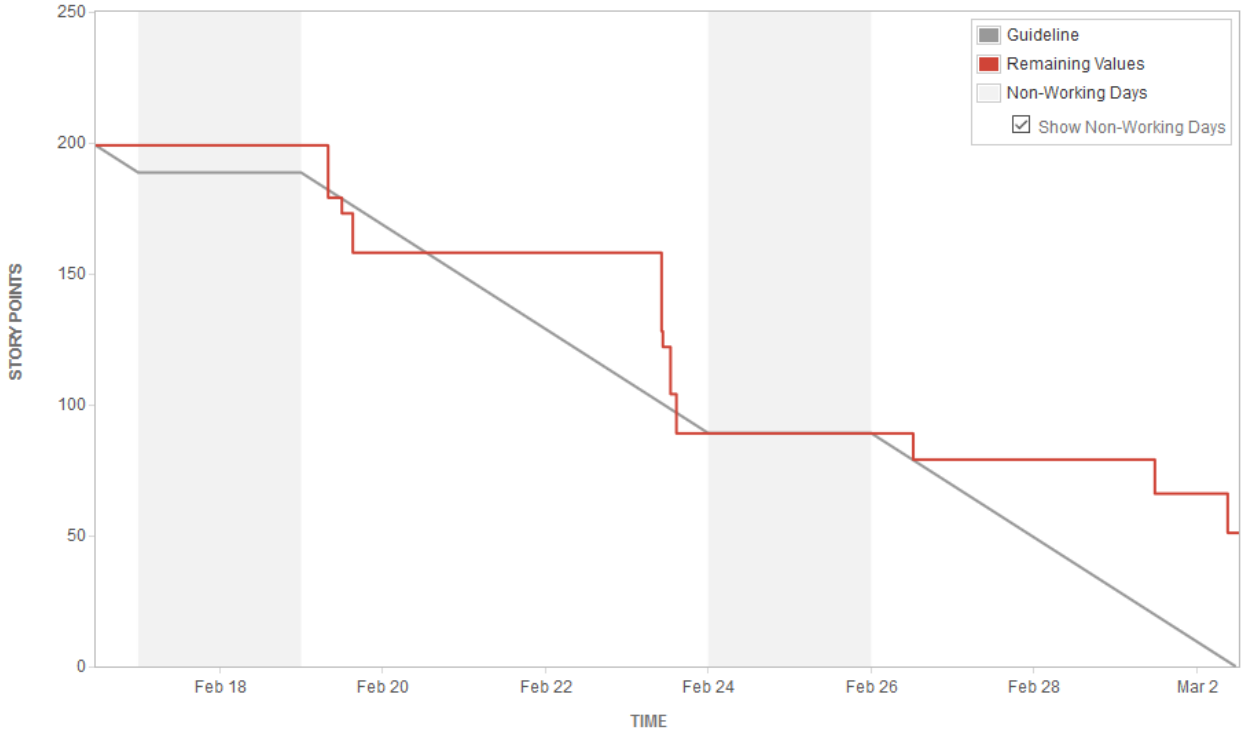
Sprint 1



Burndown Chart 1: Sprint 1 (week 6-7)

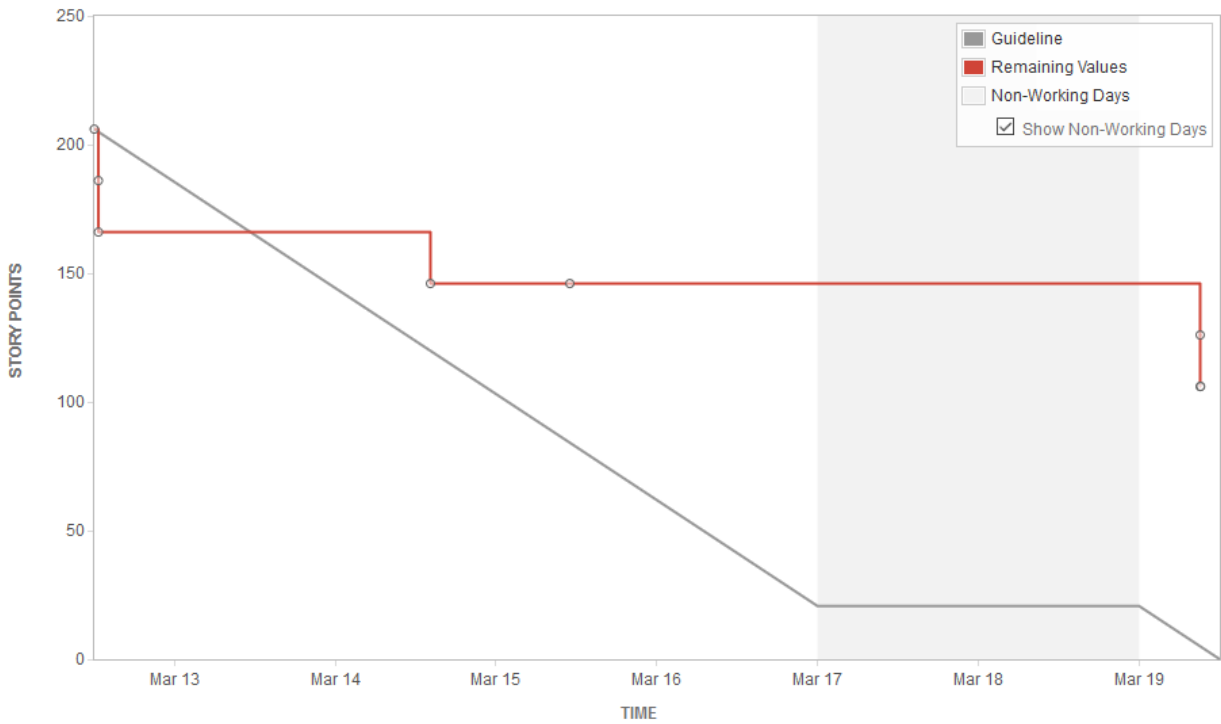
Sprint 2

APPENDIX 4 – BURNDOWN CHARTS



Burndown Chart 2: Sprint 2 (week 8-9)

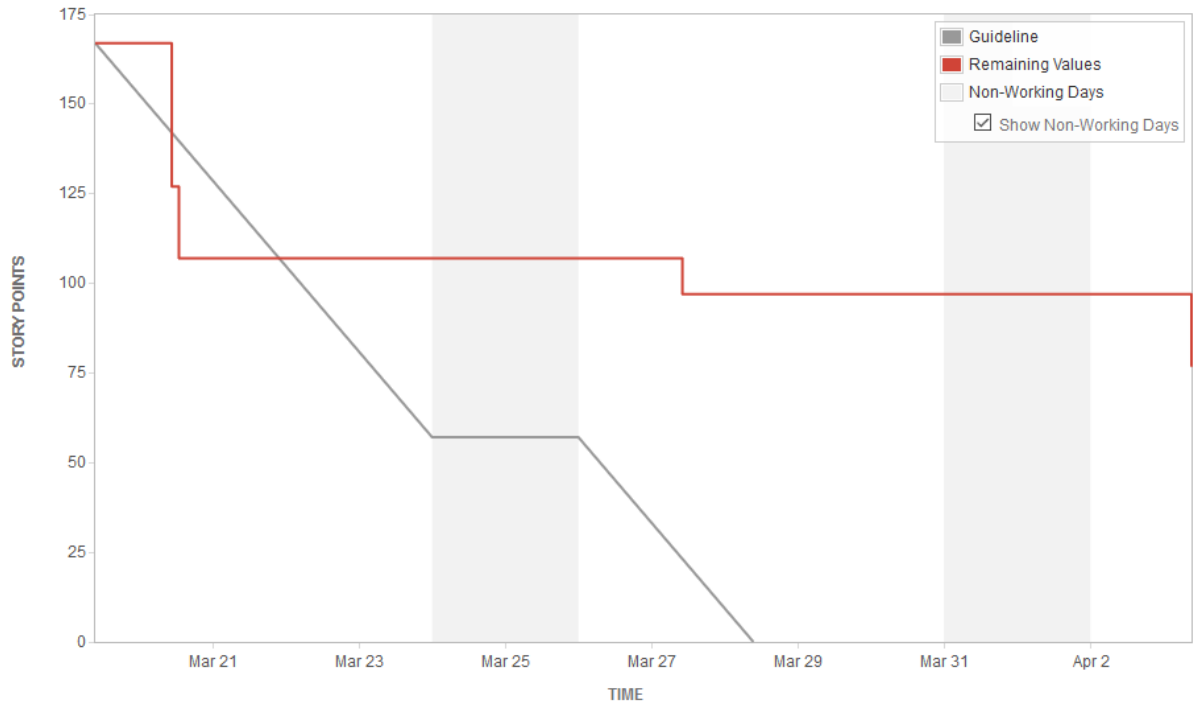
Sprint 3



Burndown Chart 3: Sprint 3 (week 11)

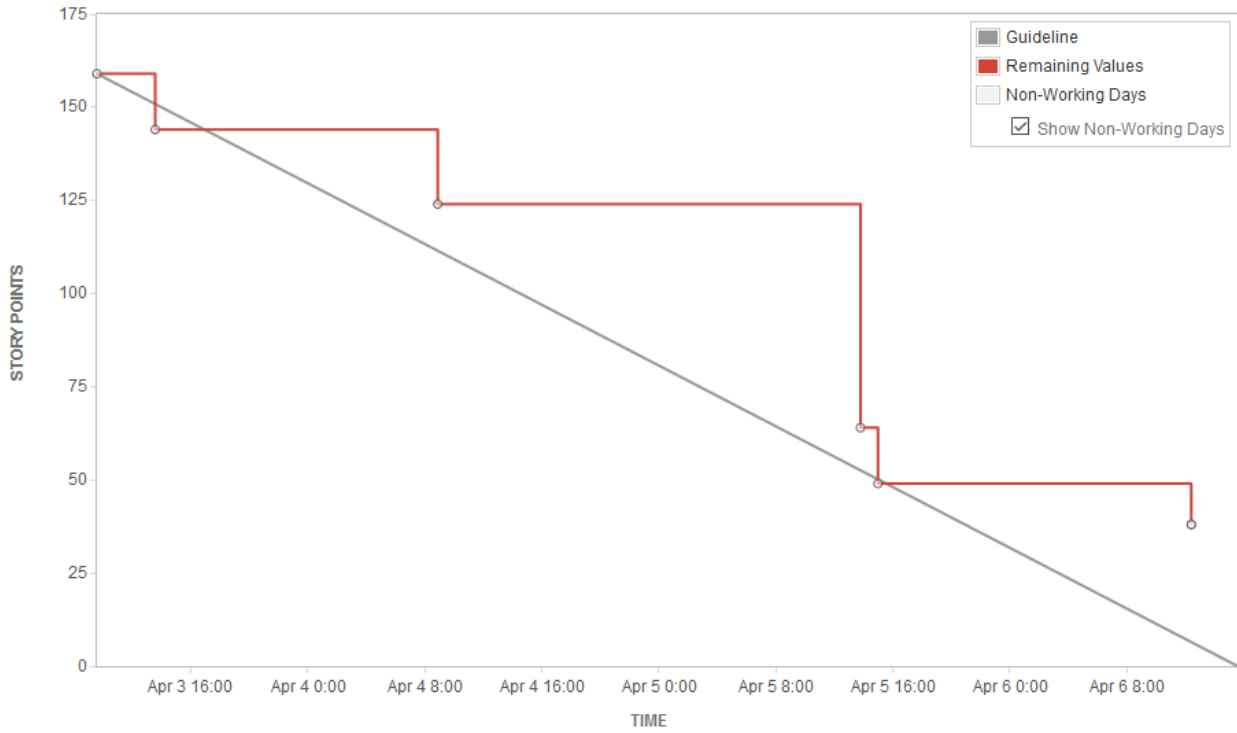
Sprint 4

APPENDIX 4 – BURNDOWN CHARTS



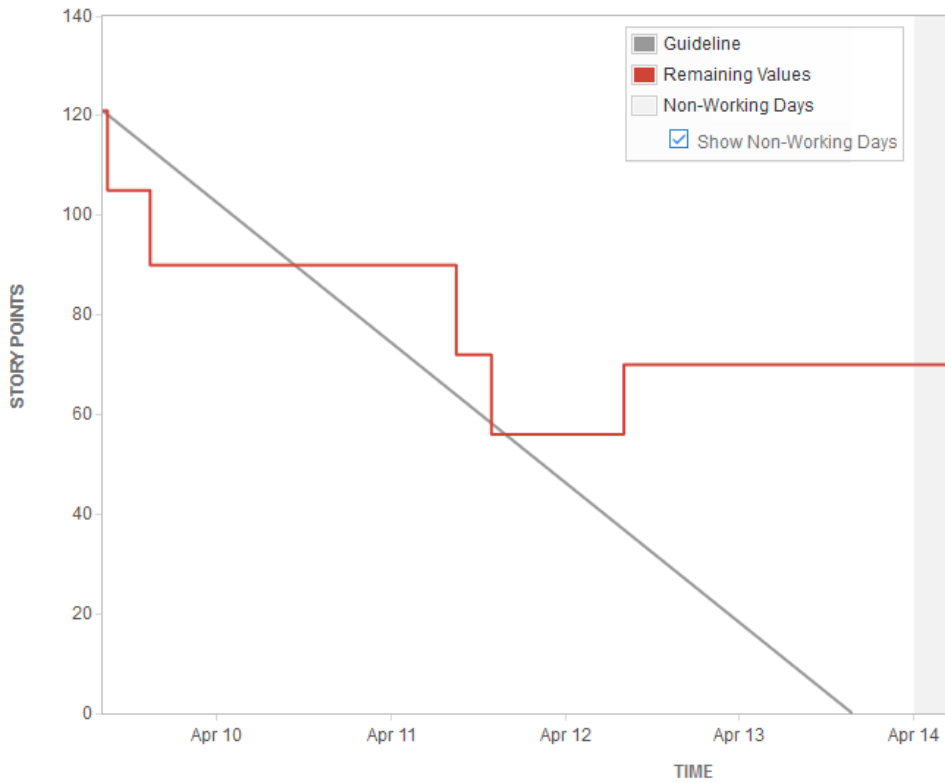
Burndown Chart 4: Sprint 4 (week 12-13)

Sprint 5



Burndown Chart 5: Sprint 5 (week 14)

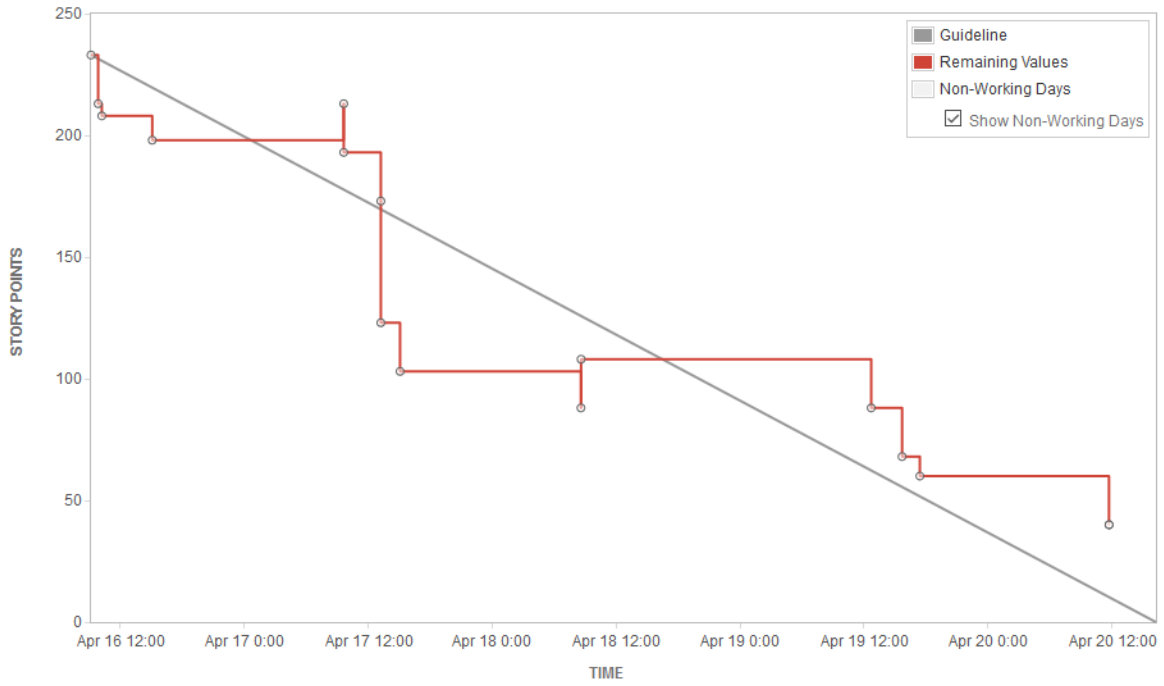
Sprint 6



Burndown Chart 6: Sprint 6 (week 15)

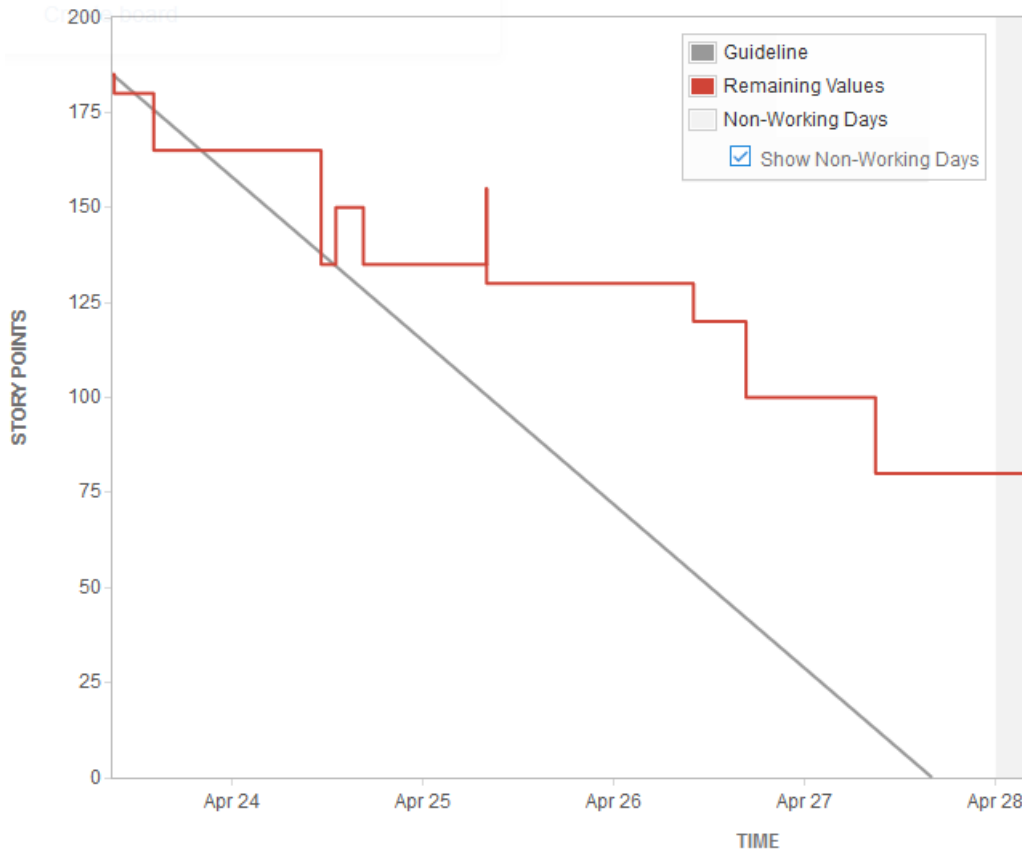
APPENDIX 4 – BURNDOWN CHARTS

Sprint 7



Burndown Chart 7: Sprint 7 (week 16)

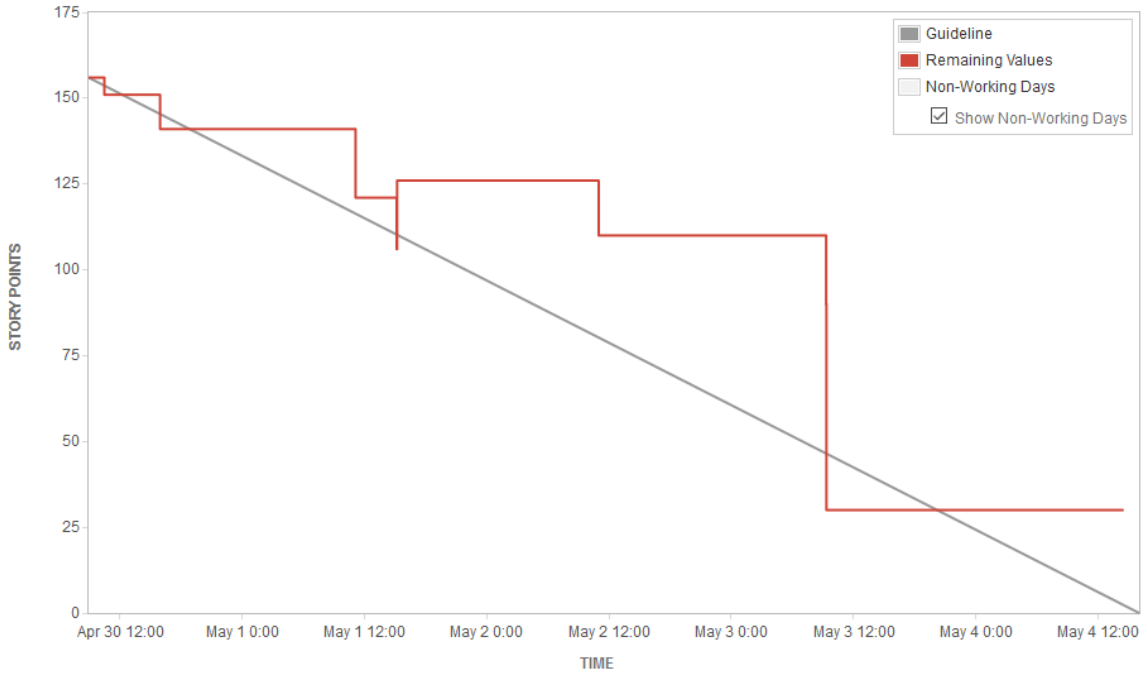
Sprint 8



Burndown Chart 8: Sprint 8 (week 17)

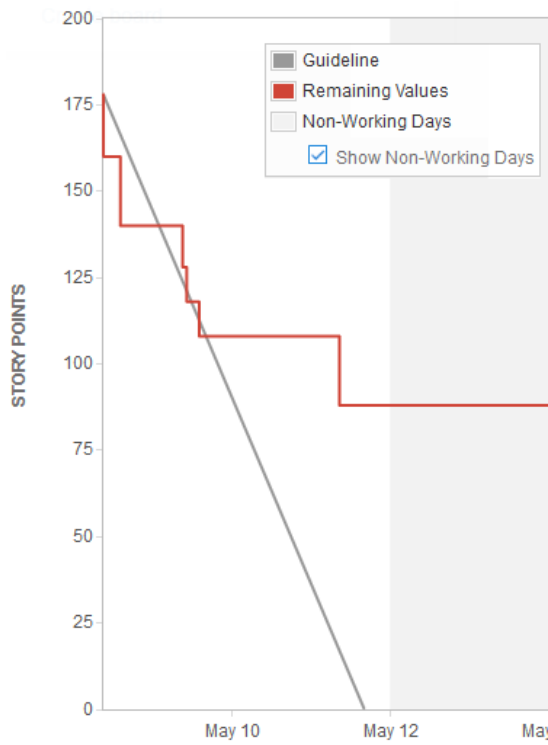
APPENDIX 4 – BURNDOWN CHARTS

Sprint 9



Burndown Chart 9: Sprint 9 (week 18)

Sprint 10



Burndown Chart 10: Sprint 10 (week 19) [forgot to close sprint]