Faculty of Science and Technology
Department of Computer Science

Láhttu

A system for Retrieval and Consolidation of Personsal Data from Activity-Tracking Web Services

—

Ida Jaklin Johansen

INF-3981 Master's Thesis in Computer Science, June 2014

Abstract

In recent years, self-tracking and recording ourself has become increasingly popular. A large ecosystem of interconnected online activity-tracking web services that record, store, analyse, and visualize personal data is evolving to provide useful services to end-users. However, these personal data can be scattered over multiple web-services, which makes it difficult for an individual to manage and maintain an overveiw of activity levels.

This thesis identifies requirements, designs, and develops a system for connectioning to a set of activity-tracking web-services. The system retreives personal data from these activity-tracking web-services for end-user, and presents and consolidates personal data stored on these web-services. The main goal for the system is to provide a homogenous, presentation and improve insight for the end-users into their own activity tracking personal data recorded at hetergoenous web-services.

The system is evaluated from a proof of concept veiw point.

Acknowledgements

I would like to thank my supervisor Dr.Åge Andre Kvalnes for being my supervisor and for great feedback. Additionally, great thanks to Dr.Håvard Dagenborg Johansen, for discussions and input during this thesis periode. In addition, I thank the rest of my colleagues at the iAD group for valuable input and discussions.

Furthermore, I would like to thank all my fellow classmates, or “my boys”: Simon, Jan-Ove, Magnus, Einar, Bjørn, Tom, Alexander and Steffen for awesome years together. Especially, thanks to Simen Lomås Johannessen, for being the greatest friend one could ever get.

Finally, I would like to thank my friends and family for their support and believing in me. Special thanks to my Father, for being a true source of inspiration and inspired me to pursuit the path of Computer Science. Maria Wulff Hauglann for being a “big sister” to me all these years at IFI and to my little sister, Lisa, your dedication is admirable. Last but not least, my boyfriend, for always being so supportive and with a kindness out of this world. I love you.

Contents

Abstract iii

Acknowledgements v

List of Figures xi

List of Tables xiii

List of Abbreviations xv

1 Introduction 1
- 1.1 Problem Definition 2
- 1.2 Methodology 2
- 1.3 Interpretation, Scope, and Limitations 3
- 1.4 Context 4
- 1.5 Terminology 5
- 1.6 Outline 5

2 Background 7
- 2.1 Health Tracking 7
  - 2.1.1 Fitbit 9
  - 2.1.2 ZXY Sport Tracking 9
  - 2.1.3 RunKeeper 11
  - 2.1.4 “Internet of Things” (IoT) 11
- 2.2 Personal Data Vaults 12
- 2.3 Authentication with OAuth 13
- 2.4 Access Control, Data Management, and Storage 14
  - 2.4.1 Database 15
  - 2.4.2 Cache and Caching Algorithms 15
  - 2.4.3 "Big Data" 16
  - 2.4.4 RESTful API 17
- 2.5 Data Integration and Interoperability Survey 18
  - 2.5.1 Data Consistency Issues 20
- 2.6 Summary 22

3 Requirement Specification 25
3.1 System Functional Overview 25
3.1.1 Frontend 26
3.1.2 Backend 27
3.1.3 Web-Services 27
3.2 Non-functional requirements 28
3.2.1 Security and Privacy 28
3.2.2 Reliability and Availability 28
3.2.3 Extensibility 28
3.2.4 Scalability 29
3.2.5 Fault-tolerance 29
3.2.6 Dependency 29
3.2.7 Interoperability 30
3.2.8 Maintainability 30
3.2.9 Usability 30
3.2.10 Performance 30
3.3 Summary 31

4 Design 33
4.1 System Architecture 33
4.1.1 Frontend 33
4.1.2 Backend API 34
4.1.3 Access Control 35
4.1.4 Storage 36
4.1.5 Crawling 37
4.2 Web Services and their APIs 38
4.2.1 Fitbit 39
4.2.2 RunKeeper 40
4.3 Summary 40

5 Láhttu 41
5.1 Frontend 41
5.2 Backend 47
5.2.1 Time, Date, and Range Query 49
5.2.2 Storage 50
5.2.3 Crawler 51
5.2.4 Web-services API 52
5.2.5 Access Control 55
5.3 Summary 57

6 Evaluation and Results 59
6.1 Methodology and Methods 59
6.2 Proof of Concept 60
6.3 Experiments 61

6.3.1 Experimental Setup 61
6.3.2 Estimation of requests 62
6.3.3 Latency 63
6.4 Evaluation of non-functional requirements 65
6.4.1 Scalability and Extensibility 66
6.4.2 Fault Tolerance and Availability 66
6.4.3 Security and Privacy 67
6.5 Summary 67

7 Related Work and Discussion 69
7.1 Related Work 69
7.1.1 Microsoft HealthVault 69
7.1.2 Consolidating Personal Data Platforms and Systems . 70

8 Conclusion 73
8.1 Achievements 73
8.2 Concluding Remarks 74
8.3 Future Work 74

A JSON Reponse from Web-services 77
A.1 RunKeeper 77
A.2 Fitbit Step 78
A.3 Fitbit Sleep 83

Bibliography 89

List of Figures

2.1 The Fitbit Flex 9
2.2 ZXY sensor belt [15] 10
2.3 ZXY radio receiver on an antenna at Alfheim Stadium [15] . 10
2.4 Overview of the ZXY Positioning Sensors [15] 11
2.5 Connected devices in the Internet of Things (figure from [7]) 12
2.6 Memory hierarchy 15
2.7 Eco-system of tracking system and devices. 18
2.8 Control over applications connected to a Fitbit end-user account 19
2.9 Connection of Application, synchronous data between 19
2.10 Fitness Feed in an end-user account at RunKeeper, the end-user are anonymously 20
2.11 Fitbit recording for one day, automatically synchronized with Endomondo and stored there. 21
2.12 A record run in RunKeeper 21
2.13 Importing the RunKeeper record into Strava 22
2.14 Importing the RunKeeper record into Endomondo 23
2.15 Importing the RunKeeper record into SportTrackLive 23

3.1 The System Model,Abstract Architecture 27

4.1 System Architecture 35

5.1 Screenshot of Láhttu login page with the RunKeeper connection button 42
5.2 Screenshot of the option for the end-user for RunKeeper . . . 43
5.3 Flow over the login process 43
5.4 Screenshot of the option for the end-user for Fitbit 44
5.5 Screenshot of the main page of Láhttu 45
5.6 Screenshot of the main page of Láhttu after an Range Query for 4th of March 45
5.7 Screenshot of the timeline with personal data in the range of 4th to 5th of March 47
5.8 Request Range Query 49
5.9 Response Range Query 53

5.10 Give access 56

6.1 The prototype paradigm. 60
6.2 Timeline of the personal data retrieve in the Appendix A . . 61
6.3 Closer look at the timeline in Figure 6.2 61
6.4 Latency figure 63
6.5 Latency Chart for table 6.2 and table 6.3 65

List of Tables

4.1 Current Data Sources for the system 39

5.1 Language at Frontend. 47
5.2 JavaScript Libraries and features. 47
5.3 HTTP methods 48
5.4 Python Libraries. 48
5.5 Register an application at Fitbit, Required Information 52
5.6 Register an application at RunKeeper, Required Information . 53
5.7 Web services represent data. 55

6.1 Correlation between number of end-user and required number of API requests. 62
6.2 The end-to-end latency from the Fitbit Web-serivce. 64
6.3 The end-to-end latency from cache. 64
6.4 Seconds for acquisition personal data from the web service . 65
6.5 Seconds for acquisition personal data from the cache 66

List of Abbreviations

ACL	Access Control List
AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
BMI	Body Mass Index
CSS	Cascaading Style Sheets
FIFO	First In First Out
GPS	Global Positioning System
GPX	the GPS Exchange Format
GUI	Graphical User Interface
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IOT	Internet of Things
JSON	JavaScript Object Notation
LRU	Least Recent Used
PDV	Personal Data Vault
PHR	Personal Health Record
RDBMS	Relational Database Management System

REST Representational State Transfer

RPE Rating of Perceived Exertion

SQL Structured Query Language

TIL Tromsø Idrettslag

UI User Interface

URI Uniform Resource Identifer

URL Universal Resource Location

UX User Experience

WHO World Health Organization

XML Extensible Markup Language

ZXY ZXY Sport Tracking System

1 Introduction

Recent advances in sensors and portable technologies have enabled ordinary people to keep track of their daily activities in a profoundly new and detailed manner. Through self-tracking, end-users might achieve self-awareness and knowledge about themselves. This has received significant attention in the consumer marked. In particular, self-tracking is changing how amateur and professional athletes train and live [9]. As a response, many consumer level devices for self-tracking, such as Fitbit and Jawbone, utilize accelerometers in the devices for tracking movements. Activity-tracking applications, such as RunKeeper and Endomondo utilize the Global Positioning System (GPS) in smart phones for tracking position and routes.

One of the key benefits of such self-tracking is to enhance the ability of individuals to keep an eye on their health parameters and fitness levels. This in order to detect emerging health problems early and to foster physical activities. The ability to foster activity by tracking personal fitness levels is in particularly becoming important in the modern society. Since 1980, the number of people with obesity has doubled. More than 1.4 billion adults are overweight, with 500 million of them being obese. This is about 11% of the world's population over the age of 20. Obesity can impair ones health due to abnormal or excessive amount of body fat. World Health Organization (WHO) defines overweight as a person with Body Mass Index (BMI) greater than or equal to 25, while anything above 30 is considered obese [31]. In Norway, one of five is overweight and around 100.000 obese. [1] Indicators show that these numbers will increase in coming years. According to the Norwegian Directorate of Health in the report, "Kunnskapsgrunnlag fysisk aktiviet"[28], they state and estimate that inactive or insufficient activity level amongst Norwegians, will cost Norway 239 billion NOK kroner per year. Simultaneously, science and technology are focusing more on health, wellness, and fitness to overcome the obesity epidemic in the western part of the world. Personal data can also be used in larger big-data medical studies by having people pool their activity data into larger research projects [40].

1.1 Problem Definition

A large ecosystem of interconnected online activity-tracking web services that store, analyse, and visualize health and activity data is evolving for collecting and analyzing personal activity data for the consumer marked. Unfortunately, little has been done to standardize data exchange and data formats between these services and we have ended in a situation where our personal health and fitness data has become scattered over multiple different, heterogeneous systems. This makes it difficult to see the big picture of their health and activity, diminishing the purpose of self-tracking [5, 6].

> *This thesis will explore system issues related to the use of personal data from activity tracking web-services. The goal is to architect and build a prototype system that provides end-users an overview of and improve insight into their online personal data.*

The system should be evaluated with focus on a *proof of concept* system that addresses the stated problem.

1.2 Methodology

The final report of the ACM Task Force on the Core of Computer Science divides computing as a discipline into three major paradigms [4]:

Theory: Rooted in mathematics, the approach is to define a problem, propose theorems and try to prove that the relationships are true, in order to determine and interpret the result.

1. http://www.vg.no/nyheter/innenriks/artikkel.php?artid=10122067

Abstraction: Rooted in the experimental scientific method, the approach is to investigate a phenomenon by forming a hypothesis, construct a model, and make a prediction. Collecting data and experimenting on this data, finally interpret the results.

Design: Rooted in engineering, the approach is to construct a system or device to solve a defined problem by stating the requirements and specifications. Design and implement the system or device. Finally, testing and evaluation of the system is done depending on the requirements and specifications.

This thesis largely adheres to the design paradigm. Given a problem, construct the prototype system by stating the requirements and specifications. A prototype system will be designed, implemented, and evaluated.

1.3 Interpretation, Scope, and Limitations

The motivation for developing this system is to give end-users increased insight in their own tracked personal data and gain access control over whom can access personal data. The main focus will therefore be on making a homogeneous system for end-users with data from a small selection of heterogeneous, online web services. We will therefore focus on integrating with at least Fitbit and RunKeeper services.

The thesis will resolve the stated problem through designing and implementing a prototype system with all the component for accomplishing the goals which are stated. The system will include components for acquisition of personal end-user data from heterogeneous web-services, storage management through database usage and Graphical User Interface (GUI) to present and get input from the end-users of the system. The User Experience (UX) is not the main focus for this thesis, but focusing on providing an intuitive experience for end-users.

Additionally, this thesis will need to investigate the eco-system of heterogeneous web-service. In particular, what type of personal data are they recording and tracking and how is this personal data represented. The limitations and restrictions of these web-services and how this can and may impact the development of this thesis, is also a concern.

Evaluation of the system will be conducted with a focus on functional and non-functional system properties. A goal is to prove that concepts adhere to specifications and stated limitations.

Primarily the limitations of the system will depend on the Application Programming Interface (API) of the web services. For example, Fitbits state: "If your Developer Application causes technical stress to the Fitbit platform, Fitbit will disable your access."

There are some limitations in making the system. When working with a third-party API, that API sets some boundaries for what a developer can do and cannot do. For example, there are limitations as to what can be read and written from the web-services.

There are some features that are out of scope for this thesis. These features can be added to potential future work. How web-services record and track the personal data and how accurate this personal data is, beyond the insight one have to the web-services. To limit the engineering scope of this thesis, we will only consider single-user scenarios in the evaluation. Because of limitation in the evaluation practice of this thesis, through testing with multiple end-users simultaneously, there will be no requirements to multiple concurrent end-users.

In addition, security is necessity; however this is not the main focus of the thesis and will be eventual future work.

Finally, one shall also investigate how these self-tracking personal data could be used in a bigger context, in form of core and case study for public health studies.

1.4 Context

This project is written as a part of the Information Access Disruption(iAD) centre. The iAD centre targets research into fundamental concepts and structures for large-scale information access. The main focus areas are technologies related to sport, analytic runtimes, and cloud computing.

Previous projects developed at iAD are Muithu[19] and Bagadus[12][37]. Muithu is a sports notational analysis system for video, developed by the iAD department at the University of Tromsø and in partnership with Tromsø Idrettslag (TIL). Bagadus is a player tracking system that uses ZXY Sport Tracking System (ZXY) and a video camera array. The system tracks individual soccer players and computes statistics by combining captured video footage and data from ZXY. The paper[15], addresses the different system developed by iAD that are used at Alfheim stadium. In addition, the paper addresses how big-data analytic can improve performance in the soccer area.

Girji[20]is a system for performing big-data analytic in the consent of preserving control access to the end-users personal data.

1.5 Terminology

Important terminology used in this thesis include:

End-User: A end-user is a person that using the system.

Component: The system is divided into several entity with their own functionality. Each entity of the system is referred to a component.

System: The prototype system that is design and implemented, later on given a specific name to be refereed to.

She: May represent a given end-user in the context of a situation.

Frontend: The side the end-user interact with, may be referred to as the client side.

Backend: Computation side, may be referred to as the server side.

1.6 Outline

The thesis is structured as follows:

Chapter 2 This chapter presents relevant technical background information for the thesis. Also, a survey around the eco-system of web-services.

Chapter 3 This chapter describes the requirement specification including the general system model with functional and non-functional requirements.

Chapter 4 This chapter describes the architecture and design.

Chapter 5 This chapter describes implementation details for the system.

Chapter 6 This chapter presents evaluation and results. Including reflection of the system through discussion.

Chapter 7 This chapter presents related work in the context of the system.

Chapter 8 This chapter presents the conclusion and potential future work for the thesis.

2 Background

Self-tracking has been popularized recently in the context of the *"Quantified self (QS)"*, a term that was first coined in 2007 by Gary Wolf and Kevin Kelly in San Francisco. Since then, the Quantified self has become an international collaboration movement for self-tracking tools, both for users and developers, which is rapidly growing[1]. Through self-tracking everyday movements, activity, food and water intake, the end-users are providing self-knowledge about themselves and their own health [39]. The Quantified Self movement holds annual meetings and conferences are held throughout the world. In 2013 and 2014, there was a Quantified Self Europe Conference in Amsterdam. In addition, there are local Quantified Self meet ups all over the world.

This chapter presents the central technical background material related to the large number of technological advances and issues emerging from online self-tracking and relation to this thesis.

2.1 Health Tracking

Health is in the context of humans the general condition of a persons mind and body and how healthy or unhealthy these are. Being healthy or having good health is to be exempt from pain, illness or injury, entirely both physically and

1. http://quantifiedself.com/about/

mentally [11]. There are various factors that impact human health, nutrition and diet so that a human is getting the right amount of protein, carbohydrates and fat. With a balanced diet one are getting the vital substances: amino acids, vitamins and minerals that helps build up a good immune system. The immune system is the body defence from diseases and illness. Eating an unbalanced diet and too much sugar and fat can lead to lifestyle diseases and obesity.

Another main factor on health is sleep. Sleep is something that has baffled scientist for centuries and they still have no definitive answer to why we need it. What they do know is that sleep is essential and a requirement for survival. It is essential for our brain's ability to function properly, especially to maintain our cognitive skills such as speech, memory and flexible thinking. Studies have shown that sleep deprivation can affect not only ones cognitive skills, but have an impact on emotional and physical health. There is no exact amount of sleep required for humans, as it varies from person to person. The record for longest time without sleep is 11 days[2]. Sleep is divided into two categories: non-REM and REM sleep. Non-REM sleep is further split into four stages: Light sleep is the first stage and is the one where one feels like one is half asleep and could easily be awakened. After 10 minutes one enters the second stage, true sleep, which lasts around 20 minutes and is where the heart and breathing slow down. Stage three, deep sleep, is when the breathing and heart rate is at its lowest and the brain begins to produce delta waves. Stage four is also called deep sleep, and is where one has a rhythmic breathing and limited muscle activity. After the non-REM stages, one enters the REM sleep, which stands for rapid eye movement, simply because our eyes move rapidly at this point. It starts after 70 to 90 minutes, and at this stage, our brain is very active, often more than when we are awake. This is also the stage when most dreams occur, our blood pressure rises, but where our body is effectively paralysed. After the REM sleep, the whole cycle starts over again. [3]

Physical fitness is a state of health that defines the ability to perform a sport, activity or everyday life assignments. Studies have shown that everyday activity and walking can improve creative thinking [30], one of many health benefits from being physically active. Abstaining from obesity through being physically active. To track parameters related to physical fitness, activities, and sleep, a large number of wearable technology and self-tracking web-services have emerged. We will give a few examples in the following sections.

2. https://science.education.nih.gov/supplements/nih3/sleep/intro/getting-started.htm
3. http://healthysleep.med.harvard.edu/healthy/science/what

2.1.1 Fitbit

Fitbit Inc. is a company which produces wearable activity tracker devices. The Fitbit devices measure personal data such as number of steps walked, distance walked or run, very active minutes, calories burned per day, and duration/quality of sleep. Fitbit offers an application and web interface for the end-users account.[4] The end-users can record and log their food and water intake, weight, and personal goals; this could be weight lost, drinking more water or having a more active day.

The application on a smart-phone synchronizes data between the device and the end-users online account. For instance, the Bluetooth enabled Fitbit Flex, as illustrated in Figure 2.1, synchronizes the data recorded on the wristband when it is in range of either the communication dongle inserted into a computer that has the Fitbit Connect software running or with a mobile device that has been paired to the given device. The end-users can interact with friends, comparing who is the most active one. It is also possible to create and organize groups, setting common goals and competing against each other.

Figure 2.1: The Fitbit Flex

Fitbit Inc. also produces a scale, Fitbit Aria, for tracking the weight of up to eight persons, their body fat percentage and BMI. This personal data is wirelessly synchronized with the end-users account at Fitbit. Fitbit offers an open API for developers to make third-party applications. This allows developers to access and utilize Fitbit data in their own third-party applications.

2.1.2 ZXY Sport Tracking

ZXY is a stationary radio-based system developed by a Norwegian-based company for recording telemetry from players in soccer matches. ZXY is used by

4. http://www.fitbit.com/

several soccer teams in Tippeligaen and Addecoligaen, the Norwegian elite series for soccer and the secondary level. Among these teams are Tromsø IL in Tromsø and Rosenborg BK in Trondheim.

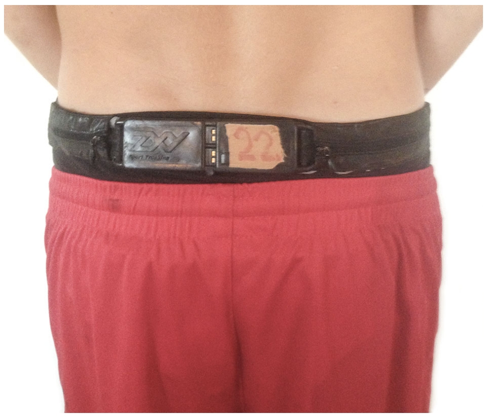


Figure 2.2: ZXY sensor belt [15]

A sensor belt is placed on the waist of all the soccer players, and on the belt is a sport chip for measuring and sending data.

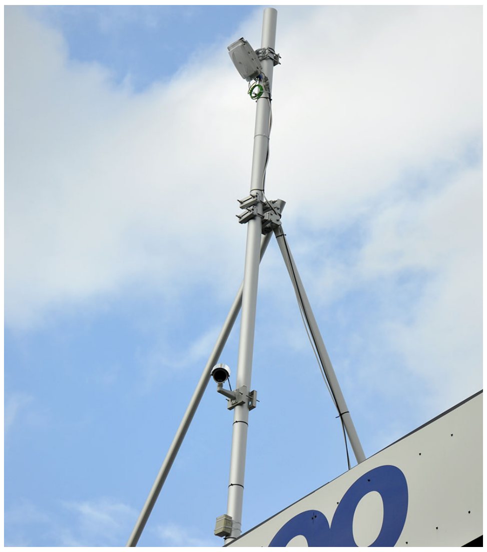


Figure 2.3: ZXY radio receiver on an antenna at Alfheim Stadium [15]

There are radio receivers placed around the stadium for receiving information from the soccer players, sampling data up to 20 times per second. This information is stored in a SQL Anywere database on a server. The data generated from ZXY are telemetry like position, acceleration, playtime, run distance, pulse and more from a soccer match[15]. It has been proven that ZXY is accurate for recording and tracking [15]. Hence, one can assert that the ZXY system is reliable.

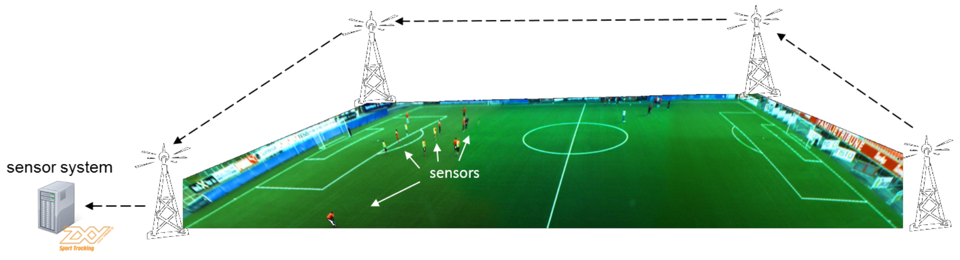


Figure 2.4: Overview of the ZXY Positioning Sensors [15]

2.1.3 RunKeeper

RunKeeper is a fitness-tracking application with more than 26.2 million end-users as of the 26th of November 2013 for both iOS and Android. RunKeeper was launched in 2008. End-users track their walking, running, cycling, hiking, biking and other activities using the GPS in their smart phones, turning the device into their own personal trainer.

RunKeeper tracks performance over time, allowing end-users to see statistics and detailed history of their activities and consequent progression. One can also share these activities by posting them on Twitter and Facebook.

RunKeeper offers an open API for third-party developers to plug into RunKeeper user feeds, making a community of applications for RunKeeper [24].

The API that RunKeeper uses for generating a cloud of health and fitness applications is the *HealthGraph*[16]. The HealthGraph is a digital map of a persons health, with health data as either interrelations or connections. HealthGraph snapshots ones current physical condition, as well as maintaining a health history and how it has evolved over time [24].

2.1.4 "Internet of Things" (IoT)

Internet of Things (IOT) is things or objects that are connected to the Internet. Although most familiar are devices such as laptops, servers, smartphones and tablets, are the concept of IOT in a much larger scale. IOT devices can be wearable devices, alarms, sensors, home appliances such as television, remote controller, lamps and vacuum cleaners as illustrated in Figure 2.5. In 2008, the number of devices connected to the internet was greater than the number of people in the world who were using the Internet [41].

Figure 2.5: Connected devices in the Internet of Things (figure from [7])

It is estimated that the IOT is to reach 50 billion devices before 2020[41]. To illustrate the exponential growth in things connected to the Internet, Cisco has created a counter to track the number of IOT[1].

2.2 Personal Data Vaults

In recent years, technology and software that focus on health which can be defined as eHealth (Electronic health).[5] have increased. mHealth (Mobile health) is mobile devices that are used for supporting the practice of health and medicine, such as communication, data exchanging and reports. These mobile devices include everything from smart phones, tablets and laptops, also including IOT devices.

Ohmage [34] is system for acquiring end-user data for recording and analyzing. An Ohmage implementation can acquire Rating of Perceived Exertion (RPE) from a cellular, which is a feedback form in a scale for measuring perceived exertion. During a test or exercise a sport coach and athlete can rate the current physical health and wellness of the athlete.

Microsoft has a platform for health and fitness information for storing and maintaining personal data, Microsoft Health Vault.[6] Every end-user has a Health-

5. http://www.who.int/topics/ehealth/en/
6. https://www.healthvault.com/

Vault account with their individual health information stored. Access control can be adjusted so that a mother may have access to her childrens account or other relatives. HealthVault have support and functionality to let the end-users connect medical devices and application to their HealthVault account. Worth mentioning is that Google had a similar project, Google Health, that has been permanently discontinued.

Personal Data Vault (PDV) [26] is a privacy architecture concept of gathering and storing personal data. Such personal data can be anything from sleep recording, nutrition and diet or everyday activity and exercises. In addition, more sensitive personal data be integrated in the PDV. A Personal Health Record (PHR) is a health record over an end-users lifetime history with diseases, allergies, illness, and hospitalizations.

2.3 Authentication with OAuth

Working and accessing personal data involve access to possibly sensitive data. There are several areas in computer security that must be taken into consideration. OAuth is an open protocol for authorization[7] of web services. Its goal is to be a secure, simple, and standard method to allow users to approve applications to act on their behalf to gain access to resources without sharing their credentials, such as usernames or passwords. For instance, large, software companies such as Facebook, Twitter, and Google uses OAuth. Any application which is able to post something on ones Facebook site have been given permission to do so by you using OAuth.

OAuth is a way to give third-party services permission to use an end-users account information, without revealing the users credentials: either username or password to the service. What differentiates OAuth from for example OpenID, which is a solution based on using a single identity account to access different sites, is that with OAuth you give each third-party the permissions and access to only what they need, without the possibility to see, modify or change anything else and keep your credentials secret.

Any website with commentaries for instance, can make their end-users use OAuth to connect to their social network account, like Facebook, Twitter or Google+ to sign their commentaries. This prevents the users from having to create accounts on every site and the websites from having to implement user accounts in their systems. It also works the other way around. If you for example want an application to see, post or change something on one of your

7. http://oauth.net/

social network sites, you do not give the application your account information, but instead you, via the application, log in to your Facebook, Twitter, or Google+ account and in turn, give the applications the permission it needs.

To achieve its goals, OAuth uses three credentials: client, temporary, and token, with the client credential supporting RSA encryption. The credentials are used to authenticate the client, allowing information to be collected, and resources provided. Tokens are used for giving out usernames and passwords.[8]

There are two version of OAuth: 1.0 and 2.0. Although, version 1.0 is upgraded to 1.0a, that fixed a security fault with the 1.0 version. The main difference between the two versions are security, where OAuth 2.0 relies on SSL using HTTPS. This means one can just send the API key and tokens as query strings, whereas with OAuth 1.0 one must «sign» requests and send two security tokens for each API call.

An important issue with Oauth is that it is non-interoperable with different implementations of the OAuth protocol. Hence, integration of one system that use different versions of OAuth is not straightforward.

2.4 Access Control, Data Management, and Storage

Data storage is how data is retained and maintained in a storage component. Data storage can be structured as a hierarchical pyramid. The top of the pyramid is fast but costly memory. Downwards the pyramid, access to the memory becomes slower but is cheaper to buy. Hence, it is natural that one has most of the cheapest memory. Two concepts that are important in data storage are: volatile, data remains after the power turn off, and non-volatile, data is removed when the power is turning off[32][42].

In data management, access control is about controlling who can access data, so that the user can control how has access to their own data. This can be done with a Access Control List (ACL), which is a list over who has access rights to given data. In the ACL, each end-users privileges are defined. This includes whether a user should have write, delete or read privileges to the data records [42]. There can be multiple or groups of end-users accessing the same data object [38].

8. http://oauth.net/

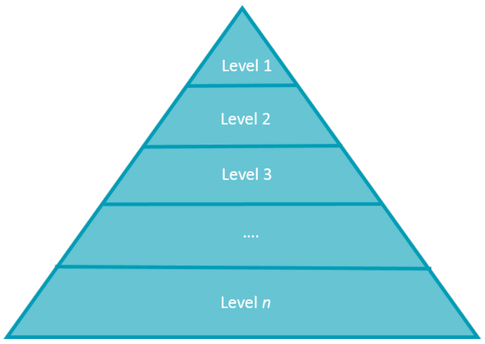


Figure 2.6: Memory hierarchy

2.4.1 Database

A database is a collection of data in a structured and organized manner, and there are many types of organized models for structuring the data. A database schema is the structure describing the database system. Relational Database Management System (RDBMS), store data in related tables, making it quite easy to understand how data is related. A table consists of columns and rows that are related. A database is volatile, and holds information after power is turned. One can expect longer access time to retrieve data stored at a database then in the higher levels of the memory hierarchy. This is of the seek time on disk for finding where the data is located [32].

2.4.2 Cache and Caching Algorithms

Cache is a storage component for temporally storing data. The cache is in one of the top levels of the memory hierarchy. Data stored in a cache is in most cases temporary and the cache is often small in size resulting in fast access time to the data. The cache is also non-volatile, meaning that data will not be preserved when power is turned off. If these data need to be preserved, one has to store data to one of the lower levels in the memory hierarchy [32].

If requested data is present in a cache, it is called a cache hit and if the requested data is not present it is called a cache miss. If a cache miss occurs, one needs to acquire the data from another storage component.

When a cache storage is full, one needs to replace an entry in the cache for a new entry of data. There are several replacement policies for caching:

Least Recently Used: Replace the data that was Least Recent Used (LRU) in the cache. That is, evict from the cache the data that are unused for the longest amount of time. One needs to keep track of when a data was last used. There are several, almost similar replacement algorithm versions and variants based on LRU.

Most Recently Used: Replaced the data in the cache, which was most recently used. That is evict from the cache, the data that is used for the recent amount of time. One need to keep track of when a data was last used.

Random Replacement: Simple, randomly select a data entry for replacement.

2.4.3 "Big Data"

Big data involves large and complex collections of data, where traditional data processing is difficult to apply due to data volume. Challenges range from acquisition of data, analysing, storage, and visualization. Big data defines challenges in three-dimensions [21]:

Volume: Increasing the amount of data volume.

Velocity: In and out speed of data.

Variety: Many heterogeneous data types and sources.

In addition, an update to the definition adds two more challenges:

Veracity: The quality and trustworthiness of the data.

Value: The value and meaning of the data, in the context of how useful or useless the data is.

Big data are used for many purposes. One thing is to find recognise patterns and derive insight in the big volume of data and utilization of that information. For processing the large amount of data, programming models such as MapReduce [8] are often used. MapReduce mainly involves the writing of two functions. A Map function takes the input and maps it into smaller key/value pairs, and assign this smaller problem to working proccesses. Then the Reduce function gathers the results from the working processes and combines these results in a holistic result. An implementation of MapReduce is Hadoop MapReduce[9], which is an open source framework. Additionally, Cogset [44] is a MapReduce implementation that is proven to be more efficent than Hadoop MapReduce in almost every case. Instead of dynamic routing of data done in other MapReduce

9. http://hadoop.apache.org/

implementation, Cogset does the routing static.

In 2012, president Obama, announced and unveiled $200 million in research and development initiative to Big data. To address the important problems that can be faced and dealt with Big data [29].

2.4.4 RESTful API

Representational State Transfer (REST) is a software architecture style principle consisting of several properties[10][36] . These properties and principles are:

Client-Server: Separation of concerns. The client have no concerns about the server- side and vice versa. For instance, the client side has no concerns about the storage at the server-side. The server-side has no concerns about the interface at the client side. Simplifies things. That each side only is concerned about itself.

Stateless: Stateless requests, were the server maintains no static accuse requests.

Uniform Interface: The separation of concern is done through encapsulation. Each part can be developed independently because of the de-coupled design.

Layered System: Load balancing benefits a layering system. Each layer only interacts with its intermediate layer. It simplifies the behaviour and responsibility for each layer by restricting the knowledge of other components in other layers.

In almost all cases REST uses the Hypertext Transfer Protocol (HTTP) for communication. It focuses on how system resources are addressed and transferred over HTTP by any client written in any language. REST architecture principle operates the HTTP methods with CRUD (create, read, update and delete) corresponding with the HTTP requests POST, GET, PUT and DELETE. Since its introduction in 2006, REST has become popular because of its simplicity and usability, often replacing other older technology like SOAP and WSDL [35].

2.5 Data Integration and Interoperability Survey

Data integration is combining data from heterogeneous sources and providing a uniform, homogeneous representation of these data. Interoperability is the ability and functionality of making heterogeneous systems and applications collaborate and work together as a whole system.

Many of the web services in the Health cloud do already connect to one another to share and exchange data. To gain insight in the growing complexity of these interconnected services, we have conducted survey on several popular activity tracking web services, as shown in Figure 2.7. In the figure, blue are systems with dedicated hardware devices, red are professional sport systems, and orange are smart-phone based systems.

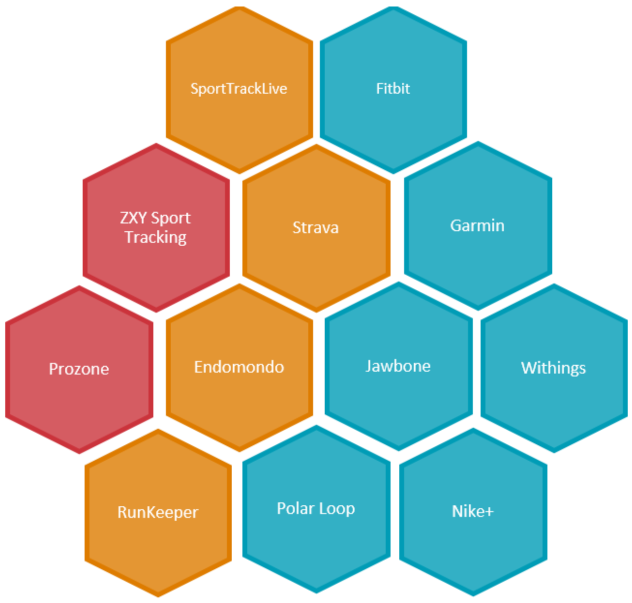
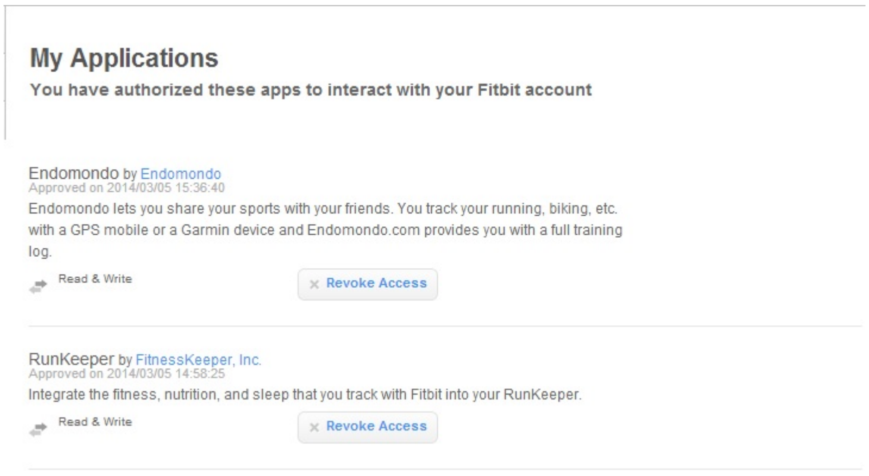


Figure 2.7: Eco-system of tracking system and devices.

We observed that connecting two services mostly consisted of adding an “app” that could intermediate between the web-services. Every end-user account has control over which application or web-services is connected to the account, and different connections between the web-services offers different support for synchronization of end-users personal tracking data. An example of connecting an app from a Fitbit end-user account to Endomnondo and RunKeeper can be found in Figure 2.8. Although interfaces are mostly simple to operate, we found no common mechanism for connecting accounts, and it is unclear what the underlying consistency and data sharing models are.



My Applications

You have authorized these apps to interact with your Fitbit account

Endomondo by Endomondo
Approved on 2014/03/05 15:36:40
Endomondo lets you share your sports with your friends. You track your running, biking, etc. with a GPS mobile or a Garmin device and Endomondo.com provides you with a full training log.

RunKeeper by FitnessKeeper, Inc.
Approved on 2014/03/05 14:58:25
Integrate the fitness, nutrition, and sleep that you track with Fitbit into your RunKeeper.

Figure 2.8: Control over applications connected to a Fitbit end-user account

By manually inspecting all services in Figure 2.7 we constructed a data flow graph, as shown in Figure 2.9, that summarize which services interact with one another. Note that we could only obtain official data flow information from the service providers. Data exchanges between third-party entities, from third-party software developers, is out of scope for this survey.

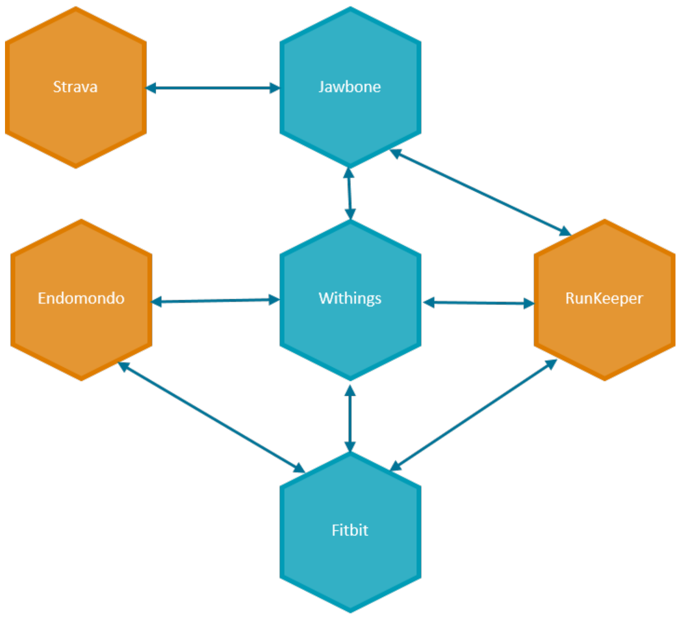
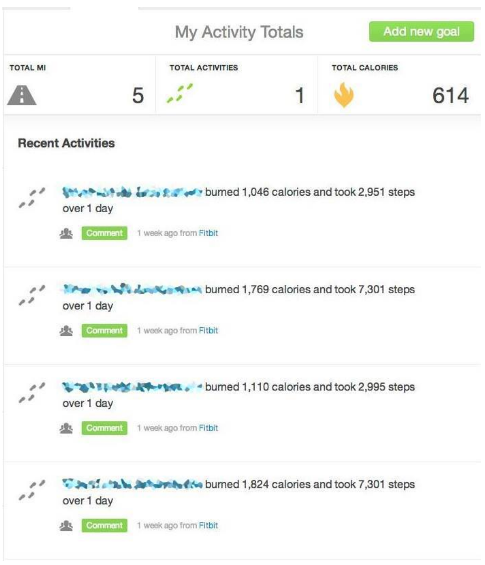


Figure 2.9: Connection of Application, synchronous data between

2.5.1 Data Consistency Issues

Although the functionality to automatically synchronize end-user data between the different services in the self-tracking ecosystem is benificial for availability, it raises the question of data consistency. Moreover, there does seemingly not exist a standard data rapresentation of data format or granularity, which might lead to data corruption or other artefacts. This leads to several problems related to how data flows between different systems, which we will exemplify next.

Data inconsistencies. For instance, when RunKeeper automatically imports Fitbit end-user activity data, this will show in the "Fitness Feed" for the RunKeeper end-user account. Although the imported Fitbit activities shows in the feed, it does not reflect in overall activity summary numbers like calories burned. A Screenshot of this is shown in Figure 2.10. In the upper bar, with information such as total miles, total activities and total calories, is the summary provided for the RunKeeper end-user account. Under this bar one can see the feed with several Fitbit posts with activity from Fitbit.


My Activity Totals

Add new goal

TOTAL MI	TOTAL ACTIVITIES	TOTAL CALORIES
5	1	614

Recent Activities

[redacted] burned 1,046 calories and took 2,951 steps over 1 day
Comment 1 week ago from Fitbit

[redacted] burned 1,769 calories and took 7,301 steps over 1 day
Comment 1 week ago from Fitbit

[redacted] burned 1,110 calories and took 2,995 steps over 1 day
Comment 1 week ago from Fitbit

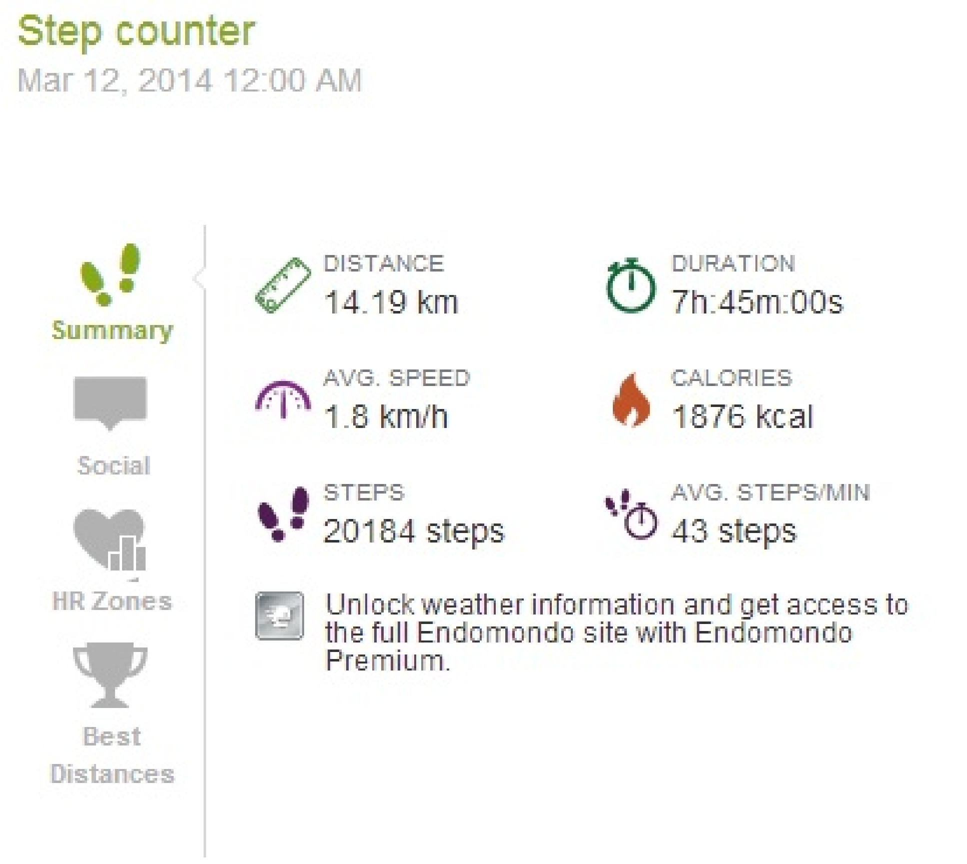
[redacted] burned 1,824 calories and took 7,301 steps over 1 day
Comment 1 week ago from Fitbit

Figure 2.10: Fitness Feed in an end-user account at RunKeeper, the end-user are anonymously

Data duplication. When an end-user has connected Fitbit and Endomondo [10], the activity from Fitbit will automatically synchronize with the Endomondo end-user account, and present the activity with the total summary for the end-user. Figure 2.11 shows the Fitbit activity with distance, duration and steps.

10. http://www.endomondo.com/

These are added to the summary of the end-user account at Endomondo. Then the end-user has duplicated of that activity on two different web-services.

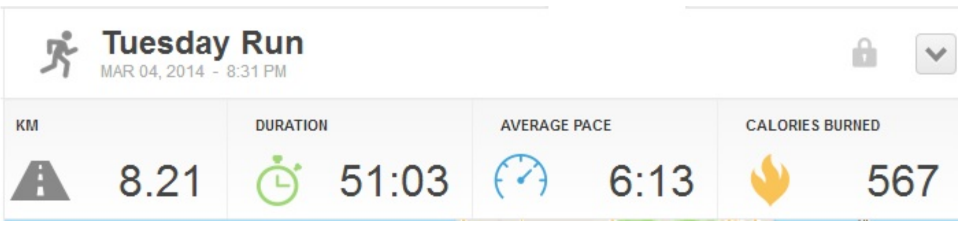

Step counter
Mar 12, 2014 12:00 AM

DISTANCE 14.19 km	DURATION 7h:45m:00s
AVG. SPEED 1.8 km/h	CALORIES 1876 kcal
STEPS 20184 steps	AVG. STEPS/MIN 43 steps

Unlock weather information and get access to the full Endomondo site with Endomondo Premium.

Figure 2.11: Fitbit recording for one day, automatically synchronized with Endomondo and stored there.

Data consistency. Given an activity record from RunKeeper, as shown in Figure 2.12. Exporting the activity in the GPS Exchange Format (GPX) file with


Tuesday Run
MAR 04, 2014 - 8:31 PM

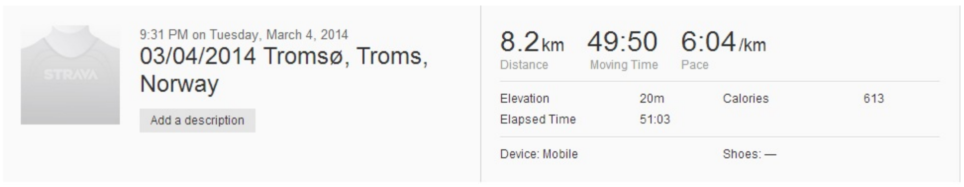
KM	DURATION	AVERAGE PACE	CALORIES BURNED
8.21	51:03	6:13	567

Figure 2.12: A record run in RunKeeper

GPS data format for tracks and routes. The GPX file format data as Extensible Markup Language (XML) format:

```
<trkpt lat="69.668616000" lon="18.916597000">
  <ele>0.0</ele><time>2014-03-04T20:31:39Z</time>
</trkpt>
<trkpt lat="69.668638000" lon="18.916554000">
  <ele>0.0</ele><time>2014-03-04T20:31:39Z</time>
</trkpt>
<trkpt lat="69.668553000" lon="18.916480000">
  <ele>0.0</ele><time>2014-03-04T20:31:45Z</time>
</trkpt>
<trkpt lat="69.668499000" lon="18.916284000">
  <ele>0.0</ele><time>2014-03-04T20:31:50Z</time>
</trkpt>
```

Taking this record and importing it manually into other web-services. These web-services are Endomondo Figure 2.14, Strava [11]Figure 2.13 and Sport-TrackLive [12] Figure 2.15. All the settings are the same, such as gender, age, height and weight.


9:31 PM on Tuesday, March 4, 2014
03/04/2014 Tromsø, Troms, Norway
Add a description

8.2km Distance | 49:50 Moving Time | 6:04/km Pace

Elevation 20m | Calories 613
Elapsed Time 51:03

Device: Mobile | Shoes: —

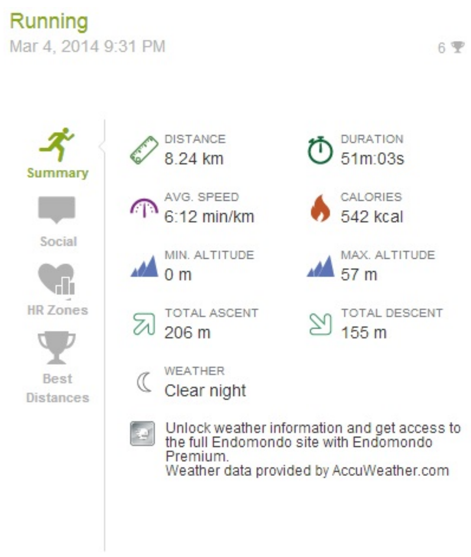
Figure 2.13: Importing the RunKeeper record into Strava

Their are some small deviations to the record after importing the record to the other web-services, and one can assume that the web-services uses different formula and calculation for getting these numbers. Hence, there is some small variation in the web-services, and consistency is not preserved.

2.6 Summary

In this chapter background material for existing self-tracking systems and online services are presented. We show that there exist an ecosystem of interconnected services that synchronize collected end-user data. Our survey over data flow in these services reveals key consistency problems in these services. Emerging out of this information, the prototype will be stated through the knowledges and technical background presented in this chapter.

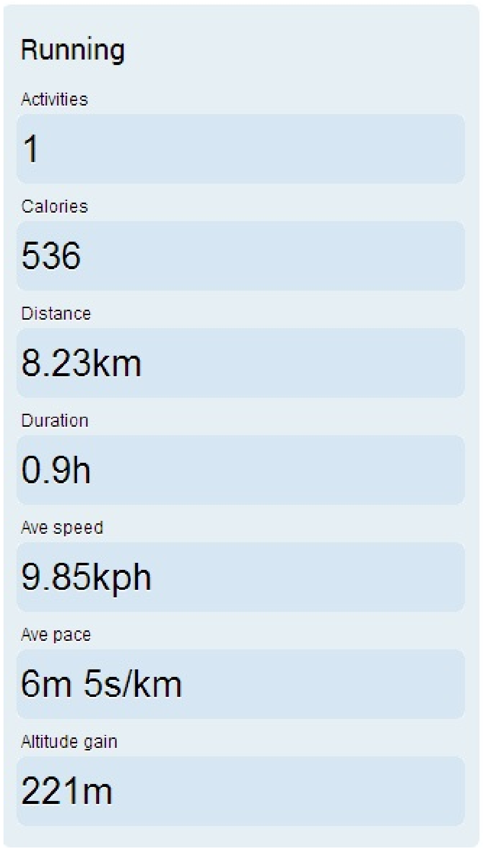
11. http://www.strava.com/
12. http://www.sportstracklive.com/


Running
Mar 4, 2014 9:31 PM
6

DISTANCE 8.24 km
DURATION 51m:03s
AVG. SPEED 6:12 min/km
CALORIES 542 kcal
MIN. ALTITUDE 0 m
MAX. ALTITUDE 57 m
TOTAL ASCENT 206 m
TOTAL DESCENT 155 m
WEATHER Clear night

Unlock weather information and get access to the full Endomondo site with Endomondo Premium.
Weather data provided by AccuWeather.com

Figure 2.14: Importing the RunKeeper record into Endomondo


Running

Activities
1
Calories
536
Distance
8.23km
Duration
0.9h
Ave speed
9.85kph
Ave pace
6m 5s/km
Altitude gain
221m

Figure 2.15: Importing the RunKeeper record into SportTrackLive

3 Requirement Specification

This chapter outlines the requirements of the system based on the problem definition in Section 1.1 and the background knowledge presented in Chapter 2. Both functional and non-functional requirements are stated and we describe the the overall conceptual system model, outlining and defining an abstract overview of the prototype system with the main components and features.

3.1 System Functional Overview

To give end-user insight into their many online PDVs and health-tracking web services, key functional requirements that we must develop are:

1. *Connection* to tracking web-services.
2. *Retrieval* of personal data from connected tracking web-services.
3. *Consolidation* of personal data from multiple tracking web services.
4. *Presentation* of the retrieved personal data to the end-user.
5. *Storage* of data.

Connection and retrieval requirements are necessary as data resides on remote web services and needs to be accessed over the Internet. There is currently no mechanism to push or install processing functions to existing web tracking services [18], and so all data consolidation and transformation must be done in our system.

Establishing connection to a web service involves some form of authentication. As argued in Section 2.3, OAuth is the most commonly used protocol for this and we must be able to store and manage web-service credentials provided by the OAuth protocol.

Once connection is authenticated and established, the system must request data for retrieval. All web service APIs we surveyed in Section 2.5 requires data to be pulled over HTTP. Our system must therefore manage the Uniform Resource Identifer (URI)s for the web service APIs that is to be used and associate each connection with specification and limitations of each web service. Our system will also need to schedule pull intervals between different web service in order to optimize non-functional requirements and UX. In particular, the end-user might specify a wide variety of date ranges which must be mapped down to data request calls for the individual web services.

Data retrieved must be consolidated and presented for the end user. As argued in Chapter 2, data from web services is often heterogeneous with varying formats and granularity. Data consistency issues might also result in data point duplication and other irregularities, as we discovered in our survey in Section 2.5.1. The system must therefore have facilities to specify and execute per web-service data transformation and consolidation rules. This in order to homogenize data for presentation to the end-user.

Based on these requirements, the system is organized in three distinct logical units: a frontend, which interacts with the user, a backend that retrieve and process data requests, and the web services. A highly, abstract overview of these unites are illustrated in Figure 3.1.

3.1.1 Frontend

Through some graphical interface, the end-user interact with the system. Input from the end-user will be in the form of a range *request* for personal data from currently supporting web-services or data sources for the system. These *requests* will be visualized in an interface for the end-user *presenting* the results. Hence, this gives the end-user increased *insight* over where their personal data is stored.

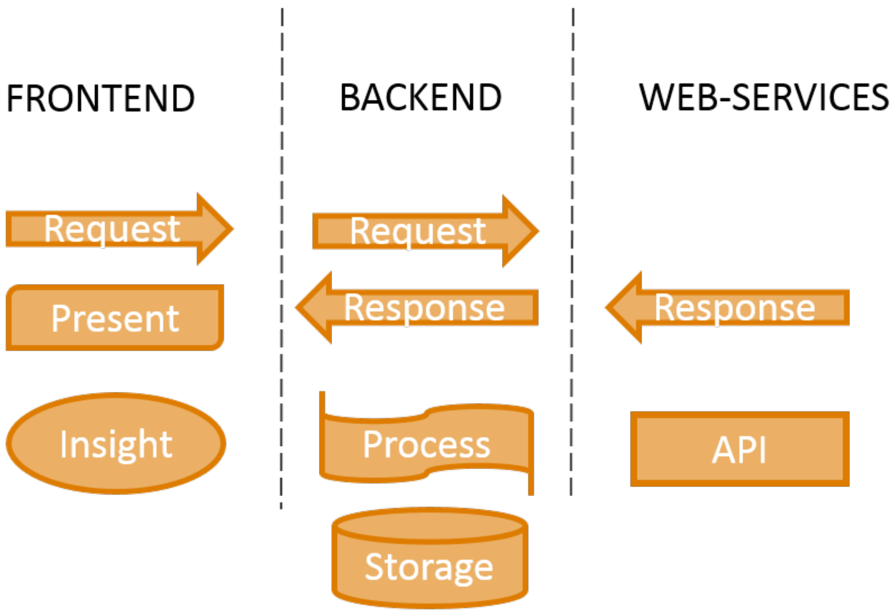


Figure 3.1: The System Model,Abstract Architecture

3.1.2 Backend

The backend unit is the intermediate between the frontend and the web-services. The backend will need to handle *requests* from the frontend and *response* from the web-service unit. The backend shall *process* the *request* from the frontend, depending on the *request* and execute instructions depending on the *request*. Action can for instance be forwarding a *request* from the frontend to the corresponding web-services. Additionally, the backend shall *process* the *response* from the web-services, *process* it and forwarding the *response* to the frontend. Finally, the backend *processes* the data that the system must store in the right level of the storage.

3.1.3 Web-Services

The web-services that will be connected to the system must offer a API for third-party developers to have access to their data and functionality. Selection of the web-service will depend on that factor. Through the API the web-services offers, it will handle requests from the backend and response depending of these requests. Upon a request from the backend, the web-services will return response data to the backend.

3.2 Non-functional requirements

In the following, we will discuss the set of non-functional requirements [43] needed to develop our system in accordance with the problem statement in Section 1.1. We will discuss to what extend each listed requirement impact the system we are to design and implement.

3.2.1 Security and Privacy

When processing personal data, security and privacy are important factors to take into consideration. Authentication and access control against the web-services is already determined by the OAuth mechanisms already in place in these services. Our system must therefore handle OAuth access credentials in a secure and safe manner. Data retrieval over Internet should also use end-to-end encryption in the form of HTTPS when available for the web services and HTTPS/SSL over internal connection that goes over untrusted networks. Personal data should also be stored encrypted on disk. Although security and privacy issues are crucial for real deployments, they are orthogonal to our main objective and therefore will not be the focus of this thesis, as stated in Section 1.3. These issues will be considered as part of eventual future work on the system.

3.2.2 Reliability and Availability

Availability defines that the system must be up and running when an end-user wants to access the system and its features. The system depends on the personal data being provided from the web-services. However the web-service may have access limitation for third-party systems, and this will affect the availability to the system. For example, an end-user may want to retrieve more data than the web-services allows from third-party system. The system will then be unavailable. Reliability states that the data presented is correct and not corrupted. How accurate and reliable the data is recorded at the web-services is out of scope for this thesis. However, the system shall under any circumstance not corrupt the data retrieved from the web-services and seek to present and consolidate the data uniformly regardless how the heterogeneous web-services represents the data.

3.2.3 Extensibility

The eco-system of web-services described in the survey 2.5 are rapidly changing and constantly under development. The system shall be designed and implemented in a way that extensible so that support for future web-services or data sources is possible. Furthermore, adding new features and functionality to the system shall be supported with out having to change the whole implementation.

In addition, the system shall support that if a web-service terminates its APIs, it will be simple to continue to function with out that web-services.

3.2.4 Scalability

The system must be able to handling an increasing number of end-users, without serenely impacting the performance. Issues involve handling personal data storage and maintaining credentials for the end-users. However, the system will not be tested fully with many end-users due to practical limitations. Hence the ability to have multiple, concurrent end-users at the same time, will be considered the future work for this thesis.

In context of computations, functionality, and components take scalability will be considered.

3.2.5 Fault-tolerance

One of the many benefits of utilizing web-services from big software companies is that they have can make these systems and APIs fault tolerant. If a failure should occur, it is likely that they have the ability to recover from it quickly. One can assume that there will be no minimum unavailability and downtime for the web-services. Through redundancy of personal data one can improve fault tolerance by having personal data several places, for instance, stored at the backend of the system. However, redundancy raises consistency concerns, how will the personal data hold it consistency. If an end-user has the functionality and opportunity to modify an activity, the same record at stored elsewhere may then be inconsistent due to modification.

3.2.6 Dependency

The system has a high degree of dependency because it relies on web-services and their APIs for data acquisition. As stated, these web-services may at any time terminate APIs.

3.2.7 Interoperability

Having heterogeneous web-services integrate into a homogeneous system, one needs to take integration and interoperability issues into consideration. Although, in most of the web-services can use the same technology and methods for the APIs, they may represent data differently. For instance, an activity can be represented in milliseconds, seconds, minute, hours or days. The system needs to wrap these data and integrate it into the system to the same data format for consolidating the data. The more inequalities between the web-services, the more one need to take into concern and consideration to make the system integrate comprehensive.

3.2.8 Maintainability

Maintenance of the system shall be simple and implemented in a way so that it is easy for other developers to do maintenance and expansions. Although, as stated by the web-services, one cannot guaranteed how long the web-services will provide an API for third-party as well as maintaining these APIs.

In addition, maintainability is important in case if the developer that started with an implementation leaves, hence no longer works on the implementation, and an another developer acquires to continue working on the implementation. She will need to understand what previously been done by the developer.

3.2.9 Usability

Who the end-users of the system should be is not defined. However, the end-users can have knowledge in ranging from non-technical to technical. This leads to that the system shall have an intuitive User Interface (UI), that is simple to understand. An intuitive design typically leads to a system with a high degree of usability. However, due to practical limitation in evaluation of the system, there will be no end-user survey to verify the usability of the system. Although, under development of the system usability issues and concerns shall be consider[27].

3.2.10 Performance

Performance defines the amount of time and resources used to perform a given task. Depending on the context, a given level of performance will be no requirement for the system. The performance depends on the APIs, which is a factor third-party developers have no control over. That the web-service API have limitations and boundaries, such as rate over access to the web-service, will impact performance.

However, one will seek to increase the performance in making design decisions, such as, storing personal data at the system for increase latency.

3.3 Summary

The system will be an independent, working system that can generate the results outlined. Another aspect is that the system can be linked to the system Girji [20]. The architecture of Girji, is to be an intermediate between the end-user and their personal data and the analytical principals that are analysing their data.

The system can be used as a component for the Girji architecture. The component functionality could then be as a connector to the web-services or data source for data acquisition. In Girji's Consent Object, the system can store the credentials needed for acquiring the data from the web-services or data source. Other systems can also utilize the components for instance for acquisition of end-users personal data to use in their system or in analytic.

This chapter has presented the functional and non-functional requirements. Stating the limitation that must take into consideration under development of the system.

4 Design

This chapter outlines the architecture and design of the system based on the background knowledge in Chapter 2 and the functional and non-functional requirement stated and described in Chapter 3.

4.1 System Architecture

In Chapter 3, the system model was presented and shown to consist of three distinct units. The presentation of the system architecture will be structured around these units.

4.1.1 Frontend

There are mainly two approaches to how an end-user can access their personal data. Either an end-user can access her personal data directly on the web-services sites. The functionality available will then depend on that is offered by the particular site. For instance, can the functionality to export a self-tracked record in an given file format be offered, the end-user can be offered the opportunity to manually input a activity.

The other approach is to use a third-party system, such as this prototype system, that has access though the OAuth authentication protocol to the personal data on the web-services.

The architecture of our system is based on a client-server model, where the client side provides the GUI. The input from the end-user will mainly be request for personal data from the web-services in form of a range query. The server side receives requests from the client side, processes and performs the requested service.

The frontend computes the personal data for presentation and consolidation in the GUI. By having the computation on the frontend the system scalability is increase. Although, given a very large number, for instance 10 000 or 100 000, end-users to the system, backend performance will impact overall system performance. A solution to this would be to have multiple homogeneous back-ends with the same component connection to a master backend that has the storage component with all the authentication credentials for all the end-users. The main reason for not implementing this approach is the limitation in testing with multiple end-users.

The frontend UI runs in a web browser, which make it possible to access from many devices, such as stationary computers, laptops, tablets and smart phones. In addition, the advantage with running the system in a web browser solution that it will be independent of the operating system or platform. One issue can be if a specific web browser does not have support for a given functionality. An alternative solution for the frontend UI is to have a program or application in-stalled at the frontend. With a program or application, the end-user must install it on the frontend, and one needs to take into consideration what operating system the program or application will be installed on.

4.1.2 Backend API

The backend API is the intermediate between the frontend and all the web-services, and provides interoperability when different heterogeneous web-services respond to a given request. By making the backend in this way, extensibility and maintainability is increased. The backend handles request from the fron-tend and processes these requests. The handling involves sending requests to the web-services and receipt of responses. The main type of requests from the frontend are range queries, for retrieval of end-user personal data from the web-services.

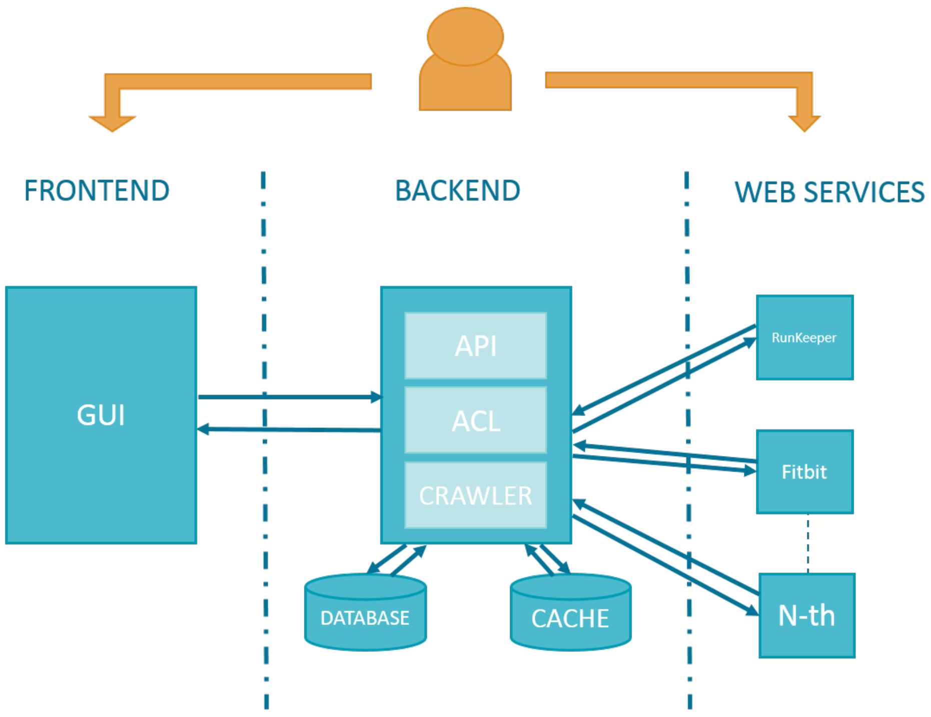


Figure 4.1: System Architecture

4.1.3 Access Control

A major consequence with personal data scattered over different systems and web-services is that an end-user can quickly lose control over who has access to their personal data. The system supports that every end-user has control over who has access to retrieve their personal data through the system from the supported web-services.

One of the main components of the system are two Access Control Lists ACL. One is for controlling who the end-user has authorized to access her personal data. The other is a list of who the end-user has access to retrieve personal data on behalf of. A use case scenario for this is if a coach wants to access a player's activity to gain insight in a player's activity or perhaps sleep quality. A coach can then be pro-active and find reasons why the performance on the field is not as expected, and analyze and use this personal data for improved coaching.

4.1.4 Storage

One of the main reasons for the system to have a storage component is for increasing performance and fault tolerance. In context of performance, given that the web-services have access and request limitations, and if these limitations are encountered, the system will then be unavailable. However, if a defined amount of end-user personal data is stored at the backend storage component it will decrease the load upon the web-services. Furthermore, having replication of personal data supports and increases the systems fault tolerance. Hence, storing personal data impact the performance by decrease the access time to retrieve the personal data.

The storage component of the system will contain a cache and a MySQL database. Retrieving end-user data from the different web services requires end-user credentials, and these credentials are stored in an end-user table in the database the very first time an end-user logs into the system. By storing the end-user credentials, the end-user does not have to the same first sign in stages the next time she uses the system. A database table will be used to hold the ACL describing who has access to end-user data.

In the cache, end-user personal data is temporally stored from current date and backwards 30 days, because it is more likely that an end-user chooses to examine and analysis the freshest data. In Haystack [3][14], although in context of photos, they prove that the newest photos are the most likely to be revisit in the nearest future. The same argument may be used for personal data; the freshest data is the most likely to be accessed.

Another scenario for supporting the choice of what personal data to cache, is that an end-user will also for current insight and progression see yesterday, last week or perhaps last months activity. The number of steps taken on a given date several months or years ago is not that likely to be accessed.

"You may cache data you receive through use of the Health Graph API in order to improve your application's user experience, but you should keep the data up to date."[1], state that caching of Health graph data is allowed.

If an end-user wants to delete her personal data and account from the system, her personal data and credentials are deleted in all storage components. Although stated at RunKeeper: "Should a user disconnect from your application, you may continue to store previously cached data, unless the user requests (via mechanism we provide) that you delete such data. Upon such request, you

1. http://developer.runkeeper.com/healthgraph/api-policies

shall delete the cached data of the disconnected user."[2]

An alternative system architecture is to remove the backend and have the frontend access the web-services directly. One will then perhaps increase the performance. One of the main issues supporting the backend solutions is that the end-user authentication credentials should never be sent to the frontend side. A security issue with having authentication credentials at the frontend, however, is that one will then have to encrypt or embed them. Another issue with not having a backend is that if an end-user accesses the web-services from different devices the authentication process needs to be complied for each device, since the credentials would be stored locally at the devices without an backend solution.

4.1.5 Crawling

The crawler component of the system is for fetching the freshest data for each end-user. This functionality ensures that the cache will contain the newest data generated by the end-users. End-user personal data from RunKeeper is not crawled because of stated in the API policies: "You cannot use web scraping, web harvesting, or web data extraction methods to extract data from the Health Graph or RunKeeper"[3], To not violate these terms, no data from RunKeeper is fetched without real-time end-user interaction. Regardless, RunKeeper have no access limitation, therefore there are no need to crawl and cache personal data from RunKeeper.

Crawlers have policies that will affect their design and behaviour. Combinations of policies are selection, re-visit, politeness and parallelization policies.

Selection Policy: Selects Fitbit Sleep and Activity through the resource Universal Resource Location (URL).

Re-visit Policy: Uniform frequency visits the selected cites every night after midnight in a fixed order.

Politeness Policy: Limitation per end-user is 150 request per hour, and only 2 requests per end-user. The crawler should not in any cases overload the server for the web-services.

Parallelization Policy: Not taken into account. Seen as out of scope for this theses, can be added to further work.

2. http://developer.runkeeper.com/healthgraph/api-policies
3. http://developer.runkeeper.com/healthgraph/api-policies

The crawler components backups the performance of the system through holding the freshest personal data in the system, which is most likely to me access by the end-user.

4.2 Web Services and their APIs

Using a API can have many advantages and disadvantages. One of the major drawbacks can be if the company that has developed the API changes it so much that current systems or applications that are using it cannot longer work properly because of the changes. A worst case scenario is if the company decides to stop supporting and terminates the API. The web-services are factor that the third-party developers have minimal control over.

An example of this is as follows:"Further, you acknowledge that Fitbit reserves the right to disable or upgrade the Fitbit API and related services at any time without notice to you and without any form of compensation or consideration to you, regardless of the status of any Developer Applications. Fitbit has no obligation to ensure that an upgrade of the Fitbit API or related services will continue to be compatible with existing Developer Applications." as stated by Fitbit.

"Termination. FitnessKeeper, Inc. may change, suspend or discontinue all or any aspect of the Health Graph API, including its availability, at any time, and may suspend or terminate your use of the Health Graph API at any time. FitnessKeeper, Inc. may terminate your access to the Health Graph API and your right to use it at any time, for any reason, or for no reason including, but not limited to, if you engage in any action that reflects poorly on FitnessKeeper, Inc. or otherwise disparages or devalues the FitnessKeeper trademarks or FitnessKeeper, Inc.'s reputation or goodwill. If you desire to terminate your agreement to comply with the Health Graph Terms and Policies, you must cease to use the Health Graph API" stated by RunKeeper.

All APIs sets some terms and policies for the third-party developers, to protect the rights and privacy of the end-users. The developers must agree on these terms and if they are violated, the access to the API can be withdrawn from the third-party.

There is an ecosystem of web-services out there, as the Survey in Section 2.5 illustrates, but not many off them offers a third-party API for developers. This will limit what web-services can be connected to our system. The web-services that the system is currently supports include RunKeeper and Fitbit, illustrated in Table 4.1.

Web Service	Type	Data
Fitbit Flex	Armband	Steps, calories, sleep
RunKeeper App	Smartphone	Position, calories, distance etc

Table 4.1: Current Data Sources

4.2.1 Fitbit

Although an end-users activities and sleep can be recorded by the same wearable device, it is not guaranteed that the end-user remembers to track her sleep. For that reason, the system sees the sleep and the activities from an end-users Fitbit account as two different sources of personal data.

Fitbit has two APIs: A one Public and a one Partner API. The Public API can developers use after they have registered their application on Fitbit developer cite[4]. Here the developer states and describes the purpose of the application and where the application should be redirected through stating the callback URL. The Public API is currently rate limited to 150 requests per hour for each end-user. Although the number of end-users of the system may increase, it will not affect the limit rate for the system using the Fitbit API. This limitation is for insuring a consistent and stable access for all applications. If the limit is exceeded, the Fitbit API will email the developer a notification to the email address registered.

The Partner API is an extension to the Public API. In addition to offering the same as the Public API it can also fetch more detailed end-user data with finer granularity. For instance, the Public API offers only number of steps per day, but with the Partner API, one can retrieve with a granularity of steps per 15 min or 1 min, making the data fetched more accurate and precise. To gain access to the Partner API a developer must inquire a personal request to Fitbit. These request are granted individually on a case-by-case basis, depending on why it is necessary to have access to the Partner API.

The system will use the partner API from Fitbit for accessing optimal granularity for activity. The access to the partner API is gain though Girji [20], that has these privileges.

4. https://dev.fitbit.com/apps/new

4.2.2 RunKeeper

RunKeeper and the Healthgraph API have not listed any limitations upon what the system needs to fetch of personal data and will not affect the design of the system.[5].

Although, it would be most preferred to utilize web-service APIs with no limitation. One must use the different web-service because they track and record different personal data. For instance, RunKeeper do not track sleep activity.

4.3 Summary

In this chapter the architecture of the system is discuses and presented. Design decisions are taken in relation to the functional and non-functional requirements stated in Chapter 3.

5. http://developer.runkeeper.com/healthgraph/api-policies

5 Láhttu

This chapter presents the Láhttu system. Láhttu is a word from the Sami language meaning making tracks in the snow done by ski. These tracks leave a visible trace for others, until the tracks eventually melt or snow away. Similarly, tracking activities can leave a virtual track of the activity for insight, studying, and analyzing. The many details around the implementation of Láhttu is describe in this chapter.

5.1 Frontend

As stated in Chapter 3 and Chapter 4, Láhttu has a frontend unit. The Frontend part of Láhttu consists of GUI for the end-users, as well for input requests from the end-users. For accessing and using Láhttu, creation of a user account and password is not necessary. Láhttu utilizes already existing technology and methods for managing end-user account. Maintenance of end-user information and corresponding end-user password are the RunKeeper handles. Through methods offered by Healthgraph, a developer can easily establish a connection to RunKeeper by configuring a login button with the proper information that is required. The required information is the Client ID, a 16 bytes HEX string, which is assigned to the developer after registration of the third-party system or application on the HealthGraph. Also, it contain a redirect URI for where the end-users should be redirected after the signing process, which is the main page of Láhttu. The developer can also configure login buttons with Cascaading Style

Sheets (CSS) and structure HyperText Markup Language (HTML) through support from the HealthGraph. This button is shown in the screenshot 5.1. This is a method of authentication when signing up for Láhttu or returning.[1]

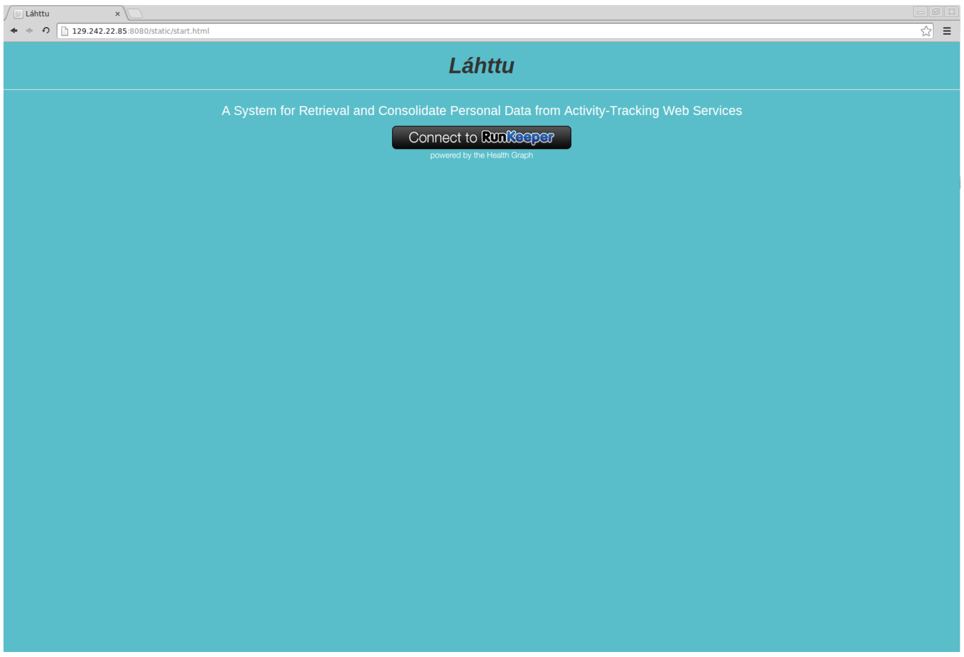
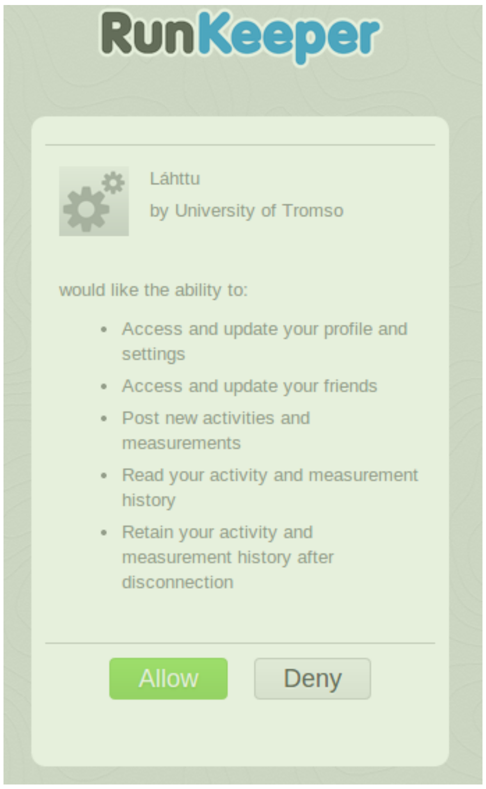

Láhttu
129.242.22.85:8080/static/start.html
Láhttu
A System for Retrieval and Consolidate Personal Data from Activity-Tracking Web Services
Connect to RunKeeper
powered by the Health Graph

Figure 5.1: Screenshot of Láhttu login page with the RunKeeper connection button

The end-user is upon first login presented with two options; either allowing Láhttu privilege to access their personal data on the terms stated at the end-users account at RunKeeper, or denying Láhttu access. Shown in screenshot 5.2.

If an end-user chooses to deny Láhttu access, the end-user will be redirected to an error page. However, accepting the terms of Láhttu for RunKeeper, the end-user will be redirected to the main page of Láhttu. The process is shown in Figure 5.3.

1. http://developer.runkeeper.com/healthgraph/login-plugin


RunKeeper

Láhttu
by University of Tromso

would like the ability to:

- Access and update your profile and settings
- Access and update your friends
- Post new activities and measurements
- Read your activity and measurement history
- Retain your activity and measurement history after disconnection

Figure 5.2: Screenshot of the option for the end-user for RunKeeper

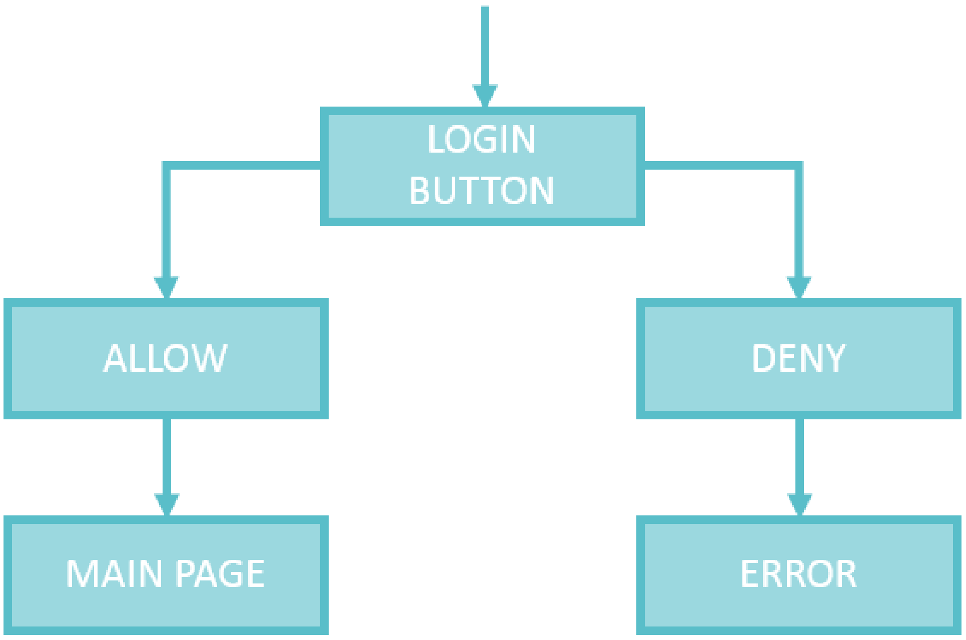
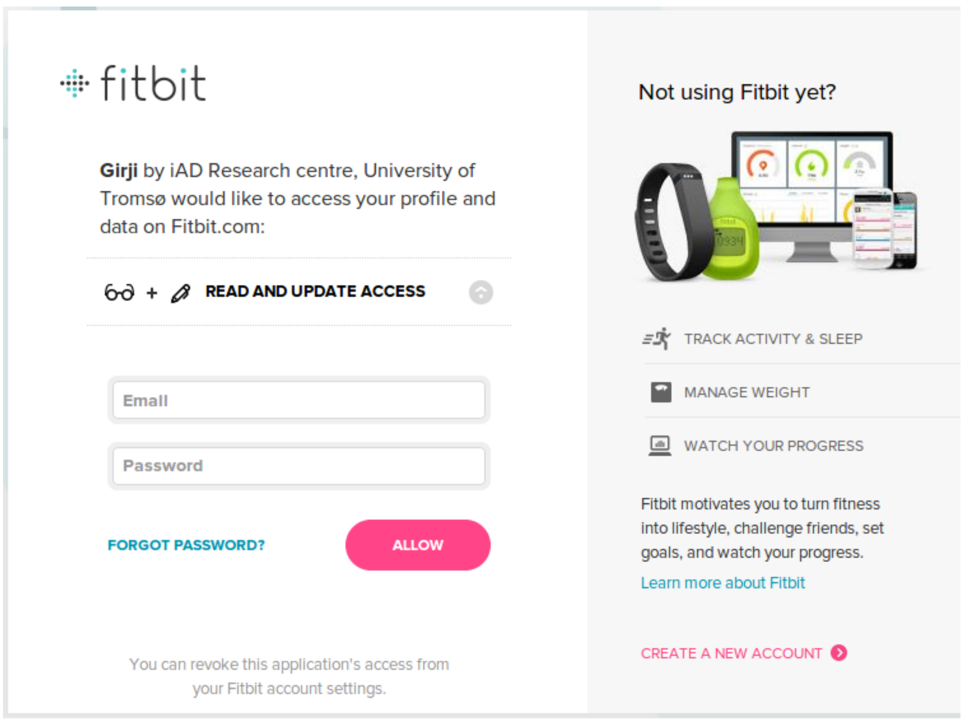


Figure 5.3: Flow over the login process

Once an end-user has allowed and accepted the terms of Láhttu for their RunKeeper account, the system checks in the database for the Fitbit credentials for the end-user. If they do not exist in the database, the end-user is redirected to the Fitbit site, where the end-user must allow or deny Láhttu privilege to access her personal data on Fitbit. Láhttu utilizes the Partner API for Fitbit, this privilege is gain through the Girji system [20] which has access to the Partner API, as Figure 5.4 illustrates.



Girji by iAD Research centre, University of Tromsø would like to access your profile and data on Fitbit.com:

READ AND UPDATE ACCESS

Email

Password

FORGOT PASSWORD? ALLOW

You can revoke this application's access from your Fitbit account settings.

Not using Fitbit yet?

TRACK ACTIVITY & SLEEP

MANAGE WEIGHT

WATCH YOUR PROGRESS

Fitbit motivates you to turn fitness into lifestyle, challenge friends, set goals, and watch your progress.

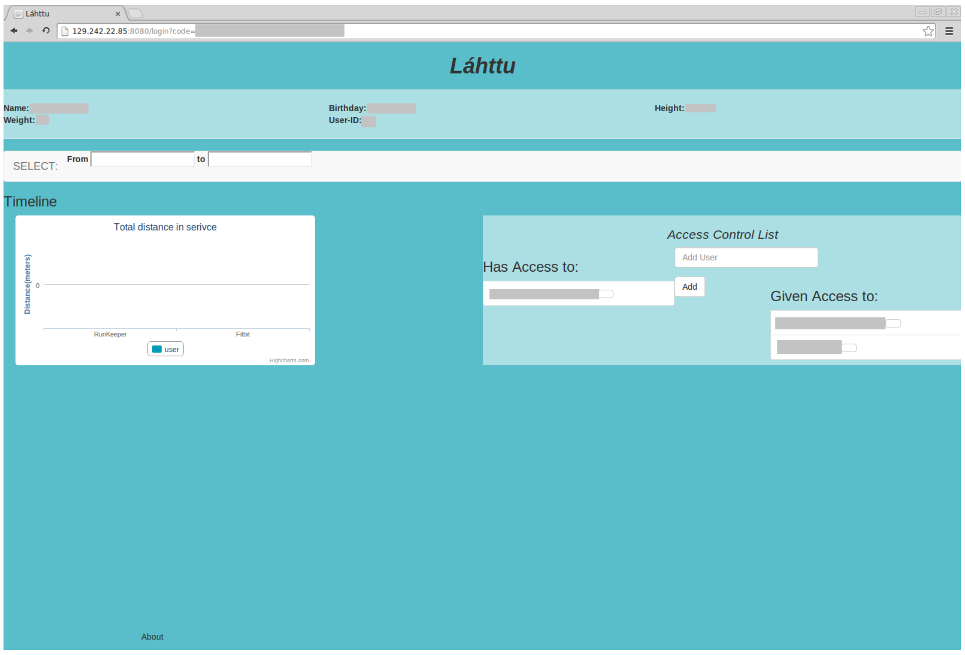
Learn more about Fitbit

CREATE A NEW ACCOUNT

Figure 5.4: Screenshot of the option for the end-user for Fitbit

Both for RunKeeper and Fitbit this will only happen the very first time a end-user uses Láhttu.

When the sign-in process is done, the end-user is at the main site of Láhttu where the end-user can choose a range query between two dates for fetching personal data from any web service Láhttu currently supports, as shown in Figure 5.5.


Láhttu

129.242.22.85:8080/login?code=[redacted]

Name: [redacted]
Weight: [redacted]
Birthday: [redacted]
User-ID: [redacted]
Height: [redacted]

SELECT: From to

Timeline

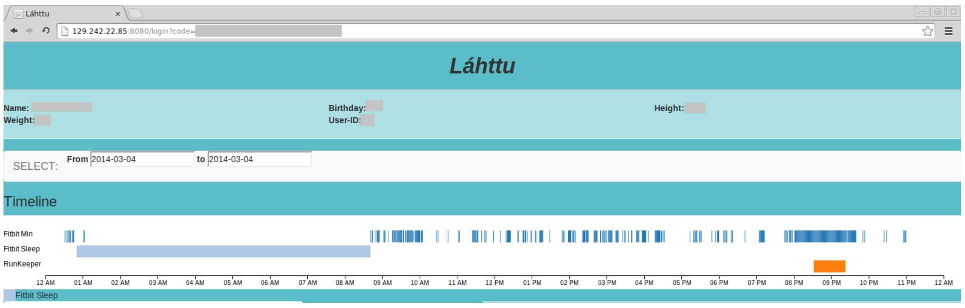
Access Control List

Has Access to: [redacted]

Given Access to: [redacted] [redacted]

Figure 5.5: Screenshot of the main page of Láhttu

The end-user will then be presented with an timeline over consolidation of activity recorded and at which web services the personal data is located. Shown in screenshot 5.6.


Láhttu

129.242.22.85:8080/login?code=[redacted]

Name: [redacted]
Weight: [redacted]
Birthday: [redacted]
User-ID: [redacted]
Height: [redacted]

SELECT: From 2014-03-04 to 2014-03-04

Timeline

Figure 5.6: Screenshot of the main page of Láhttu after an Range Query for 4th of March

Láhttu at the client-side offers a dynamic, interactive web interface, and this is implemented by using JavaScript with several JavaScript libraries, as Shown in Table 5.2. Communication and requests from the end-user are done by Asynchronous JavaScript and XML (AJAX) in an asynchronous style with HTTP methods between the frontend and backend, without reloading the interactive web interfaces of the frontend[23]. However, communication could have been done with WebSockets, which initialize a TCP connection between the client and server side. The communication channel is held open until the connection is closed, making it possible for the server side to send content to the client side. A major drawback with WebSocket is that not all web browsers have support for WebSockets.

AJAX takes advantage of JavaScript event loop. This is simply a queue where messages to be processed are added. JavaScript runs in a single thread meaning executing functions will block events, like an end-user pressing a button, until it has finished executing. Only then, the JavaScript runtime can process the button click event. Every message in the message queue has a function attached that the JavsScript runtime will execute. For AJAX, when a response is received from the back-end, a message is added to the message queue, with a callback function to be executed attached. The callback specifies what to do, for instance, to process the response data and add it to a HTML DIV so it will be visible in the web browser for the end-user. Any time an event occurs, the message queue is updated, but only if there is a listener attached to the event. A button without an event listener will fade silently.

The AJAX calls are done with jQuery[2], which is a JavaScript library that simplifies all the standard JavaScript functionalities. In addition, jQuery is utilized in the implementation for accessing different HTML DOM-elements. The argument for this is for the simplicity. jQuery is one of the most popular JavaScript libraries available, and big company such as Google, Microsoft, IBM and Netflix are utilizing jQuery.

Data between the frontend and backend is represented by JavaScript Object Notation (JSON) format[3]. Although JSON originates from JavaScript, JSON is a completely independent language data interchange format. It represents data in collections of key-value pairs in a similar way as dictionaries, struct, and hash tables do in various languages. In addition it supports ordered lists of values similar to arrays, lists or vectors. Additionally, JSON is faster and easier to parse than XML and has proven to be a good replacement for XML.

The UI is done with CSS and HTML, with several included libraries. These libraries are the Bootstrap libary that has design templates for CSS and HTML and Highchart libary for charts. The main features of the GUI are the timeline that presents the range between two given dates with personal data from all the web services. The timeline is from the D3-library [4], with several methods for representing data in timelines. The timeline represents date and time in

2. http://jquery.com/
3. http://www.json.org/
4. //github.com/jiahuang/d3-timeline

Language	Purpose
JavaScript	Interacktiv website
HTML	Structure of the website
CSS	Styling of the website

Table 5.1: Language at Frontend.

Library	Purpose
Highcharts	varies charts
Bootstrap	Twitter framework
D3-timeline	Timeline
jQuery	simplify
AJAX	Asynchronous communication between front-and-backend

Table 5.2: JavaScript Libraries and features.

milliseconds, so personal data from the varies web-services must wrap up the date to integrate with the timeline.

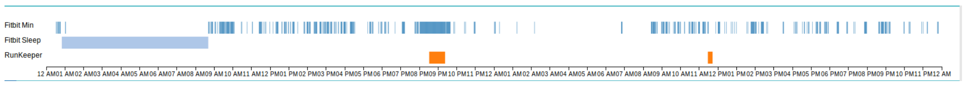

Fitbit Min
Fitbit Sleep
RunKeeper
12 AM01 AM02 AM03 AM04 AM05 AM06 AM07 AM08 AM09 AM10 AM11 AM12 PM01 PM02 PM03 PM04 PM05 PM06 PM07 PM08 PM09 PM10 PM11 PM12 AM01 AM02 AM03 AM04 AM05 AM06 AM07 AM08 AM09 AM10 AM11 AM12 PM01 PM02 PM03 PM04 PM05 PM06 PM07 PM08 PM09 PM10 PM11 PM12 AM

Figure 5.7: Screenshot of the timeline with personal data in the range of 4th to 5th of March

5.2 Backend

The backend is the server-side software and intermediate that performs, processes and handles the request upon the web services and from the frontend. When selecting the programming language for implementing the backend, there are several aspects to take into consideration. One of the main justifications will depend on what programming languages the web-services support. Personal preferences will also affect the choice. The backend is implemented in Python 2.7.4[5], which is a high-level, object-oriented programming language. Python offers numerous libraries for functionality. The libraries utilized in this implementation are stated in table 5.4. Although C would have better performance and low-level details than Python, one must code more to achieve the same functionality.

5. https://www.python.org/

HTTP methods	Action
HTTP GET/RunKeeper	Get range query of activity from RunKeeper
HTTP GET/FitbitCredentials	Check for credentials, if not present, retrieves
HTTP GET/index	Start html page
HTTP GET/login	Login button
HTTP GET/FitbitSleep	Get sleep activity for the query range
HTTP GET/FitbitActivityMIN	Get steps activity for the query range
HTTP GET/FitbitUserInfo	User info from fitbit
HTTP GET/FitbitDistance	Total distance in query range from fitbit
HTTP GET/checkAccessID	Check the access ID
HTTP GET/getUserID	Get user ID for the current end-user
HTTP GET/checkAccess	Retrieves all the end-user one have access to
HTTP PUT/GivenAccessTo	Give access to an end-user
HTTP DELETE/removeAccess	Removes access from a end-user

Table 5.3: HTTP methods

Library	Purpose
JSON	json support
Requests	Sending HTTP request
web	Web.py
rauth	Oauth lib
datetime	manipulating date and time
MySQLdb	MySQL lib
pystache	Mustache lib
healthgraph	HealthGraph lib
urllib	fetching data
sys	access the interpreter
os	operating system interface

Table 5.4: Python Libraries

The web framework is done with web.py [6], a simple, lightweight Python web framework, exposing the REST interface. The backend exposes a API for the JavaScript to use. The end-users send data requests from the frontend, through a HTTP GET with a query string embedded in the URL. The query string contains data passed from the frontend request to the backend, to be operated on at the backend. The HTTP methods is in Table 5.3.

All the components on the frontend and backend side supports UTF-8 encoding

6. http://webpy.org/

for representing every character in the unicode character set. The main reason for that choice is to support the Norwegian letters ’Æ’,’Ø’ and ’Å’.

5.2.1 Time, Date, and Range Query

Time and date is represented in "YYYY-MM-DD" format, both for RunKeeper and Fitbit. When selecting a range for the range query, one selects from a starting date to a stop date. Further, three seperate HTTP GET request are generated and sent to the backend with a URL with a query range embedded in a query string. At the backend side each request is routed to the correct function based on the URL. All three functions at the backend get the end-user credentials from the storage. For RunKeeper, the request is that the starting date is "noEarlierThen" and stop date is "noLaterThen", and RunKeeper returns all activity between these dates. If an end-user request a whole year from RunKeeper, and there is only one activity in the requestet date range, the response will only contain one item. However, if an end-user request for a year from Fitbit, the response will contain all dates whether the dates have recorded an activity or not. If a date do not contain a date, it will only contain ’0’ values.

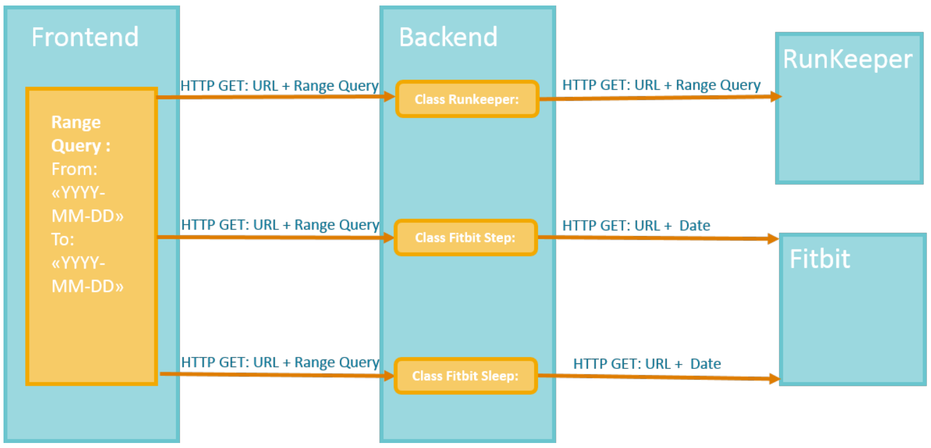

Frontend
Range Query : From: «YYYY-MM-DD» To: «YYYY-MM-DD»
HTTP GET: URL + Range Query
HTTP GET: URL + Range Query
HTTP GET: URL + Range Query
Backend
Class Runkeeper:
Class Fitbit Step:
Class Fitbit Sleep:
HTTP GET: URL + Range Query
HTTP GET: URL + Date
HTTP GET: URL + Date
RunKeeper
Fitbit

Figure 5.8: Request Range Query

Fitbit, for both sleep and steps, needs a loop for retrieving all personal data in the desired range. Thus, the greater the range in the range query is, the more processing the backend has to perform. The frontend sends a range query request with the range in the "YYYY-MM-DD" format. On the backend side, the Python library datetime[7] converts the date into a datetime object.

7. https://docs.python.org/2/library/datetime.html

```
start_range = datetime.datetime.strptime(noEarlierThan,
"%Y-%m-%d".date().isoformat())

stop_range = datetime.datetime.strptime(noLaterThan,
 "%Y-%m-%d".date().isoformat()
```

In the loop, as long as start range is less or equal to the stop range, a request to Fitbit for the date is sent. When Fitbit has responded to the request, the datetime object is increased by one day. This is done until all dates in the range are retrieved.

```
start_range = start_range + datetime.timedelta(days=1)
```

The whole request is illustrated in Figure 5.8. Although, this could have been differently, by sending only one request from the frontend to the backend. Having three separate request handles makes the maintenance of the system simpler, if one were to remove a web-service. In addition, responses are easier to handle by separating the requests.

5.2.2 Storage

Láhttu has an hierarchy of storage components: a cache and a MySQL database, each for different purposes.

Cache

The cache of Láhttu is a key-value storage, where the key is "unique 16 bytes HEX string + which data source + date" for each insertion in the cache. The cache holds the last 30 days from current date and backwards. The crawler component is the one that insert into the cache for the end-users. If an end-user requests an older record of data than the previous 30 days, this record will not be inserted into the cache. When a new date is to be inserted, the oldest date is replaced in the cache. The cache has a First In First Out (FIFO) queue implemented as a stack for holding all the keys. When one replaces a key, the oldest key is popped of the FIFO queue. LRU replacement policy is not necessary because the cache will always hold the freshest 30 days of data, and not the personal data the end-users has recently accessed. Also, achieving a good LRU replacement policy one will need to keep track of when a value was last accessed.

The cache could have been implemented as a list structure with 30 indexes and removing always the value in the last index. However, when the data source can be of various source, it is better to have a key and direct look up on that key instead of iterating through the list. Another list structure solution would be to have several list where each list contained a unique data source.

When an end-user request a date the system check first in the cache, if the date is not contained in the cache, there is a cache miss resulting further that the backend request the web-services for the date for personal data.

Database

The main storage of Láhttu is a MySQL database performing Structured Query Language (SQL) queries upon. There are two tables: one for the user credentials and one for ACL. Every delete and insert query upon the database tables has commit and rollback functionality.

The user credentials hold for each row in the tablets, all the different web services credentials for accessing the web services for the end-user. In addition, the ID-number and Name of the end-user is included.

The ACL tables hold end-user access authorizations organized around end-users ID-numbers, each has a one-to-one mapping for access privilege. Thus, a row describes which end-users that have given the access and which end-user now have these privileges.

The database handles SQL injection by escaping the variables in the query. Using "%s" instead of direct variables, the variable or value is then passed through a tuple or a list. This is done for security and protection reasons, so that the database table would not be altered through end-user input in the system.

```
query = cur.execute("SELECT Fitbit_rok, Fitbit_ros
FROM user_credentials WHERE ID = %s", ID)
```

5.2.3 Crawler

One of the main components of Láhttu is the crawler, which fetches personal data from the web-services automatically without any real-time end-users input. A cron job in Linux is scheduled to start every night after midnight for starting up the crawling. It starts by using the current date, to calculate the previous date of interest.

```
today = datetime.date.today()
yesterday = today - datetime.timedelta(1)
```

Then the crawler queries for all the credentials for all the end-users currently in the database table. These credentials are the RunKeeper Access Token, Fitbit Resource Owner Key and Secret. These credentials are needed for retrieving the personal data on behalf of the end-users. For each end-user in the table, the crawler checks if there exists a cache entry for that end-user, if not, it will be initialized. A key is generated for the cache with the right name. Then the Fitbit activity is fetched and inserted into the cache for the end-user.

The same process is repeated for the Fitbit sleep data for the end-users. The crawler continues in the same order until all the end-users personal data is fetched and inserted into the cache.

5.2.4 Web-services API

Láhttu currently supports two different web services: Fitbit and RunKeeper. There are several web services which have no API for third-party developers.

Getting started with these APIs, the developer must register her system or application on the website to the web service. For Fitbit and RunKeeper, registration of an application the developer insert information such as in table 5.5 and 5.6. The third-party developer must read and agree to the terms of the web-services when registering. After the registration, the developer will receive for her application a Client Key, a 16 bytes HEX string and a Client secret, a 16 bytes HEX string. Also, she will be given a URL for the authorization part, including URL for token retrieving. Although, Fitbit and RunKeeper have many of the same processes here, they use different versions of OAuth.

Type	Information
Application Name	Developers choice of name
Description	Developers description of the application
Application Website	main url of the application
Application Type	Desktop or Browser
Callback URL	URL for redirecting
Default Access Type	Read and Write or Read-Only

Table 5.5: Register an application at Fitbit, Required Information

Type	Information
Application Name	Developers choice of name
Description	Developers description of the application
Application Icon	picture
Permission requests	Read, Edit and Retain Health Information
Authorization Removal Callback URL	max. 2048 charther

Table 5.6: Register an application at RunKeeper, Required Information

Both APIs offer response in JSON, and additionally Fitbit API offers responses in XML. Taken the request with the range query and the date "2014-03-04", illustrated in Figure 5.8, an successful response is illustrated in the Figure 5.9.

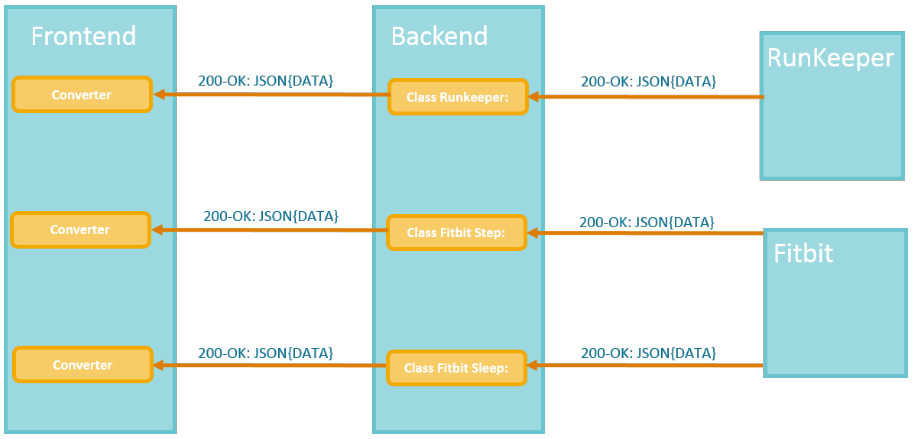


Figure 5.9: Response Range Query

The response from RunKeeper is as described in Appendix A.1. The backend forwards the response to the frontend without modifying the data. On the frontend side, the data retrieved from the backend is converted to adapt and integrate with the timeline. RunKeeper represents the activity duration in seconds and the timeline in milliseconds. On the frontend the response data is stored in the variable runKeeperdata. Furthermore, the convert function accesses the values in the JSON for the start time of the activity and duration of the activity. The function uses Date.parse() to parses the time into a UNIX time. Converting the duration of the activity from seconds into milliseconds.

```
var start_time = runKeeperdata["items"][i]["stater_time"];
var unix_start_time = Date.parse(start_time);
var duration = runKeeperdata["items"][i]["duration"];

duration = duration * 1000;
```

```
var unix_stop_time = unix_start_time + duration;
```

The response from Fitbit Steps is as described in Appendix A.2. On the frontend the response data is stored in the variable FitbitMINData. Furthermore, the convert function accesses the values in the JSON for steps in the date. If there are no steps in a given minute, the value for that minute is '0' and will not be taken care of. The function uses Date.parse() to parse the time into a UNIX time. The steps are represented in steps per minute, and converted this to milliseconds to integrate with the timeline for consolidation.

```
for(var i=0; i < FitbitMINData.length;i++){
    var date_time = FitbitMINData[i]["activities-steps"]
    [0]["dateTime"];
    for(var j=0; j < FitbitMINData[i]["activities-steps-intraday"]
    ["dataset"].length;j++){
        var times_dict = {};
        var value = FitbitMINData[i]["activities-steps-intraday"]
        ["dataset"][j]["value"];
        if(value != 0){
            var min = FitbitMINData[i]["activities-steps-intraday"]
            ["dataset"][j]["time"];
            var temp_date = date_time + "␣" + min;
            var unix_date_time = Date.parse(temp_date);
            times_dict["starting_time"] = unix_date_time;
            var min_in_millisec = 60000;
            unix_stop_time = unix_date_time + min_in_millisec;
            times_dict["ending_time"] = unix_stop_time;
            times.push(times_dict);
      }
        }
}
```

The response from Fitbit Steps is as described in Appendix A.3. On the frontend the response data is stored in the variable FitbitSleepData. Furthermore, the convert function accesses the values in the JSON for sleep start time and stop time in the date. The sleep from Fitbit is represented in milliseconds. However, one need to subtract one hours or 360 0000 milliseconds to adjust to the timeline. The function uses Date.parse() to parse the time into UNIX time.

```
for(var i=0; i < FitbitSleepData.length;i++){
        for(var j=0; j < FitbitSleepData[i]["sleep"].length;j++){
                var times_dict = {};
                var date_time = FitbitSleepData[i]["sleep"][j]
```

Service	Data Granularity
RunKeeper	Seconds
Fitbit Sleep	Milliseconds
Fitbit Steps	1 min or 15 min

Table 5.7: Web services represent data

```
                        ["startTime"];

                        var unix_date_time = Date.parse(date_time);
                        unix_date_time = unix_date_time - 3600000;

                        var sleep_milli = FitbitSleepData[i]["sleep"][j]
                        ["duration"];

                        var unix_stop_time = unix_date_time + sleep_milli;

                        times_dict["starting_time"] = unix_date_time;
                        times_dict["ending_time"] = unix_stop_time;
                        times.push(times_dict);
                        }
}
```

5.2.5 Access Control

One of the main features of Láhttu is offering access control over their personal data, by implementing an ACL. One list contains every other end-user the end-user has given permission to access her personal data. The other list contains whose personal data the end-user has access to. When an end-user gain access to another end-user personal data, this end-user gains only principle of least privilege, meaning that the end-user that is given privileges to access on behalf, only have the essential privilege to perform the access, nothing more [38].

Each end-user is assigned a unique ID-number, and with this ID-number end-users of Láhttu can share their ID-number for giving other end-users permission.

For instance, say there is a coach C and an athlete A, C wants to access A personal data for greater insight and overview.

Then C gives her ID-number to A, then A enters while signed into Láhttu C the ID-number for the given permission to C. A can now see that C has permission to her personal data from the web-services through Láhttu. Now C can login to Láhttu an see what that A may have permission and access to, and then chose to access personal data on their behalf. When C chooses to access personal data of A, Láhttu executes a SQL query on the ACL table in the database for permission check. A can whenever she desires revoke and remove the availability to access her personal data. Then, a delete SQL query is performed on the database.

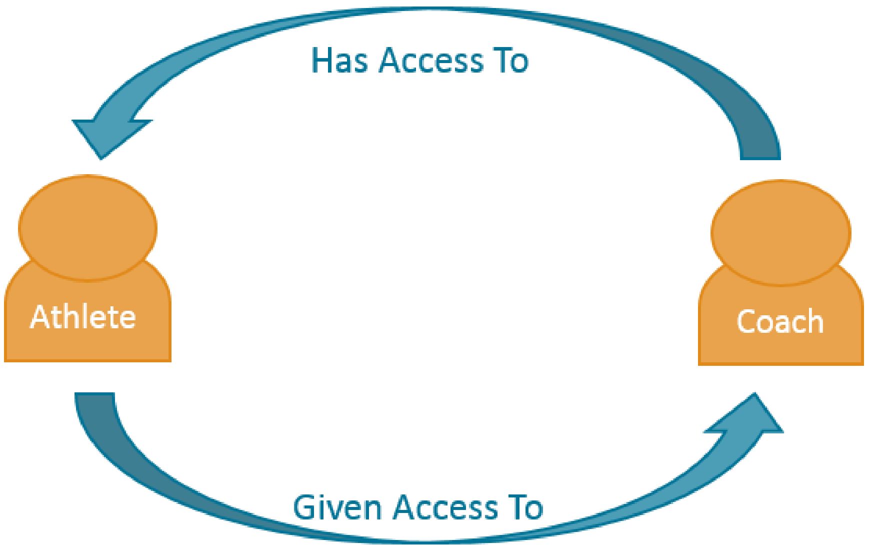


Figure 5.10: Give access

When an end-user wants to give another end-user permission to access her personal data, she inserts the other end-users ID-number in the insertion field and clicks on the add button. The frontend sends an HTTP request containing a URL with an query string with the other end-users ID number as data. On the backend, a checks first done for duplication of the access privilege in the ACL table of the database. If the other end-user is currently in the table with the access permission to the end-user, the backend returns immediately and the frontend pops up an alert message saying that the other end-user has already access permission.

If there is no duplication in the table, the backend executes a SQL insert query with ID number of who has given access from and who has been granted access to. The name of the end-user that has been given permission is returned and appended the ACL list in the GUI main page of Láhttu. In addition, the backend checks if the end-user has inserted an ID-number tries to add an ID number that is non-existing in the system. If the ID number is not current, the backend returns and an alert message pops up.

An end-user can revoke this access to their own personal data any time. When the end-user clicks on the delete button. The frontend sends an HTTP request containing a URL with an query string with the other end-users ID number for which access is to be revoked. The backend executes a SQL delete query with ID number of whom has given access from and who has been granted access to for deleting the row in the table. The GUI removes that end-user from the ACL list over whom one has given access to.

When an end-user wants to access other end-users personal data, the end-user clicks on the name of the desire end-user from the ACL over who one has access to. Then a field in the GUI will show, with the opportunity to selected a range between to dates.

5.3 Summary

In this chapter we have presented our system, Láhttu, which is an intermediate between end-users and the activity-tracking web services. The many implementation details have been descried in this chapter.

6
Evaluation and Results

This chapter presents an evaluation of Láhttu. Several important aspect of this thesis will also be discussed.

6.1 Methodology and Methods

Stated in section 1.2, this thesis follows the *Design* paradigm. The design process for the thesis is using *prototyping*[33], which is the best approach when not all details are identified and defined. This will make the process more abstract and non-rigorous, exploring ideas more easily, when constructing a system with requirements and specifications.

Firstly, the prototype process starts with an idea, which is an abstract description of a problem and defining loosely a thinkable solution to the problem. The idea is what will shape and form the system.

Then we need to define the design and architecture with all the components and their functionality. In addition, more details for the system will be defined in this stage, taking into consideration requirements and specifications. The next stage is to implement the design, which is itself the prototype system. Although the limit prototype system needs improvements, it provides a basic solution to the idea. However, performing experiments on the prototype system provides empirical insights, which validates the features and functionality.

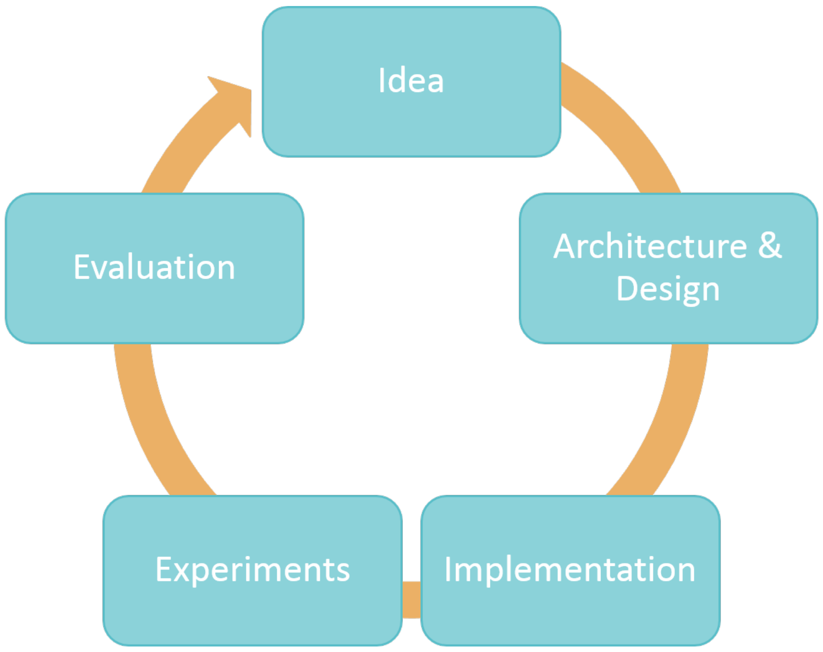


Figure 6.1: The prototype paradigm.

Much experience is gained through this stage.

These steps are iterated over, until a system is more properly defined and finished. If a good given idea or design is not enough, one can start all over again or move back to a previous step.

As stated in the problem definition in Section 1.1, the prototype system should be evaluated through *proof of concept*, meaning that, testing and evaluation is done to investigate if the prototype system is feasible. Hence, that the prototype system fulfills the stated problem definition.

6.2 Proof of Concept

The main idea with the prototype system is to give end-user insight and control over of their personal data on the various of web-services. From listings in Appendix A, we see that personal data can be retrieved from heterogeneous, activity-tracking web-services.

In addition, through the GUI, presented in Chapter 5 and the timeline illustrated in Figure 6.2, we see shows that the prototype system present, and consolidates personal data from Appendix A.

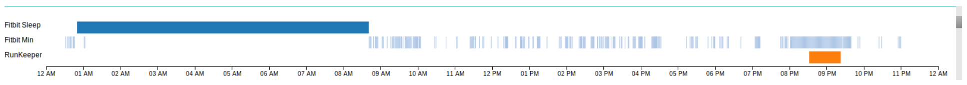


Figure 6.2: Timeline of the personal data retrieve in the Appendix A

Taking a closer look at the timeline illustrated in Figure 6.2, that the granularity of the personal data in optimal for insight and integrate into a homogeneous presentation.

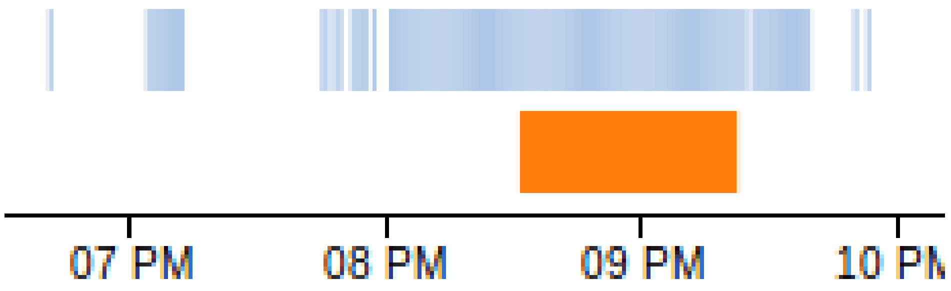


Figure 6.3: Closer look at the timeline in Figure 6.2

The prototype system verifies and proves through proof of concept in the form of testing the system, that it is feasible.

6.3 Experiments

6.3.1 Experimental Setup

All experiments were conducted on the following hardware and software:

- A Dell Precision 390 with processor 4x Intel(R) Core(TM)2 Quad CPU Q6600 @2.40 GHz.
- 4 GB DDR2 RAM and ATA SAMSUNG HD501LJ 500 GB.
- Operating system is Linux Mint 15 Olivia.
- Láhttu is tested on the web browsers Google Chrome and Mozilla Firefox with the URL Localhost on port 8080.

End-Users:	1	10	100	1000	10.000
One day	2	20	200	2000	20.000
One Month	60	600	6000	60.000	600.000
Three Month	180	1800	18.000	180 000	1800000
Half year	365	3650	36500	365000	3650000
A year	730	7300	73000	730000	7300000
Two year	1460	14600	146000	1460000	14600000
Few year	3650	36500	365000	3650000	36500000
Ten year	7300	73000	730000	7300000	73000000

Table 6.1: Correlation between number of end-user and required number of API requests.

In addition, for testing one need to create a test account at RunKeeper and Fitbit. The personal data used is our own personal data.

6.3.2 Estimation of requests

We are interested in a scenario where a researcher wants to collect and analyze large amounts of personal end-user data. In an epidemiological study done through the Tromsø Study [17], if a researcher would like to use step/everyday activity and sleep for the case study, the researcher would estimate the time it would take to retrieve all these data. Given for instance, an experiment with a time period of 10 years and 10 000 test end-users. The number of requests depending on the numbers of end-users and number of days is illustrated in the Table 6.1. The researcher would then request the Fitbit API 73 million times for all the personal data she would need for the research. If the researcher takes only one end-user at a time, she would request all 10 years of personal data for only one end-user before continuing on to the next end-user in the case study. With the limitation of 150 requests per end-user per hour, 10 years of personal data from Fitbit would take 48-49 hours per end-user. Requesting personal data for 10 000 end-users a time of 485 000 hours, 20 208 days or 55 years. Table 6.1 takes requests for the end-users with both details for sleep and steps through the Fitbit API-Get-Sleep and Fitbit API-Get-Intraday-Time-Series.

A successful request to the web-services returns the response code 200- OK. If the limit rate of the end-user is exceeded, the response code 409 - Conflict, is returning with a message telling that one must wait until the next hour before continued requests to the web service can be issued.

Another improvement could be for the researcher to issue requests to the web service in parallel for different end-users. Executing several requests in parallel would decrease time extremely. Another solution could be to divide the request into smaller portions. For instance, taking a half a year of one end-user, counting with the next end-user half a year until all 10 000 end-users personal data is retrieved, basically context switching between the end-users. The researcher can request cache of Láhttu for the same information, for better latency, which we will experiment next.

6.3.3 Latency

The latency will depend much on the network and processing of server implementation for the web services. Another factor that is worth mentioning is that the request will be sent from Tromsø, Norway and the web services servers are in the USA.

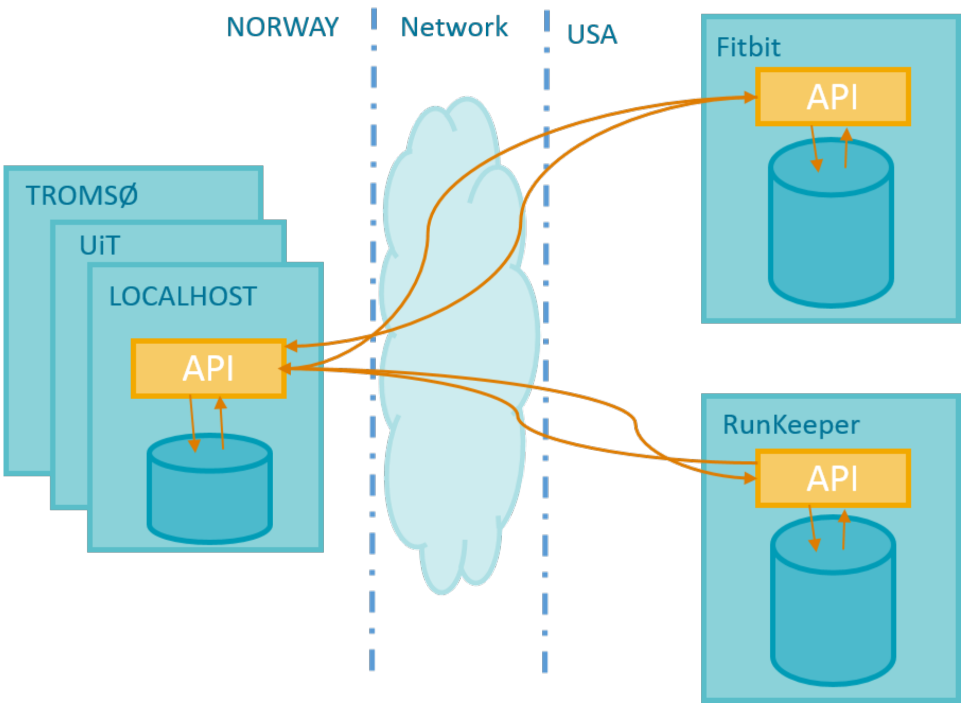


Figure 6.4: Latency figure

The end-to-end latency would have been different if one were testing from another country. Taking into consideration that the server load at the web-services can affect the time. The same range are tested on the same dates, for accuracy.

The Python function time.time() returns the time the request takes in seconds

Fitbit	One day	One Week	One Month	Three Months	Six months	a Year
Steps	0.87051	6.90366	26.59282	76.22344	156.95865	297.04248
Sleep	0.88108	7.02704	27.15323	73.04234	147.33407	307.59995

Table 6.2: The end-to-end latency from the Fitbit Web-serivce.

Cache	One day	One Week	One Month	Three Months	Six months	a Year
Steps	0.00010	0.00031	0.00057	0.00240	0.00449	0.01012
Sleep	0.00001	0.00007	0.00085	0.00146	0.00261	0.00616

Table 6.3: The end-to-end latency from cache.

as a floating point number. In this context, latency is measured from the backend side and roundtrip back to the backend, not from when an end-user clicks on the request button on the frontend GUI. Another aspect is that while testing only one web-service is tested at a time, so the results would not be disturbed or polluted.

```
sum_time = 0
start = time.time()
#request goes here
end = time.time()

total_request_time = start - end
sum_time = sum_time + total_request_time
```

When there are several dates to request, the measuring sums the time in a variables for holding the time.

Every request to Fitbit takes roughly 0.85 seconds, estimated from the results in Table 6.2. Estimating that without any limitation from the web service or hardware specification limitations, the researcher would use almost 2 years for acquisition all the personal data for the study. Latencies are Shown in Table 6.4.

Every request to the cache of Láhttu takes roughly 0.00005 seconds. The researcher would for the same acquisition of the personal data use approximately an hour. Table 6.5 illustrates that the cache is an extremely improvement for big data acquisition. However, the problem in practice is that the cache is limited in size and old data will have been evicted. Table 6.5 and Table 6.4 initially of the Table 6.1, for the number of request the researcher would needed for the case study. Although, in practice the cache is limited in size, this showed that

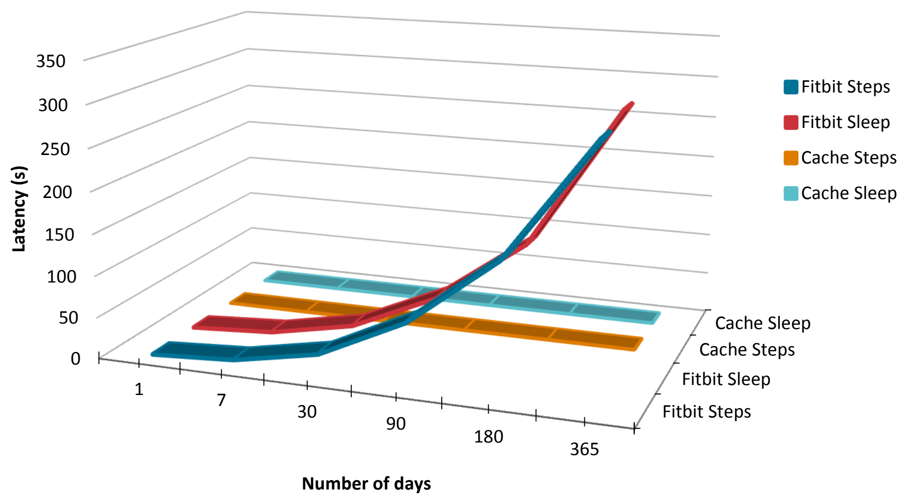


Figure 6.5: Latency Chart for table 6.2 and table 6.3

WEB SERVICE	1	10	100	1000	10.000
One day	1.7	17	170	1700	17 000
One Month	51	510	5100	51000	510.000
Three Month	153	1530	15300	153 000	1530000
Half year	310.25	3102.5	31025	310250	3102500
A year	620.5	6205	62050	620500	6205000
Two year	1241	12410	124100	1241000	12410000
Few year	3102.5	31025	310250	3102500	31025000
Ten year	6205	62050	620500	6205000	62050000

Table 6.4: Seconds for acquisition personal data from the web service

storing the personal data at the system would increase the acquisition of data extremely.

6.4 Evaluation of non-functional requirements

The non-functional requirements were defined in the Section 3.2. In context of the implementation of Láhttu these requirements will now be discussed.

CACHE	1	10	100	1000	10.000
One day	0.00010	0.00100	0.01000	0.10000	1
One Month	0.00300	0.03000	0.30000	3	30
Three Month	0.00900	0.09000	0.90000	9	90
Half year	0.01825	0.18250	1.82500	18.2500	182.50
A year	0.03650	0.36500	3.6500	36.50	365
Two year	0.073	0.7300	7.300	73	730
Few year	0.18250	1.82500	18.2500	182.50	1825
Ten year	0.365	3.6500	36.50	365	3650

Table 6.5: Seconds for acquisition personal data from the cache.

6.4.1 Scalability and Extensibility

The backend implementation is structured using classes, where each feature is encapsulated in a class. Through having separate classes for each feature, extensibility is increased, because one can add more classes into the system. Also, the class structure is argues for decreased dependability, because the classes do not depend on each other. In addition, on the frontend, one can apply new features simplify adding a new function in JavaScript.

One main factor that can affect scalability is the storage of the end-user credentials. For retrieving personal data from the web-services, credentials are required, and accessing this on disk may be a bottleneck. In a scenario where the backend is scaled horizontally to serve more traffic, the storage would become the bottleneck as the credentials for the end-user is needed for authenticating with the web-services. General if there are some share variables or resources, there could be a bottleneck and affect scalability.

Having adopted the REST design principles, described in Section 2.4.4 at the backend API increases scalability. The backend is stateless in its design, easing replicating of state between servers. It is layered so that a load balancer can be introduced without any problems. It is autonomous to the frontend it interacts with and the frontend can interact with any backend server.

6.4.2 Fault Tolerance and Availability

The backend cache provides an extra replica of the personal data for the end-users. If the web service is unavailable, the system serves as a potential extra failover handler by having replicas of the personal data for the end-users. Machine failures or network partitioning will lead to unavailability for end-users until the machine and the backend is up and running again.

6.4.3 Security and Privacy

Láhttu offers privacy to the end-users of the system for their personal data, but Láhttu cannot vouch for the privacy of the web-services. This can be illustrated with one example:

The web-service Strava, wants to sell end-users personal running (GPS) and cycle data to city planners. This can be used by city planners to determine popular routes and traffic hot-spots by tracking patterns over which streets have the highest traffic of end-users. Strava has already entered a deal with Oregon, London and Orlando[25].

A question arises if the web-services selling end-user data[22]. Fitbit states in their policy that they may sell the personal data for an end-user: "At times Fitbit may make certain personal information available to strategic partners that work with Fitbit to provide services to you."[1].

Although, they states that it is for the greater good and making the world a better place. Selling the end-users personal data can backfire at the end-user. Given that an end-user applies and wants to purchase a life insurance, she can be denied the life insurance because the insurance company has the self-tracking data for that end-user, showing that the end-user is in bad physical health, inactive, and obese.

Security is important and necessary. In case of sensitive end-user personal data, security is critical. One can discuss in which degree the personal data Láhttu have support for is critical, but is not that critical as for PHR for end-users. Anyway, personal data should in any case be encrypted. There is several methods for encryption, symmetric-key algorithms, AES and DES, and asymmetric-key algorithms, RSA and elliptic curves.

6.5 Summary

In this chapter, we have stated and evaluated that the prototype system Láhttu is feasible through proof of concept. Retrieval of personal data, then presenting and consolidating the personal data for the end-user.

1. https://www.fitbit.com/privacy

In addition, we have learned that having personal data available through Láhttu increase latency and data acquisition extremely in a bigger context. Although, primarily the cache will not hold that much personal data, one sees opportunities to use the system for core or case study for public health as stated in Section 1.3.

7 Related Work and Discussion

This chapter presents and discusses some related work to the thesis.

7.1 Related Work

The eco-system of all the web-services with their connections between each-other can be seen as related work to some degree from the Survey in Chapter 2.5. Here some other systems will be included and briefly discussed.

7.1.1 Microsoft HealthVault

Microsoft HealthVault is Personal Data Vault(PDV), as mentioned briefly in section 2.2, for end-users with their health related personal data. The personal data include PHR, fitness goals, pharmaceuticals, health measurements and testing. The HealthVault supports interoperability for end-users so they can upload personal data from a small number of compatible devices and system. These devices include body scales such as the Withings WS-50 and Fitbit Aria. Furthermore, one can also connect Fitbit Flex for steps and sleep. The systems and devices that have the opportunity to connect to the HealthVault are very rigorous, according to the HealthVault only total 150 system and application have this opportunity.

Security and privacy are top concerns for Microsoft in HealthVault [1]. The privacy policy for HealthVault states that the end-user account and the corresponding personal data belongs to the end-users and is not Microsoft property. Microsoft states that it will not use the personal data unless explicitly asking the end-user for that permission. One of the main concerns in the context of security of the end-users system is the management of end-users and their credentials for gaining access to the system. One of the weak points is that end-users are making too weak and simple passwords that are easy to hack [2]. HealthVault requires strong passwords, typically a password with a minimum length in characters including upper and lower case letters and numbers. If the password is not strong enough, the password will be rejected. In addition, Microsoft isolates and logs all traffic to the separate network and data silos holding the personal data of HealthVault. Also, Microsoft limits employees that have the necessary permissions to perform essential operations on HealthVault.

End-users of HealthVault can grant access to other end-users of HealthVault to their account[2]. There are different levels of access privileges, either read, read-write or time limited access. The end-user decides the granularity of sharing. This can be a single journal or all the personal information stored at HealthVault. Discarding records is done instantaneously, but Microsoft will hold the discarded record in a private cache for 90 days in case of malicious or mistaken deletion of the record.

Compared to Láhttu, HealthVault handles more sensitive personal data, and needs a higher level of security. Furthermore, HealthVault is more restricted than Láhttu with regard to systems and applications that can to connect the HealthVault than Láhttu. Given that one wants to connect, a system, it will take time to integrate it into HealthVault. Here is the one of the main motivation for Láhttu; taking control over which web-services or data source to connect and integrate.

7.1.2 Consolidating Personal Data Platforms and Systems

The Locker Project [3], is a cloud storage of personal data. These personal data can be end-users photos, places a end-users have visited, links shared by the end-user, how the end-user have communicated with and contact details. Privacy for

1. http://www.pcmag.com/article2/0,2817,2191920,00.asp
2. https://account.healthvault.com/help.aspx?topicid=PrivacyPolicy
3. http://lockerproject.org/

the end-users of The Locker Project is able to decide and take control over what they share with social networks. The Locker Project is open source software available at Github [4], and is currently under development.

Another system is Personal[5] that is an PDV with cloud storage. All end-user information can be stored at Personal. This Information can be passwords to vary systems and applications, ID-numbers, credit card information, end-user fill out information, addresses and much more. The PDV for the end-user at Personal is encrypted with a 256-bit AES encryption and RSA 2048 asymmetric key encryption, also with default 128-bit SSL encryption. In addition, only the end-user knows the password to Personal. However, if an end-user forgets password to Personal and need to reset the password, all the encrypted information will be delete for the end-user. Privacy of Personal states that the end-users own the data and only the end-user can share their personal data with others.

Both The Locker Project and Personal are part of "Personal Data Ecosystem Consortium" [6]. The main purpose is to develop systems and tools for benefiting personal control over end-users personal data.

4. https://github.com/LockerProject/Locker
5. https://www.personal.com/
6. http://personaldataecosystem.org/

8 Conclusion

This chapter presents the achievements of the thesis and offers some concluding remarks. Furthermore, possible future work for the thesis is outline.

8.1 Achievements

This thesis describes the design and implementation of the Láhttu prototype system that gives insight for end-users in their personal data generated in several heterogeneous web-services. The problem definition stated the following, in Section 1.1:

> *This thesis will explore system issues related to the use of personal data from activity tracking web-services. The goal is to architect and build a prototype system that provides end-users an overview of and improve insight into their online personal data.*

In the ecosystem of all the system and devices available for the end-users, losing control over where the personal data is stored is not optimal. Láhttu offers an end-user to use range queries for retrieval personal data from the different, heterogeneous web-services that Láhttu supports. Another feature Láhttu offers is to allow other end-users of the system to retrieval personal data on behalf of other end-user, after been granted that permission.

The main reason end-users want to track themselves is to quantify themselves, and not just assume they are active but getting a proof of just how active, they are. An overview and insight into personal data can inspire and motivate a person to change their lifestyle. Hence, due to increasing obesity amongst people in the world, more particular in the western part of the world[1], self-tracking and "Quantified Self" can make the world a better place. However, the large eco-system of self-tracking web-services provide little interoperability and opportunity to gather personal data from heterogeneous web-services.

8.2 Concluding Remarks

This thesis has shown that by introducing an intermediate system, Láhttu, it is possible to consolidate and present the heterogeneous personal data to the end-user for increased insight over their physical activity level for activity-tracking web-services.

One of the core problem under development of the system was getting over the limitations of what each API for the web-services offers for third-parties. In addition, integration and interoperability issues such as different representation and format and the personal data must be considerate.

Through evaluation, estimation and experiments shows that the prototype system, Láhttu, can be used in bigger context for case study involving public health through personal data studying and analysing.

8.3 Future Work

Consequently, there are several improvements and functionality that can be expand the current prototype system:

Security: Improve security aspect of the system.

Multiple, concurrent end-users: The storage has support for multiple end-users. However, the system currently does not support concurrent end-users. Changing it with adding session event or cookies to the system.

Support for more web-services: Through system extensibility, it is possible

1. http://www.vg.no/nyheter/utenriks/nesten-en-av-tre-i-verden-veier-for-mye/a/10124327/

to expand with more web-services. Also, one can look into integrating professional system, such as ZXY.

GUI: The systems GUI can be improved in structure. In addition, the representation of larger ranges in the timeline could be better.

Duplication: An end-user may use two web-services simultaneously, which leads to having the same activity recorded on two different web-services. Although, this increases redundancy, one issue is that summaries of the end-users activity level will be inconsistent. The system currently supports in the timeline GUI, that the end-user is present with these consolidated personal data. A future approach could be to have computing that personal data prioritized over another source.

Analyze: The system currently can be used as an data acquisition component for other system that performs analyses on the personal data.

Deployment: The system is not deployed, one need to set up a server with the system running on it.

A JSON Reponse from Web-services

Response data in JSON from the webs-services in a range query of 2014-03-04.

A.1 RunKeeper

```
{
    "items": [
        {
            "entry_mode": "API",
            "total_calories": 586,
            "has_path": true,
            "uri": "/fitnessActivities/313921905",
            "source": "RunKeeper",
            "total_distance": 8210.13869291205,
            "duration": 3063.829,
            "type": "Running",
            "start_time": "Tue, 4 Mar 2014 20:31:39"
        }
    ],
    "size": 1
}
```

A.2 Fitbit Step

Fitbit Steps: The whole reponse generated around 6000 lines of JSON, cutting big parts away.

```
{
    "activities-steps-intraday": {
        "datasetType": "minute",
        "datasetInterval": 1,
        "dataset": [
            {
                "value": 0,
                "time": "00:00:00"
            },
            {
                "value": 0,
                "time": "00:01:00"
            },
            {
                "value": 0,
                "time": "00:02:00"
            },
            {
                "value": 0,
                "time": "00:03:00"
            },
            {
                "value": 0,
                "time": "00:04:00"
            },
            {
                "value": 0,
                "time": "00:05:00"
            },
            {
                "value": 0,
                "time": "00:06:00"
            },
            {
                "value": 0,
                "time": "00:07:00"
            },
            {
                "value": 0,
                "time": "00:08:00"
            },
            {
                "value": 0,
                "time": "00:09:00"
            },
  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
            {
                "value": 156,
                "time": "21:02:00"
            },
            {
                "value": 164,
                "time": "21:03:00"
            },
            {
                "value": 169,
                "time": "21:04:00"
            },
            {
                "value": 174,
                "time": "21:05:00"
            },
            {
                "value": 162,
                "time": "21:06:00"
            },
            {
                "value": 175,
                "time": "21:07:00"
            },
            {
                "value": 165,
                "time": "21:08:00"
            },
            {
                "value": 173,
                "time": "21:09:00"
            },
{
    "value": 168,
    "time": "21:10:00"
},
{
    "value": 167,
    "time": "21:11:00"
},
{
    "value": 173,
    "time": "21:12:00"
},
{
    "value": 172,
    "time": "21:13:00"
},
{
    "value": 174,
    "time": "21:14:00"
},
{
    "value": 175,
    "time": "21:15:00"
},
{
    "value": 175,
    "time": "21:16:00"
},
{
    "value": 172,
    "time": "21:17:00"
},
{
    "value": 173,
    "time": "21:18:00"
},
{
    "value": 173,
    "time": "21:19:00"
},
{
    "value": 167,
    "time": "21:20:00"
},
{
    "value": 176,
    "time": "21:21:00"
},
{
    "value": 176,
    "time": "21:22:00"
},
{
    "value": 134,
    "time": "21:23:00"
},
{
    "value": 119,
    "time": "21:24:00"
},
{
    "value": 0,
    "time": "21:25:00"
},
{
    "value": 131,
    "time": "21:26:00"
},
{
    "value": 107,
    "time": "21:27:00"
},
{
    "value": 121,
    "time": "21:28:00"
},
{
    "value": 122,
    "time": "21:29:00"
},
{
    "value": 135,
    "time": "21:30:00"
},
{
    "value": 135,
    "time": "21:31:00"
},
        {
            "value": 114,
            "time": "21:32:00"
        },
        {
            "value": 104,
            "time": "21:33:00"
        },
        {
            "value": 127,
            "time": "21:34:00"
        },
        {
            "value": 101,
            "time": "21:35:00"
        },
    ..................
        {
            "value": 0,
            "time": "23:53:00"
        },
        {
            "value": 0,
            "time": "23:54:00"
        },
        {
            "value": 0,
            "time": "23:55:00"
        },
        {
            "value": 0,
            "time": "23:56:00"
        },
        {
            "value": 0,
            "time": "23:57:00"
        },
        {
            "value": 0,
            "time": "23:58:00"
        },
        {
            "value": 0,
            "time": "23:59:00"
            }
        ]
    },
    "activities-steps": [
        {
            "value": "19282",
            "dateTime": "2014-03-04"
        }
    ]
}
```

A.3 Fitbit Sleep

```
{
    "sleep": [
        {
            "logId": x,
            "isMainSleep": true,
            "minutesToFallAsleep": 65,
            "awakeningsCount": 12,
            "minutesAwake": 151,
            "timeInBed": 471,
            "minutesAsleep": 244,
            "awakeDuration": 3,
            "efficiency": 62,
            "startTime": "2014-03-04T00:50:00.000",
            "restlessCount": 12,
            "duration": 28260000,
            "restlessDuration": 224,
            "minuteData": [
                {
                    "value": "2",
                    "dateTime": "00:50:00"
                },
                {
                    "value": "2",
                    "dateTime": "00:51:00"
                },
                {
                    "value": "2",
                    "dateTime": "00:52:00"
                },
{
    "value": "2",
    "dateTime": "00:53:00"
},
{
    "value": "2",
    "dateTime": "00:54:00"
},
{
    "value": "2",
    "dateTime": "00:55:00"
},
{
    "value": "2",
    "dateTime": "00:56:00"
},
{
    "value": "2",
    "dateTime": "00:57:00"
},
{
    "value": "2",
    "dateTime": "00:58:00"
},
{
    "value": "2",
    "dateTime": "00:59:00"
},
{
    "value": "2",
    "dateTime": "01:00:00"
},
{
    "value": "2",
    "dateTime": "01:01:00"
},
{
    "value": "2",
    "dateTime": "01:02:00"
},
{
    "value": "2",
    "dateTime": "01:03:00"
},
{
    "value": "2",
    "dateTime": "01:04:00"
},
{
    "value": "3",
    "dateTime": "01:05:00"
},
{
    "value": "3",
    "dateTime": "01:06:00"
},
{
    "value": "2",
    "dateTime": "01:07:00"
},
{
    "value": "2",
    "dateTime": "01:08:00"
},
{
    "value": "2",
    "dateTime": "01:09:00"
},
{
    "value": "2",
    "dateTime": "01:10:00"
},
{
    "value": "2",
    "dateTime": "01:11:00"
},
{
    "value": "2",
    "dateTime": "01:12:00"
},
{
    "value": "2",
    "dateTime": "01:13:00"
},
{
    "value": "2",
    "dateTime": "01:14:00"
},
            {
                "value": "2",
                "dateTime": "01:15:00"
            },
            {
                "value": "2",
                "dateTime": "01:16:00"
            },
            {
                "value": "3",
                "dateTime": "01:17:00"
            },
            {
                "value": "2",
                "dateTime": "01:18:00"
            },
            {
                "value": "2",
                "dateTime": "01:19:00"
            },
..........................................

            {
                "value": "1",
                "dateTime": "08:29:00"
            },
            {
                "value": "2",
                "dateTime": "08:30:00"
            },
            {
                "value": "2",
                "dateTime": "08:31:00"
            },
            {
                "value": "2",
                "dateTime": "08:32:00"
            },
            {
                "value": "2",
                "dateTime": "08:33:00"
            },
            {
                "value": "2",
                    "dateTime": "08:34:00"
                },
                {
                    "value": "2",
                    "dateTime": "08:35:00"
                },
                {
                    "value": "2",
                    "dateTime": "08:36:00"
                },
                {
                    "value": "2",
                    "dateTime": "08:37:00"
                },
                {
                    "value": "2",
                    "dateTime": "08:38:00"
                },
                {
                    "value": "2",
                    "dateTime": "08:39:00"
                },
                {
                    "value": "2",
                    "dateTime": "08:40:00"
                }
            ],
            "awakeCount": 0,
            "minutesAfterWakeup": 11
        }
    ],
    "summary": {
        "totalTimeInBed": 471,
        "totalMinutesAsleep": 244,
        "totalSleepRecords": 1
    }
}
```

Bibliography

[1] Connections counter: The internet of everything in motion. cisco.com, July 2013.

[2] Howard Baldwin. Passwords are the weak link in it security. computerworld.com.

[3] Doug Beaver, Sanjeev Kumar, Harry C. Li, Jason Sobel, and Peter Vajgel. Finding a needle in haystack: Facebook's photo storage. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, pages 1–8, Berkeley, CA, USA, 2010. USENIX Association.

[4] D. E. Comer, David Gries, Michael C. Mulder, Allen Tucker, A. Joe Turner, and Paul R. Young. Computing as a discipline. *Commun. ACM*, 32(1):9–23, January 1989.

[5] European Commission. Attitudes on data protection and electronic identity in the european union. *SPECIAL EUROBAROMETER 359*, 2011.

[6] European Commission. Safeguarding privacy in a connected world a european data protection framework for the 21st century. COM(2012)9, Sep 2012.

[7] Tony Danova. Morgan stanley: 75 billion devices will be connected to the internet of things by 2020. businessinsider.com.

[8] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008.

[9] Peter Dizikes. Sports analytics: a real game-changer. *MIT News*, Mar 2013.

[10] Roy Thomas Fielding. Architectural styles and the design of network-based software architectures. Doctoral dissertation, University of California, Irvine, 2000.

[11] Frank P. Grad. The preamble of the constitution of the world health organization. Bulletin of the World Health Organization 2002, 80 (12), 2002.

[12] Pål Halvorsen, Simen Sægrov, Asgeir Mortensen, David K. C. Kristensen, Alexander Eichhorn, Magnus Stenhaug, Stian Dahl, Håkon Kvale Stensland, Vamsidhar Reddy Gaddam, Carsten Griwodz, and Dag Johansen. Bagadus: An integrated system for arena sports analytics: A soccer case study. In *Proceedings of the 4th ACM Multimedia Systems Conference*, MMSys '13, pages 48–59, New York, NY, USA, 2013. ACM.

[13] John A. Hoxmeier, Ph. D, and Chris Dicesare Manager. System response time and user satisfaction: An experimental study of browser-based applications. In *Proceedings of the Association of Information Systems Americas Conference*, pages 10–13, 2000.

[14] Qi Huang, Ken Birman, Robbert van Renesse, Wyatt Lloyd, Sanjeev Kumar, and Harry C. Li. An analysis of facebook photo caching. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, SOSP '13, pages 167–181, New York, NY, USA, 2013. ACM.

[15] Pål Haloversen Håvard D. Johansen, Svein Arne Pettersen and Dag Johansen. Combining video and player telemetry for evidence-based decisions in soccer. *Regional Centre for Sport, Exercise and Health - North*, 2013.

[16] FitneessKeper Inc. Health graph api. http://developer.runkeeper.com/healthgraph.

[17] Bjarne K Jacobsen, Anne Elise Eggen, Ellisiv B Mathiesen, Tom Wilsgaard, and Inger Njølstad. Cohort profile: The tromsø study. *International Journal of Epidemiology*, 2011.

[18] Dag Johansen and Joseph Hurley. Overlay cloud networking through meta-code. In *Proceedings of the 2011 IEEE 35th Annual Computer Software and Applications Conference Workshops*, COMPSACW '11, pages 273–278, Washington, DC, USA, 2011. IEEE Computer Society.

[19] Dag Johansen, Magnus Stenhaug, Roger Bruun Asp Hansen, Agnar Christensen, and Per-Mathias Høgmo. Muithu: Smaller footprint, potentially larger imprint. In *Proc. of 7th International Conference on Digital Information Management*, pages 205–214. IEEE, August 2012.

[20] Håvard D. Johansen, Wei Zhang, Joseph Hurley, and Dag Johansen. Man-

agement of body-sensor data in sports analytic with operative consent. In *of the 2014 IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*. IEEE, April 2014.

[21] Dogulas Laney. 3d data management: Controlling data volume, velocity and variety. Gartner, Feb 2001.

[22] Dana Liebelson. Are fitbit, nike and garmin planning to sell your personal fitness data? motherjones.com, Jan 2014.

[23] Lori MacVittie. The impact of ajax on the network. *F5 Networks, Inc*, 2007.

[24] ERIK MALINOWSK. How runkeeper could become the facebook of fitness, 2011.

[25] May. Strava, popular with cyclists and runners, wants to sell its data to urban planners. wsj.com, 2014.

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

[27] Jakob Nielsen. *Usability Engineering*, chapter Interactive technologies. Morgan Kaufmann Publishers, 1993.

[28] The Norwegian Directorate of Health. "kunnskapsgrunnlag fysisk aktivitet. innspill til departementets videre arbeid for økt fysisk aktivitet og redusert inaktivitet i befolkningen.". helsedirektoratet.no, February 2014.

[29] Office of Science and Technology Policy. Obama administration unveils big data initative announces 200 million dollar in new r and d investement. whitehouse, March 2012.

[30] Marily Oppezzo and Daniel L. Schwartz. Give your ideas some legs: The positive effect of walking on creative thinking. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, April 2014.

[31] World Health Organization. Health topics obesity, 2013.

[32] D.A. Patterson and J.L. Hennessy. *Computer Organization and Design: The*

Hardware/Software Interface. The Morgan Kaufmann Series in Computer Architecture and Design. Elsevier Science, 2013.

[33] Roger S Pressman. *Software Engineering: A Practitioner's Approach, 7 edition*. R. S. Pressman & Associates, Inc, 2010.

[34] N. Ramanathan, F. Alquaddoomi, H. Falaki, D. George, C. Hsieh, J. Jenkins, C. Ketcham, B. Longstaff, J. Ooms, J. Selsky, H. Tangmunarunkit, and D. Estrin. ohmage: An open mobile system for activity and experience sampling. pages 203–204, 2012.

[35] Alex Rodriguez. Restful web services: The basics, 2008.

[36] RICHARD N. TAYLOR ROY T. FIELDING. Principled design of the modern web architecture. ACM Transactions on Internet Technology, Vol. 2, No. 2, May 2002.

[37] Simen Sægrov, Alexander Eichhorn, Jørgen Emerslund, Håkon Kvale Stensland, Carsten Griwodz, Dag Johansen, and Pål Halvorsen. BAGADUS: An integrated system for soccer analysis (demo). In *Proc. of the ACM/IEEE International Conference on Distributed Smart Cameras*, November 2012.

[38] R.S. Sandhu and P. Samarati. Access control: principle and practice. *Communications Magazine, IEEE (Volume:32 , Issue: 9)*, 2002.

[39] Emily Singer. The measured life. *MIT Technology review*, June 2011.

[40] Ian Steadman. 'ibm's watson is better at diagnosing cancer than human doctors". wired magazine, www.wired.co.uk, Feb 2013.

[41] Melanie Swan. Sensor mania! the internet of things, wearable computing, objective metrics, and the quantified self 2.0. *Journal of Sensor and Actuator Networks*, 1(3):217–253, 2012.

[42] Andrew S. Tanenbaum. *Modern Operating Systems*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2007.

[43] Andrew S. Tanenbaum and Maarten van Steen. *Distributed Systems: Principles and Paradigms (2Nd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2006.

[44] S.V. Valvåg, Dag Johansen, and Åge Kvalnes. Cogset: a high performance mapreduce engine. *Concurrency and Computation: Practice and Experi-*

ence, 25(1):2–23, 2013.