UiO : Department of Informatics
University of Oslo

pmSys

Implementation of a digital Player Monitoring System

Thuc Tuan Hoang

Master's Thesis Spring 2015

pmSys

Thuc Tuan Hoang

Abstract

A football match can be determined by the smallest factors such as mood, however, but other factors as injuries can determine whether you place first or second. The teams with the least injuried players would have a better edge in reaching the top each season. Since the beginning of monitoring in football it has all been registered by hand using paper and pen. During the 21th century technology has been one of the best and most accurate helping hand any area within monitoring can get. Being able to process large amounts of data in split seconds has proven to be worth the investment in going digital when it comes to monitoring. On the basis of this, pmSys was created to enhance the power of processing personal data in real time.

In this master thesis we wanted to develop a system for both the football players and trainers to be able to register and follow up the submitted data in real time. By giving a team these tools we wish to constitute the small factor that can push any football team to the limits without going over the edge and into an injury nightmare.

Contents

List of Figures

List of Tables

List of Equations

Acknowledgement

I would like to thank my supervisor Pål Halvorsen, for all the advices you have given me the past two years. Thank you for providing me guidance, ideas, motivation, discussion and constructive feedback throughout this thesis. Your knowledge, commitment, hard work and positiveness are truly inspiring, and I am grateful for getting the chance to work with you on this project.

Furthermore, I would like to thank my best friends and collegues; Nguyen Cong Nguyen and Kennet Khang Vuong for working together with me on pmSys. Their commitment, hard work and positive attitude has constantly made me want to out do myself. I am really looking forward to the future, and all the future work we are going to do together!

I would also thank the Norwegian School of Sports Sciences, especially Håvard Wiig and Thor Einar Anderson for giving us the opportunity to create the monitoring tool needed for your PhD research. Thank you for sharing your expertise within sports medicine, your constructive feedback and discussion throughout this thesis. I would also thank everyone that have been there for us during our user studies, tested the application, the system and also providing us feedback!

Moreover, I would like to thank Håvard Johansen, Dag Johansen, Svein Arne Pettersen from University of Tromsø, as well as Carsten Grizwods and Håkon Kvale Stensland from Simula Research Lab for providing us with input and great insight.

Finally, I would like to thank my family and friends for their continuous morale support and encouraging words throughout my education. Especially my incredible girlfriend, Phuoc Hanh Thi Nguyen who has supported me throughout this whole process. Thank you for motivating, encouraging and for always being there for me!

Oslo, May 18th 2015
Thuc Tuan Hoang

Chapter 1

Introduction

1.1 Background

In recent years, information about a football player's physical state has been collected by only pen and paper. Crunching numbers into formulas by hand or using third party programs such as Excel to compute a report faster, still requires a lot of time. The idea is to add more advanced technology into the equation so that we can accomplish at least the same results in just a few seconds just by clicking on a button. Even though computing all the data is faster on a computer, formulas are needed to compute all the numbers. In our case, it is physics and well documented algorithms used for detection of strain and workload on the players. The formulas have to be implemented in code, but also thoroughly tested to ensure that everything is correct. From an informatics perspective, this is the perfect case of use where technology can be used as a quality assurance tool and giving output with customized quality for a small percent of the cost and time used.

During this master thesis, I have worked with two other master students, Cong-Nguyen Nguyen and Kennet Khang Vuong on the systems we have created. Our master thesis is based on the same core code, however, later in the master thesis timeline, each of our projects branched out into different focus fields. To make this system possible (with the sufficient medical background), we paired with PhD student Håvard Wiig and MD PhD PT Thor Einar Andersen with their project within health monitoring from a medical perspective [1]. Our part in this project involves creating a system that can ease the calculation and detect when a player's health is getting close to self-inflicting injury due to overtraining.

During spring 2014, four teams from the Norwegian Premier Football league "Tippeligaen" used using a very first version of the system using Ohmage to collect their health[1] data. This was meanwhile pmSys was under the early stages of development, by doing this we could get better feedback from the players on how Ohmage worked. Creating reports and graphs had to be done by hand due to a bug with the OpenCPU module in Ohmage. This meant that even though the information was collected through the Ohmage mobile application, it would still take a lot of manual labor for the coaches or any other person to create all the detailed reports perfectly. Discovering trends and abnormalities would be time consuming due to the time it takes to compile all the information by hand. In addition it is not possible to compile a detailed report where the information are updating itself all the time. With technology this can be done in just a matter of seconds. Discovering and alerting the correct people can make a huge difference before an injury occurs. Avoidance of injury can be translated into a player's value in the football market, the less the player is injured and performs well, the better is the value for both the player and the club. Investing millions of dollars in one player just to have him overextend his physique would cost a lot of

[1]It is important to note that we have not been working with health data in our thesis, but created tools for collecting health data.

money every day he is not actively performing as a football player.

Because of the complexity of Ohmage's current front-end, we decided to create pmSys. It originally started as an alternative front-end for Ohmage, however, during this thesis, pmSys evolved into it's own system by reusing Ohmage's REST API to manipulate the data the way we want it to be. By doing this, we can improve Ohmage and in addition present the data on a whole new level, i.e., making the system more approachable for both football players, medical staff and coaches.

1.2 Problem Definition

Since the beginning of monitoring health of football player has always been recorded by using pen and paper. *In this thesis, we wish to find out how can we create a digital monitoring system that optimizes the collection, the storage of data, the data analyzation and the visualization of a player's health data. How can a digital tool help a team towards their goal by monitoring their own players?.*

We will therefore address the main problems concerning monitoring health of football players. By using technology to speed up the process and to increase the detection rate of pre-injuries before it is too late. We will research and develop a health monitoring system, which we named, Player Monitoring System (pmSys). The system will be the main reporting tool for the football player, the coach and the medical staff.

For the players, a new mobile application has to created because of the lack of usability of Ohmage MWF [2], due to the nature of an athlete's schedule the mobile application has to be as easy as possible. Registration should take as little time as possible so that the focus can be channeled into performing at the maximum limit during training sessions.

Even though the players will be receiving a new tool to use, the coaches should not be using their time to enforce players to register health data. We will therefore create a web portal with rich features where intuitiveness should be one of the most important factor for using the system. The data will be presented in a way so that the medical staff and the coach can use it to monitor their players, which means that they will not be needed to work the data to receive the results they wish for.

Collection and storage of objective data requires objective devices that can record physical data from the players, and one of the possible ways to do this is to use third party devices such as hardware and wearables. These devices can provide a team the possibility of valuable data from a player daily life for closer analysis, for example during training sessions and outside of the training sessions.

The primary goal of this thesis is therefore to create a system for both the players and the coaches to replace the pen and paper. Everything can be processed and presented in the most effective way possible. Because of the time limit set for this thesis, we need to provide a stable and efficient implementation. By reusing the API back-end of Ohmage [3], pmSys will provide the users with a simple, but yet powerful tool to analyze and follow a player's progression throughout a season by detailed visualization of subjective and objective data.

1.3 Limitations

Since pmSys is more or less an extension of Ohmage, only the most basic descriptions and the features of Ohmage will be presented. We will go into how pmSys is using the Ohmage back-end as a supporting pillar of the system, and why we chose to use it instead of creating our own back-end. As for the security aspect, we will not provide any in depth description of the solutions pmSys is currently using, but rather scratch the surface of what security issues we can come across while implementing pmSys. We will also point out the flaws of Ohmage and how we implemented pmSys in order to fill the gaps and lack of functions of the Ohmage back-end.

pmSys supports subjective data through questionnaires, but wishes to support objective data through third party hardware and wearables in the future. For a more detailed analysis of the data through trends and expected values is covered in Cong-Nguyen Nguyen's master thesis [4]. Meanwhile for the theory and decisions on how pmSys can become more scalable and the efficiency of our code and frameworks will be researched and discussed in Kennet Khang Vuong's master thesis [5].

1.4 Research Method

In this thesis, we will be following the design methodology described by the ACM Task Force in Computing as a Discipline [6]. This involves the design, implementation and evaluation of the integration module where the objective data should be registered and visualized.

For all systems, we have created we have been following a Scrum approach [7]. Scrum allows us to do "sprints" which is more or less small deadlines for version releases without too much pre-planning except from selecting tasks to finish since we were working in one team, Scrum was a perfect fit for our situation. We had a lot of thoughts which went on our Project Management Board on Trello [8], but in the beginning, we had no categorization, everything was "urgent". It was after the proof of concept period was over that we added Scrum into our project to be more efficient when releasing the application frequently. The reality was that only 10% of all of our "stories" were urgent, i.e., the some should be prioritized, and some were only an idea.

pmSys itself has been deployed and real football teams are currently using it every single day. The use of a prototype in a real environment allows us to see how the prototype is reacting when several teams is using it at the same time. By doing this, the module could allow us to verify how effective the implementation is and tweak it to become more effective.

1.5 Main Contributions

In this thesis, we show what customizing an already existing general system for collecting health data can provide. As a result of this thesis, several systems and a mobile application under the name pmSys has been released to fulfill the holes and features in which Ohmage (both server and mobile application) is missing. One of the main contributions of this thesis is the implementation of pmSys mobile application, which also has given us media exposure [9] on NRK's (Norsk rikskringkasting AS) web news. Providing the players a better set of tools which is more intuitive and faster to register their self-assessment of health status every day by half the time compared to the Ohmage application (see section 5.6.5). This has proven to help the coaches, adjusting the training sessions to the team's response instead of possibly overtrain the players. The answer rate for one of the teams that has used the first version and the current version of pmSys, has increased by **555%** in the first four months compared to how much they answered the whole year before (see section 5.6.4). 167 people in total have downloaded the mobile application, but the number of active users of pmSys is a bit lower than what Apple Store and Google Play reports, since even though the application has been downloaded it does not mean that the application is being used. 111 users of pmSys are iOS users and 9 users are Android users, which means that the real number of active users of pmSys mobile application is 120/167 (72% of the reported number).

In 2014, the first version of pmSys had four Tippeliga teams as participants. The teams were: Rosenborg Ballklubb (RBK), Strømsgodset Idrettslag (SIF), Viking Fotballklubb (VIF), Sarpsbord 08 Fotballforening (SAR) and Tromsø Idretslag (TIL). During the first deployment, the teams gave us vital feedback about their percieved experience with the system which has helped us in our process of developing a new monitoring system. In 2015, SIF and SAR wished to continue to participate with NiH and us. We also added a local football team into our test group, Lørenskog Idrettsforening G16. They immediately became our primary test group for all functionality; the reason is due to the fact that we were able to contact them directly for feedback instead of going through NiH.

The second contribution of this thesis is the implementation of the pmSys-trainer web portal. The web portal itself gives the coach and the medical staffs full access to all responses of the players, with the possibility of checking the subjective health data and also use objective data from third party systems and wearables.

In this thesis, HUR Labs Jumping board (see section 8.5) and fitbit support (see section 8.4) has been implemented, which makes it possible to get more detailed physical state information about the player without the data being tampered by the player. As a web portal and as a tool it has been successfully used to replace the pen and paper, which were the main idea of creating pmSys.

The pmSys-Push project also shows that creating a push message system that supports both iOS and Android takes little to nearly no time to implement (without the security features), but it also shows that the pmSys-push could become competition of existing services (Table 7.1). For such a simple system, the effectiveness and cost of deploying pmSys-push should not prevent pmSys or any other projects using the same push message system to pay for the same service.

1.6 Outline

Chapter 2, gives a brief presentation of the related background and other systems that has had an impact on the development process and the features of pmSys. We then evaluate Ohmage in chapter 3 more thoroughly, to see which part of Ohmage that can be interesting for pmSys to adapt and copy from. Then, we present our functional and non-functional requirements for the whole pmSys-system in chapter 4. In chapter 5, we present the mobile application with the sole purpose of optimizing the registration of surveys. This chapter covers everything from the development phase to the functionalities the mobile application has to offer. In chapter 6, we present the pmSys-Trainer web portal created spesifically for the coaches. We go through how the web portal works with the mobile application, how they are interconnected with each other and how they solve the problems the teams have been dealing with since the beginning of player health monitoring. We will also present the middleware-system between the pmSys-app and the web portal, pmSys-Push in chapter 7. The system enables a one-way communication channel between the coaches to their players through push notifications. In chapter 8, we will research and find out to what extent integration of objective data can do and what objective data could mean for the analysis for each player in pmSys. Finally, in chapter 9, we conclude the thesis by the results from the previous chapters, and the outcome of this thesis. We also discuss future work for the system and what we could do to make pmSys even better.

Chapter 2

Background and related work

In this chapter, we will be reviewing related work to pmSys. We will be looking into some medical related background that pmSys uses for the systems and frameworks that are used to collect data, which in this case is health data.

2.1 Medical related background / Sport related background

2.1.1 Athlete Physical Status

In the previous chapter, we explained that all medical formulas we are using in pmSys comes from studies conducted to evaluate the physical status of athletes for analysis of workload (Rating of Perceived Extertion), and how well the athlete are feeling when they wake up (Wellness). The physical status of an athlete means everything in terms of how much the athlete should train, prevent injuries and it also makes sure that the athlete can rest enough before matches. Furthermore, we will in explain briefly the surveys conducted to monitor the physical state of an athlete, in this case, a football player.

2.1.1.1 Rating of Perceived Exertion (RPE)

Rating of Perceived Extertion (RPE) [10] are one of the most important surveys for subjective data in pmSys , this survey provides an indication of how the athlete perceives the training based on the intensity of the training session. The data from this survey gives the coach an overview of how the player perceives the training, which translates in better understanding of the player's physical status. But, it also measures the degree of training to a certain extent (given that the data collected are truthful), which may be the cause of injuries as a result of overtraining.

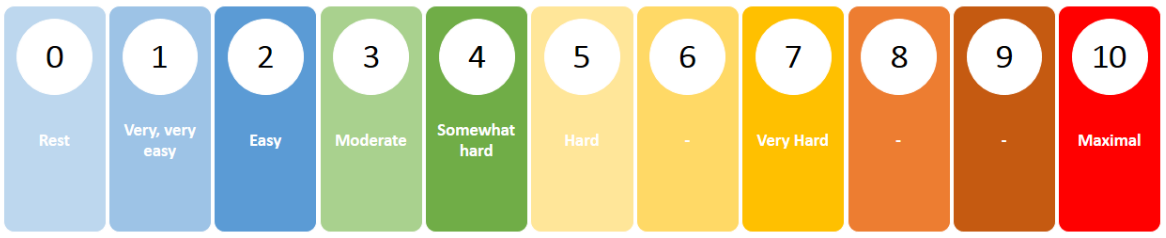


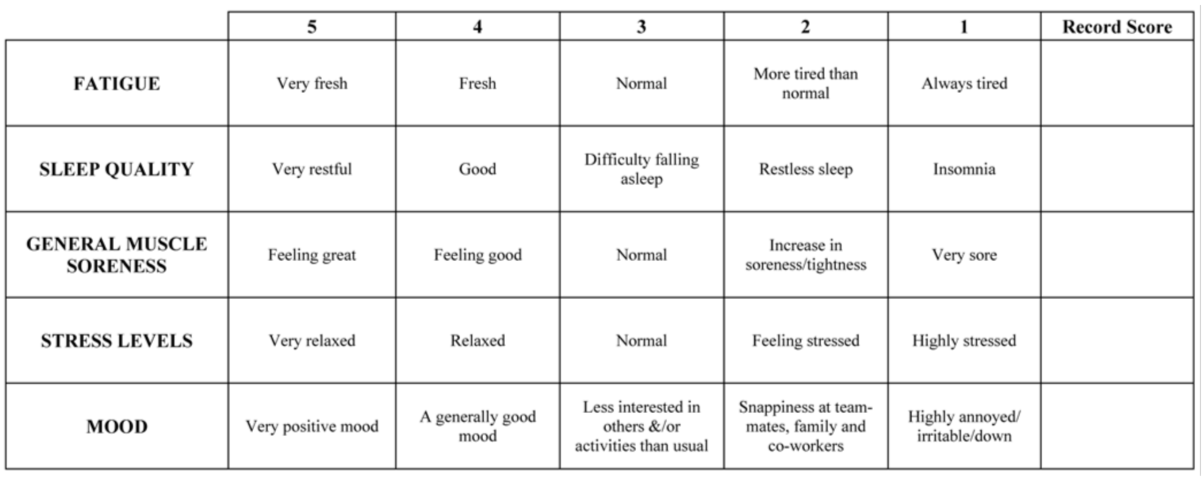
Figure 2.1: Rating of Perceived Extertion (RPE) scale [11]

Right after a training session, either it is a team session or an individual session; the player should answer the RPE survey as soon as possible. For the most optimal response, the survey should be answered within 15 minutes after the session has finished. By using the formula below (see equation 2.1), a load can be calculated and plotted into graphs for a better visual understanding of the physical status of the player. When a survey answer is above a certain amount of load, it is important that the coach can be notified in order to follow up the player's performance, and also various things related to the reason for high RPE load.

$$\text{Training load} = \text{training volume (minutes)} * \text{training intensity (1-10 scale)} \quad (2.1)$$

2.1.1.2 Wellness

The study about the wellness of an athlete gives the medical staff an indication of how the athlete perceives the training based on measuring fatigue, stress, sleep duration, mood and muscle soreness. The survey is a self-assessment of the athlete's physical and mental health after a night sleep, due to this the survey should be answered right after waking up for the most optimal response. The answer scale is a Likert-scale [12] where the answer is translated into a value between one to five (see figure 2.2), and plotted on a graph, where it can show and be used to predict the athlete's health.



	5	4	3	2	1	Record Score
FATIGUE	Very fresh	Fresh	Normal	More tired than normal	Always tired	
SLEEP QUALITY	Very restful	Good	Difficulty falling asleep	Restless sleep	Insomnia	
GENERAL MUSCLE SORENESS	Feeling great	Feeling good	Normal	Increase in soreness/tightness	Very sore	
STRESS LEVELS	Very relaxed	Relaxed	Normal	Feeling stressed	Highly stressed	
MOOD	Very positive mood	A generally good mood	Less interested in others &/or activities than usual	Snappiness at team-mates, family and co-workers	Highly annoyed/ irritable/down	

Figure 2.2: Wellness chart [13]

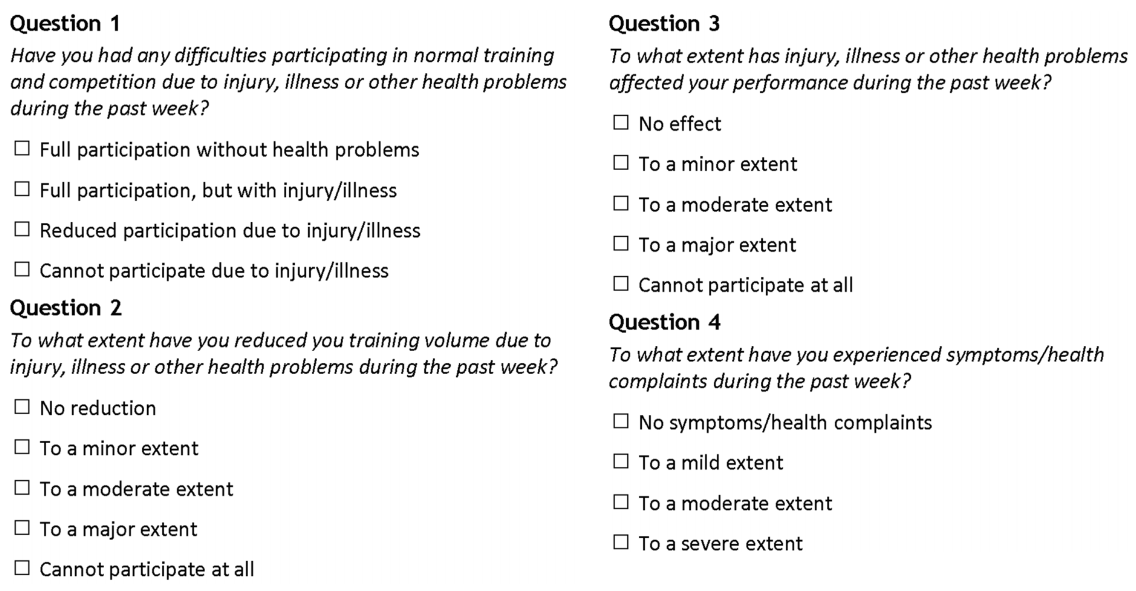
2.1.1.3 Injury

The study about injuries of an athlete gives both the athlete and the medical team an indication on how severe the injury is. By answering a set of questions about what and where the problem lies, a score is generated. The higher the end-score is, the more severe is the injury or illness.

For questions with 4 answers the score are given (based on ascending order): 0-8-17-25 points. And for questions with 5 answers the score given for each choice option the score are given: 0-6-13-19-25 [14].

With a total score of 100, the score can give the coach and the medical staff an indication on how injured the player is even though it is not entirely accurate (due to being subjective data), it does show some simple post-analysis nonetheless.

In pmSys we are using the four questions below (Figure 2.3) as a base to generate the injury score, the



Question 1

Have you had any difficulties participating in normal training and competition due to injury, illness or other health problems during the past week?

☐ Full participation without health problems

☐ Full participation, but with injury/illness

☐ Reduced participation due to injury/illness

☐ Cannot participate due to injury/illness

Question 2

To what extent have you reduced you training volume due to injury, illness or other health problems during the past week?

☐ No reduction

☐ To a minor extent

☐ To a moderate extent

☐ To a major extent

☐ Cannot participate at all

Question 3

To what extent has injury, illness or other health problems affected your performance during the past week?

☐ No effect

☐ To a minor extent

☐ To a moderate extent

☐ To a major extent

☐ Cannot participate at all

Question 4

To what extent have you experienced symptoms/health complaints during the past week?

☐ No symptoms/health complaints

☐ To a mild extent

☐ To a moderate extent

☐ To a severe extent

Figure 2.3: Injury questions [14]

survey has four main questions that are based on the Oslo Sports Trauma Research Center questionnaire for reporting health problems [13]. The reason why there is 11 questions in pmSys RPE survey is because we need to clarify where the injury or illness lies, since this information is vital for the medical staff if they want to conduct more detailed tests on the players (see figure 2.4).

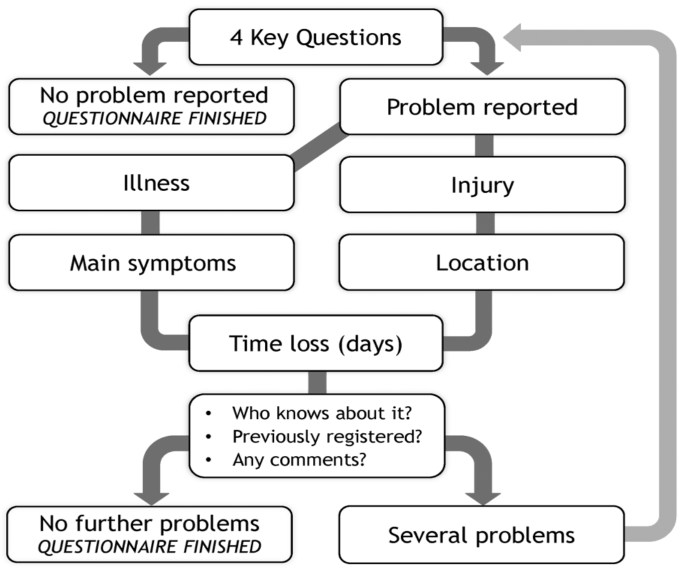


Figure 2.4: Injury flowchart [14]

2.1.2 Analysis of RPE

The data we get from RPE answers can provide a deeper analysis of a players physical state. This can be calculated based on a total **training load** for each day by using the formula in equation 2.1. This value we get is the total load of all RPE registrations, the higher the value, the more physical load has been performed by the player.

When the load has been calculated, we can calulate the **training monotony**. The definition of training monotony is lack of variety when it comes to the training routine, and it is considered a key factor in the syndrome *Overtraining Syndrome* [15]. The higher the value is more similar is the daily training performed by the player, and by reading of this value over time can determine the effectiveness of the training program. It has been suggested that training with low monotony may prevent injuries compared to high monotony training [16]. The idea of calculating this value is to make sure that the player gets sufficient variation between the sessions to prevent injuries. In pmSys, this value is calculated by the standard deviation of each days average TL over the course of seven days (see equation 2.2).

$$\text{Training monotony} = \text{average daily training load} / \text{standard deviation} \tag{2.2}$$

The last equation that the pmSys system calculates is **training strain** (see equation 2.3). This calculation will provide the coaches with a value that is determined based on the total training load for the past seven days multiplied with the monotony value calculated from the previous step (equation 2.2). High strain value can only be achieved by high training load over a long period of time (within the last seven days), and if the value is high then it means that the player has not been given sufficient time to recover from the past training sessions.

$$\text{Training strain} = \text{total weekly training load} * \text{training monotony} \tag{2.3}$$

2.2 Participatory Sensing

The concept of what pmSys is doing and the related systems below are called **Participatory Sensing** [17]. Participatory Sensing is a distributed data collection and analysis approach that revolves around the individual that uses the system. According to a survey conducted by TNS Gallup [18], in Q3 of 2014, 81% of the Norwegian population had a smartphone. The definition of a smartphone when this survey was conducted were as following: *A smartphone has to be able to connect to both the internet and accept emails. It also has to have a touch-screen or be able to install programs*. By using smartphones to analyze, the individual can gather information about certain aspects of their normal daily life (for example step counting) in an easy way without keeping track of the data themselves. In other words, smartphones can be used to send in self-reports (surveys in pmSys) and to be used as monitors and recording devices for sensors such as wearables connected to a mobile device (see chapter 8).

In order to achieve the goals of a participatory sensing system, there is a model which defines the general user roles needed to fulfill this approach:

- *Initiators* - Users being able to create campaigns and specify the data collection
- *Gatherers* - Participants (users) of the system
- *Evaluators* - User that verifies and classify collected data in the campaign
- *Analysts* - User that process, interpret, presenting data and conclusions

2.3 Open mHealth

Open mHealth [19] is a registered non-profit organization which revolves around building an architecture that has shared data standards. The way they have created a set of standardlized frameworks with optimalized data schemes for clinical usage. Instead of only collecting data from closed systems, the frameworks enable the possibility of retrieving all types of data from third party systems, such as wearables.

Mobile Health (mHealth) [20] is a practice of medicine and public health supported by mobile devices. mHealth encourages reuse of code to create an open platform for collecting health data, but also following a standard for data capture. The objective of mHealth is to make it easier to collect data and exchange the data between different systems and platforms, but also so that the patient can collect and share their own personal health data as they wish.

Regarding the increasing rate of smartphones per person in the world [21], gathering health data is made possible for better treatment. This would perhaps make it easier to change doctors and retrieve correct treatment when a patient is in need. However, in order to do this frequently, systems has to change their approach of developing health data systems from a silo-approach (the same as "Stovepipe" in Figure 2.5) [22] (the lack of collaboration and standardization between other systems) to how the mHealth data schemes work or else the work of mHealth will be held back by limitations from lack of collaboration. mHealth also makes it possible for patients to share data to anyone they want to share with, even without visiting clinics or hospitals. New upcoming mHealth systems is Apple's new HealthKit which supports and makes it possible to connect multiple systems into the application for data analysis.

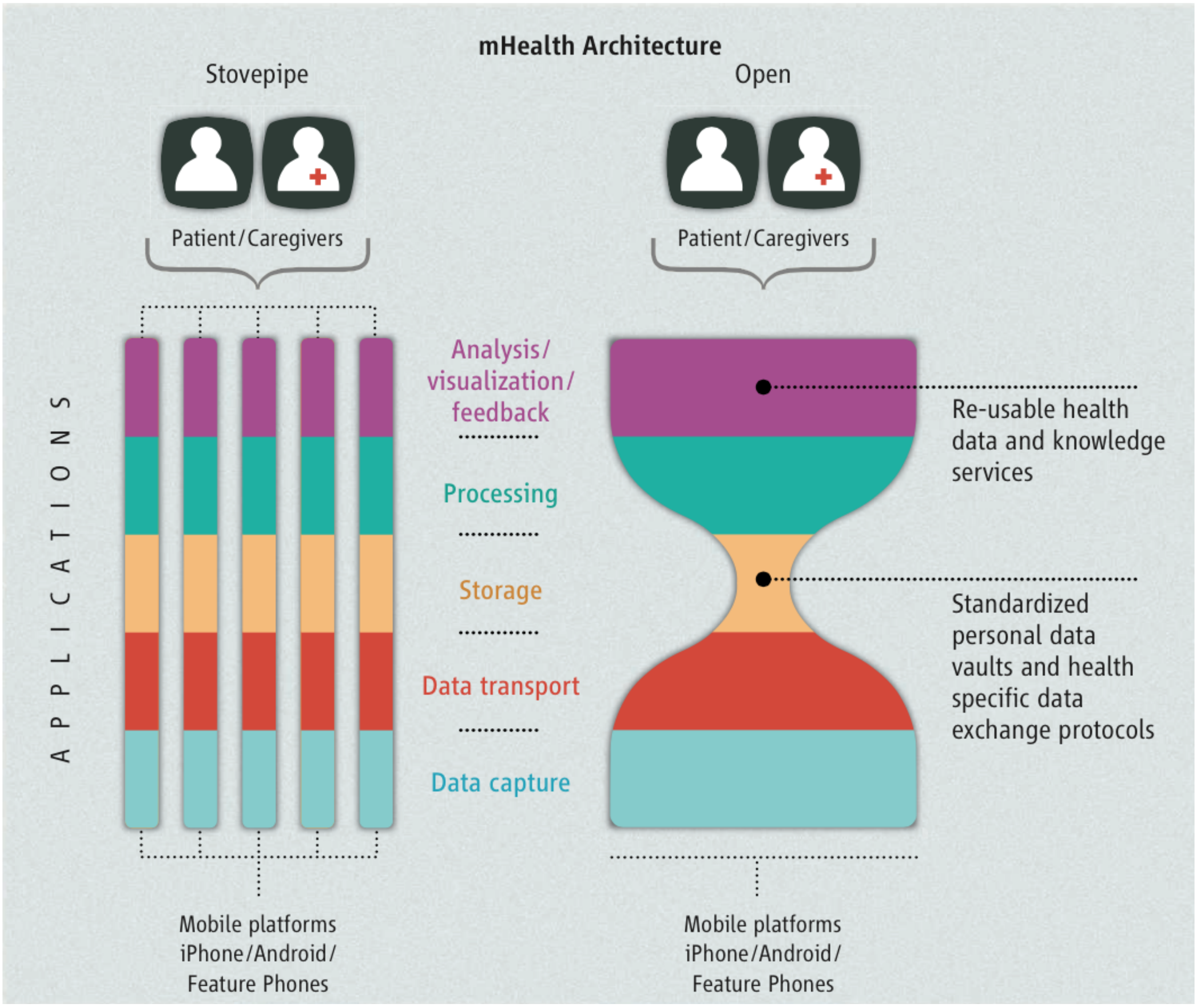

mHealth Architecture
Stovepipe
Open
Patient/Caregivers
Patient/Caregivers
APPLICATIONS
Analysis/ visualization/ feedback
Processing
Storage
Data transport
Data capture
Re-usable health data and knowledge services
Standardized personal data vaults and health specific data exchange protocols
Mobile platforms iPhone/Android/ Feature Phones
Mobile platforms iPhone/Android/ Feature Phones

Figure 2.5: Stovepipe vs mHealth [20]

2.4 Ohmage

Ohmage [3] is an open-source project initiated by Deborah Estrin at Cornell University where it has become a platform for collecting health data. Ohmage is a system created to collect data from users by either manual registering through the mobile application or by letting the application collects data automatically (continous data streams). All captured data is timestamped, geocoded and uploaded to the Ohmage server for analysis. Ohmage is a product of many participatory sensing systems combined to provide a generic platform with the possible to customize for different scenarios. Before Ohmage, each focus group created their own system specifically for gathering information by using smartphones. What all the systems have in common is that they all share the same ideas on how to gather health information, how the data is stored and that it has to be made simple to manage the data on the system. This approach discouraged code reuse, which caused high development and maintenance when using multiple systems at a time [23]. The Ohmage platform offers a big range of applications and features (Figure 2.6), with a rich featured Application Programming Interface (API). With the Ohmage back-end in the middle of everything, participatory sensing systems and applications can be built independently and customized for each purpose revolving around the back-end which meant the data would still be standardized.

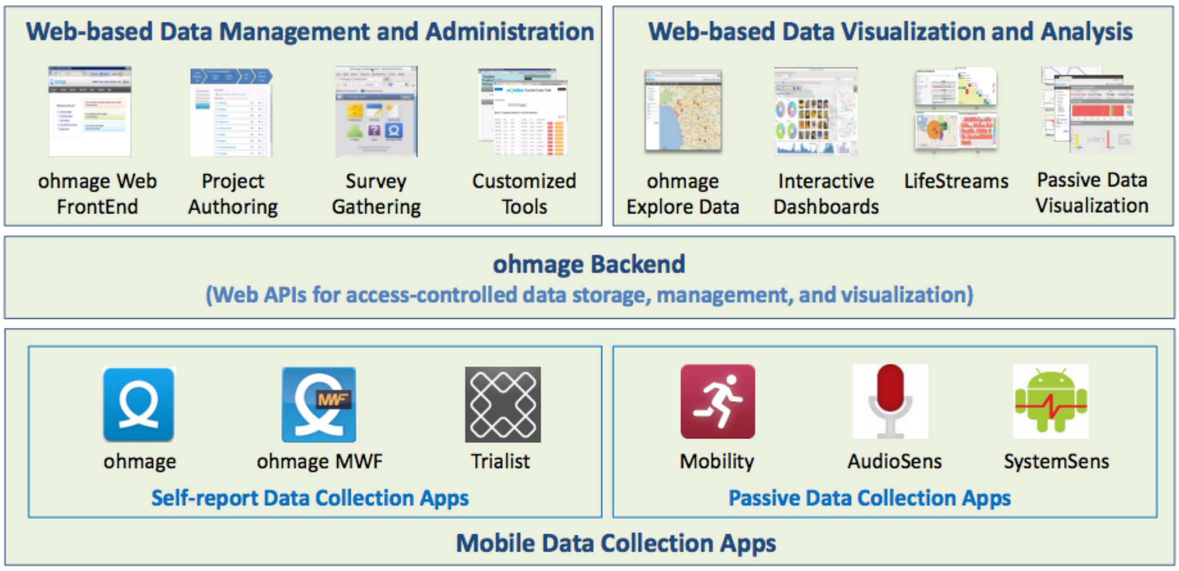

Web-based Data Management and Administration
ohmage Web FrontEnd
Project Authoring
Survey Gathering
Customized Tools
Web-based Data Visualization and Analysis
ohmage Explore Data
Interactive Dashboards
LifeStreams
Passive Data Visualization
ohmage Backend
(Web APIs for access-controlled data storage, management, and visualization)
ohmage
ohmage MWF
Trialist
Self-report Data Collection Apps
Mobility
AudioSens
SystemSens
Passive Data Collection Apps
Mobile Data Collection Apps

Figure 2.6: Applications and features of Ohmage [23]

2.5 District Health Information Software

District Health Information Software (DHIS) [24], version 2 is an open-source project and tool for collecting data and validation of data initiated and created at University of Oslo and is widely used in Africa and Asia. The system collects health data in order to group and discover outbreaks of lethal viruses such as Malaria and Ebola. DHIS2 was created as a Health Information System (HIS), but in the later years incorporated the participatory sensing approach for data collection, in order to follow up with the new technologies to add additional ways to register data. As a participatory sensing system, DHIS2 aims for reporting from mobile devices and with low bandwidth usage mainly for countries with bad cellular coverage. For mobile clients the DHIS2 dashboard has a mobile interface for easier navigation and one of the focus areas is to not use too much bandwidth for development countries due to the high cost of data traffic. In addition DHIS2 also supports text message notifications.

One of the goals of DHIS2 is to create a digital framework that supports all the stages of the information cycle (see figure 2.7), which includes functions such as; collecting data, running quality checks, data access at multiple levels, reporting, analysis, enable comparison of the data across time and space and see trends. A web application has to be uploaded as a DHIS2 module to be able to use the API that DHIS2 provides because cross-domain requests is disabled by default. And one way to bypass the cross-domain lock is by using *JavaScript Object Notation with Padding* (JSONP) [25], because web browsers does not enforce the same restriction on <*script*> tags in JavaScript.

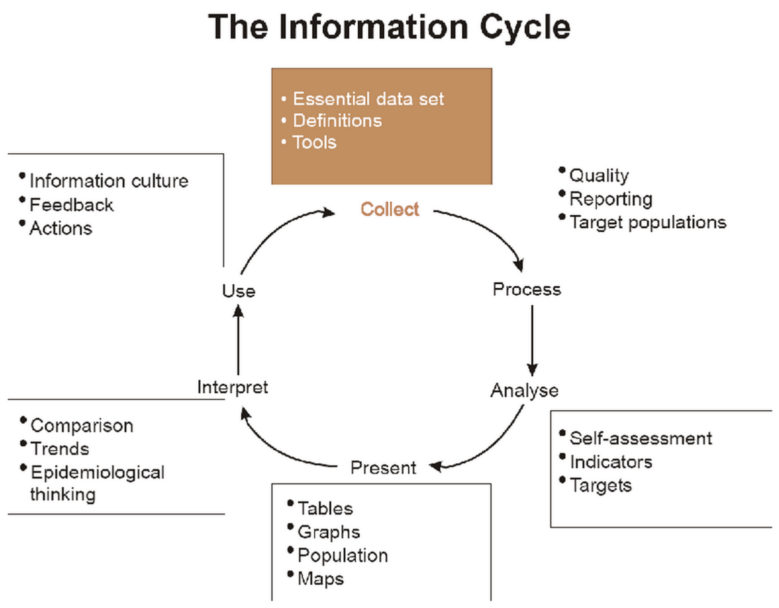


Figure 2.7: The health information cycle [26]

2.6 MilanLab

MilanLab [27] is one of the most known laboratory that does analysis based on a player's health. The lab has been operating since March 2002 with Jean-Pierre Meersseman in charge. Meersseman was a Belgium chiropractor with the idea that it was possible to determine a player's health, by collecting data from all sorts of areas (like teeth and feet). His goal was to provide the best possible analysis and management of individual players in AC Milan and the Italian National team. The whole idea of the analysis is to prevent injuries of players, which increases the average age of a professional football player in AC Milan. The lab itself had the responsibility of assessing players in all types of areas (neurology, biochemistry, psychology etc.) in all stages of a player's stay in the club, whether it is in the pre-signing stage or during a season.

Their equipment is state-of-the-art and by using a jumping board connected with electromyography [28] attached to the leg muscles, they could collect nearly 60.000 data points from one single jump which would provide them data on the player's flexibility and speed. It was also mentioned that the prediction after one of these jumps had a 70% accuracy [29], which is insanely high when you think of the chances of flipping a coin is 50 per cent. Even though Meersseman left MilanLab in 2011 to open his own clinic, MilanLab still exists today.

2.7 Summary

To create a participatory sensing system customized for a football team, there is a selected areas that has to be improved in order to make it useful. From a player perspective, the application used to register data has to be as fast and effective as possible. The amounts of clicks per screen has to be kept to a minimal in order to save time, but it also has to be intuitive enough for anyone to use without setting up a workshop. Ohmage has created a mobile application to register data, both subjective and objective data

(Figure 2.6) through the applications Ohmage and Mobility. The problem with these applications is that it is too generic for our case, but it also lacks features on the platform they support. DHIS2 has yet to release a mobile application for their system, but as a workaround, DHIS2 has created their web portal so that it is responsive for mobile devices. This way any user of DHIS2 can use the web portal, no matter what type of phone the user has (old / new).

From the coaches' perspective, it is important for them to be able to use the tools to analyze and monitor the physical state of their players. Ohmage does support this feature, but it is timeconsuming and complex operation to get the correct data. DHIS2 has solved this by letting the users develop and maintain their own applications inside of DHIS2, also known as *Web application* [30]. This way the user of the system can create anything by using DHIS2 as their back-end, and manipulating the API provided to fetch and create content as they please.

Another important aspect for a health monitoring system is that they must be able to support third party hardware and wearables. The way DHIS2 is created, the only way to support this is by creating a web application, but the complexity of the applications are limited to what HTML, JavaScript and CSS can do. Ohmage do support this feature, however, adding a new third party system into Ohmage requires a lot of time and work due to the complexity of how Ohmage has been implemented.

System	Responsive web portal	Visualization	Mobile application	Available API	Support for third party
Ohmage	No	Yes	Yes	Yes	Yes
DHIS2	Yes	Yes	No	Yes	No

Table 2.1: Ohmage vs DHIS2

For a system such as pmSys, it has to include all the features mentioned above (general functionality is rhetorical). In addition, be easy to add support for new wearables since we are rapidly moving towards the era of *Internet of Things* [31] (IoT, see section 8.3.2). Based on the discussion above, Ohmage is the most suitable for what we want pmSys to offer regarding functionality. We also want to use the standardized frameworks for data schemes that Open mHealth has created (which are also partly implemented in Ohmage), so that pmSys can support all sorts of data from other products and system without problems in the future. Therefore, in the next chapter, we will discuss what features Ohmage can provide pmSys. We will also present briefly the most vital parts of Ohmage that pmSys currently uses and supports.

Chapter 3

Ohmage

In this chapter, we will discuss the Ohmage server that we want to use as pmSys back-end, since this server could be the core of our system it is important to get a overview on how the system works. We will start by explaining why we did not choose to go for creating our own back-end and also why we used Ohmage as our back-end instead. Next, we will get an overview of the features the Ohmage server can provide and how these features are applied in pmSys.

3.1 Early ideas

In the early stages of our system, we had an idea on how we wanted to build this system. After taking a course at University of Oslo about Open Source Development, we were eager on trying out a new up and coming framework written in JavaScript; NodeJS. We had read articles and reviews about NodeJS as a whole whether it was ready for the enterprise world [32] [33]. After that we wanted to create our back-end in JavaScript and using NodeJS as our platform, due to the reason that we wanted to test if these frameworks was a better fit than the more standard back-end solutions (Java, Tomcat and MySQL).

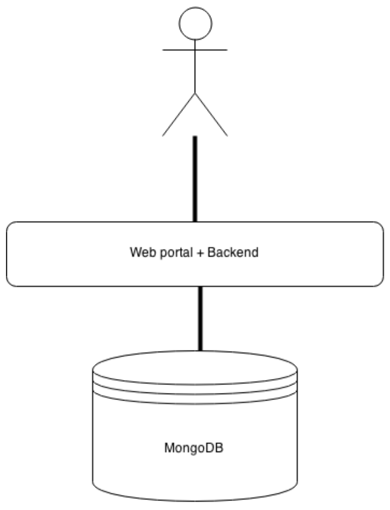


Figure 3.1: (Early) pmSys architecture

The important thing that were different from the early idea for pmSys and how Ohmage is currently working, is that pmSys were supposed to be one server for both front-end and back-end. Meanwhile Ohmage's front-end layer (Figure 3.3) were optional if we chose to use Ohmage, but as a standard the front-end were installed no matter what if you installed the Ohmage server.

```
// SQL-query
SELECT *
FROM users
WHERE status = "Approved"
ORDER BY user_id DESC

// MongoDB No-SQL query
db.users.find({ status: "Approved" }).sort({ user_id: -1 })
```

Figure 3.2: A simple MongoDB query (NoSQL)

Our vision was to use the new "up and coming" open-source frameworks to create our back-end, NodeJS [34] as our server back-end and MongoDB [35] as our data storage. NodeJS is a platform built on Chrome's JavaScript V8 library written in C and C++. The framework is asynchronous event driven and one of the main goals are non-blocking I/O compared to other types of servers such as Java. Meanwhile, MongoDB is one of many NoSQL databases where data entries in the database is no longer added in tables and rows, but rather added as a JavaScript Object Notation (JSON) objects in a collection of data objects. Instead of writing queries in SQL to search for data, in NoSQL you query for a document (see figure 3.2).

After a couple of days had passed, we found out a series of critical points we did not account for when we discussed the architecture. In other words, we simply did not think of all the features we would be needing to implement from scratch to create pmSys. Here are some of the features:

- User registration / creation
- Team allocation
- Authentication
- Security (of data and user credentials)
- Surveys
- Reports
- Data storage

Within the timeframe we had, creating the system with NodeJS and MongoDB was possible, but not to the extent that the system itself would be stable enough to be set into production by summer 2014 (planned release date for The Norwegian National team). It were mentioned that Tromsø IL was currently using Ohmage for quire some time to collect health data. After hearing this, we did some research on what Ohmage could provide and how they solved most of our criteria's (Table 2.1), we began to use Ohmage to understand the system better. After that we got the idea to replace the Ohmage dashboard into something more user friendly for coaches, and to recreate the Ohmage mobile application for both Android and iOS with better functionality for the users (see section 3.6). The Ohmage system itself is too generic and can be used to collect different type of data, and that is why pmSys will provide an alternative front-end where the user can use more customized features specifically for football monitoring. As a result, our original plan had to be scrapped, however, we have kept NodeJS as a part of our master thesis when we've developed other sub-systems of pmSys that we will explain more about later in this thesis.

3.2 System design

System requirements to run the Ohmage server and can be hosted on all platforms as long as the following minimum requirements is met:

- Java 7 or later.
- MariaDB/MySQL 5.5 or later.
- Tomcat 7.0.28 or later.

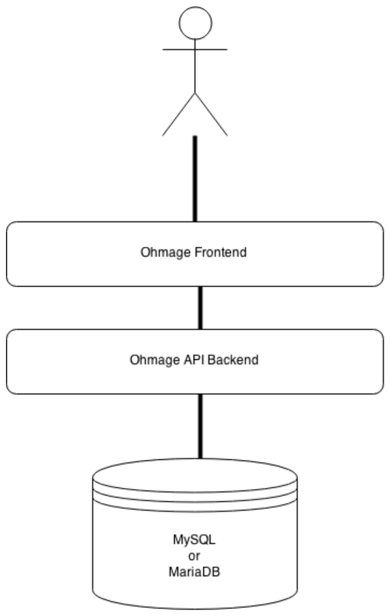


Figure 3.3: Overview of Ohmage architecture

3.3 Ohmage back-end API

The Ohmage back-end is written in Java and provides a RESTful API where HTTP calls can be done towards it in order to do CRUD [36] operations. The back-end has several endpoints where an authenticated user can access the data in campaigns, this is one of the most crucial parts that the mobile application needs to do all it's operations successfully. Even though the back-end has many endpoints to retrieve data there is still one thing Ohmage lacks, which is customization. The queries can not be customized enough, some HTTP calls are locked and returns a lot of unnecessary data which takes a lot of bandwidth and creates extra overhead in the long run.

Operation	Description
User Authentication	CRUD operations for authenticating
User Manipulation	CRUD operations for user information
Campaign Manipulation	CRUD operations for campaigns
Survey Manipulation	CRUD operations for surveys
Class Manipulation	CRUD operations for classes

Table 3.1: Most used operations used by pmSys

In pmSys, we have two systems that use the Ohmage API, pmSys-app and pmSys-Trainer. Both the mobile application and the web portal we have created uses the back-end to authenticate, collect, validate and visualize data. The most frequently used operations in pmSys is illustrated in table 3.1. All HTTP calls to the back-end has to be sent as a JavaScript Object Notation (JSON) [37] which is an open standard format that uses a human-readable text to transmit data as objects with a key-value pair. All responses are also returned as JSON objects, which makes it easier to manage the data and the formats when using the API. By using the API, the developers can get direct access to Ohmage data model when requesting a resource by providing an endpoint.

3.3.1 Authentication

All HTTP calls in Ohmage has to be authenticated, and it exists two types of authentication method in Ohmage. The first type is *Stateless* authentication (see figure 3.4), which is a hash returned by the API when the user has authenticated with the correct username and password. The hash returned is encrypted, however, if decrypted, it reveals the username and password. The only way to change the returned hash is to change the password, since the salt and hashed password will remain the same as long as the content is the same. This type of authentication is created for the purpose of allowing mobile applications to provide authentication once, and then have access to the API at all time.

```
{
  "result" : "success",
  "hashed_password": "42f812nmk3p1iuy7"
}
```

Figure 3.4: Stateless authentication - Hash based

The second type of authentication is *Stateful* (see figure 3.5). This authentication method is created for usage in a limited timeframe, which cannot be set by server configuration (even though the API documentation states otherwise). This authentication method does not suit the way pmSys-app is fetching data from the Ohmage back-end, but they are still needed for some CRUD calls where the Stateless authentication do not function (see table 5.2).

```
{
  "result" : "success",
  "token": "2i9sh34hjklu"
}
```

Figure 3.5: Stateful authentication - Token based

3.3.2 User roles

Ohmage has 3 user roles that are slightly different from each other (access-wise). When it comes to having several roles in Ohmage one user can be both "Priviliged" in one campaign, but also be "Restricted" in another campaign, which gives the user account the opportunity to become a part of several campaigns at the same time with different roles.

When the Ohmage system is deployed, an administrator account is created to manage the whole system. For all newly created accounts after the administrator account, they must follow these requirements:

- Username - The length of the username has to be between 3 and 25 characters. The system can only accept usernames that contain A-Z, a-z or digits from 1-9.
- Password - The length of the password has to be between 8 and 16 characters. The password also requires containing at least one lower case character, and at least one digit. The Ohmage wiki [38] also states, that the password also has to contain one special symbol. But in pmSys, this default system requirement has yet to show itself.

3.3.2.1 Administrator

The role as "System administrator" is the same on all types of systems; the user has access to do everything on the web portal.

3.3.2.2 Privileged

Ohmage has a role that they call "Privileged" which have some extra privileges when they are a part of a campaign. They can read all collected data for all users in a campaign, which means if a user is set as "Privileged" they must be trusted by the administrator. In pmSys this user role is allocated for coaches and staff of a team (medical / physical).

3.3.2.3 Restricted

"Restricted" users are more or less the equivalent of "participant" in this context, users with the role as "Restricted" can only answer and see their own data in a campaign they are assigned to. In pmSys this user role is allocated for all football players.

3.4 Class roles

A class is the same as a group created to contain users for access control. When creating a class, some properties must meet in order to be able to create the class:

- A Uniform Resource Name (URN) - This URN has to be unique for the class
- Name - This name is used as an alias for the class
- Description (optional) - This property can be skipped if wanted, but it is just used to describe the class (example: UiO players)

As stated in section 3.3.2, there is three user roles a user can have. But the "Administrator" role gives a user the same privileges as "Privileged", and then some more access as a system administrator. For a more detailed list of what users with different roles shown in table 3.2.

Operation	**Privileged**	**Restricted**
Read the class properties	Anytime	Anytime
Read list of logins for the class	Anytime	Anytime
Read detailed information of the class	Anytime	Never
Read list of campaigns that is associated with the class	Anytime	Anytime
Add and remove users from the class	Anytime	Never
Update class information and change user roles	Anytime	Never

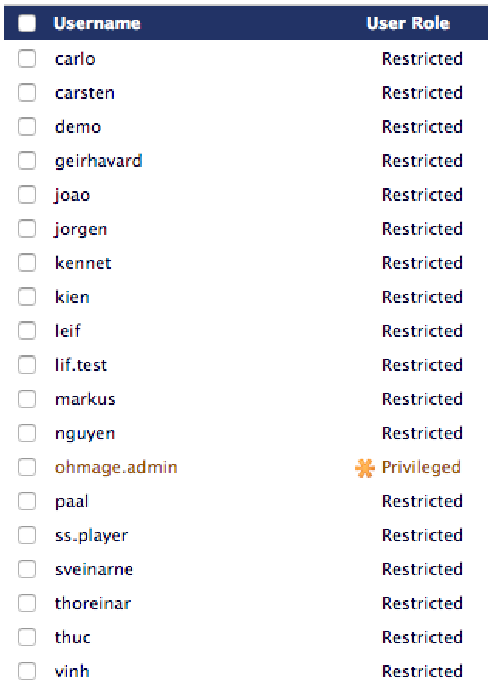
Table 3.2: Class restrictions

3.5 Campaign and surveys

A campaign is a container for surveys [38], which means that it can contain several surveys (1:M-relation). The campaigns are defined by Extensible Markup Language (XML) which allows the creator to create a survey with serveral types of prompts (Figure 3.7):

- Audio
- Video
- (Custom) Multiple choice
- Number
- Image
- Remote activity
- (Custom) Single choice
- Text
- Timestamp

The idea of using campaigns in pmSys is that every team have their own campaign, this way the users can be added into classes (which are more or less a list of names) to give them either access as "Restricted" or "Privileged" (Figure 3.6).



Username	User Role
carlo	Restricted
carsten	Restricted
demo	Restricted
geirhavard	Restricted
joao	Restricted
jorgen	Restricted
kennet	Restricted
kien	Restricted
leif	Restricted
lif.test	Restricted
markus	Restricted
nguyen	Restricted
ohmage.admin	Privileged
paal	Restricted
ss.player	Restricted
sveinarne	Restricted
thoreinar	Restricted
thuc	Restricted
vinh	Restricted

Figure 3.6: An example of how users are listed in Ohmage

When a class has been added to a campaign, their access roles within the campaigns and surveys are based on their class role. Privileged users receives a new role **supervisors**, and restricted users become **analysts**. The new roles (in the campaign) defines what information will be visible to the user, and also what the user can do in the campaign. In table 3.3, a more detailed list of what actions the users can do within the campaigns they belong to.

3.6 Ohmage MWF mobile application

The Ohmage MWF mobile application is released for both iOS and Android, even though their names are the same their features are somewhat different from each other. MWF stands for **Mobile Web Framework** and it is a framework developed by UCLA [39]. The framework is following the principle *develop once, use everywhere* [40], by using this framework the developers does not have to create a spesific implementation for each device, but rather one implementation that works on multiple devices

```xml
<?xml version="1.0" encoding="UTF-8"?>
<campaign>
<campaignUrn>urn:campaign:demo:eng:srpecoach</campaignUrn>
<campaignName>Surveys for coaches-demo-eng</campaignName>
<surveys>
<survey>
<id>plansrpe</id>
<title>Planned sRPE</title>
<description>Plan sRPE</description>
<submitText>Survey is done. Thank you!</submitText>
<showSummary>true</showSummary>
<editSummary>false</editSummary>
<summaryText>Results</summaryText>
<anytime>true</anytime>
<contentList>
<prompt>
<id>time</id>
<displayLabel>Date and time</displayLabel>
<displayType>event</displayType>
<promptText>Date and start time for the session</promptText>
<promptType>timestamp</promptType>
<skippable>false</skippable>
</prompt>
</contentList>
</survey>
</surveys>
</campaign>
```

Figure 3.7: Example of a campaign (in XML) for RPE (coaches)

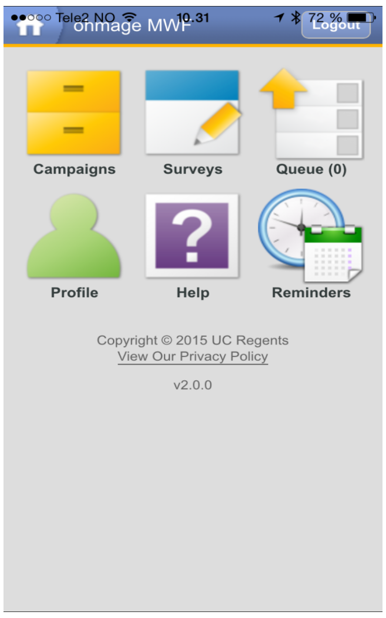
Operation	Supervisor	Analyst
Read campaign properties	Anytime	Anytime
Read which classes participates in the campaign	Anytime	Anytime
Read user roles of other users in the campaign	Anytime	Only the author is shown
Update campaign state (running/stopped)	Anytime	Never
Update privacy state (public/private)	Anytime	Never
Add/remove class, supervisor, analyst to the campaign	Anytime	Never
Delete campaign	Anytime	Never
Read own private or shares survey responses	Anytime	Anytime
Read someone else's shared survey responses	Anytime	Only if the campaign is shared
Read someone else's private survey responses	Anytime	Never

Table 3.3: Campaign restrictions [38]

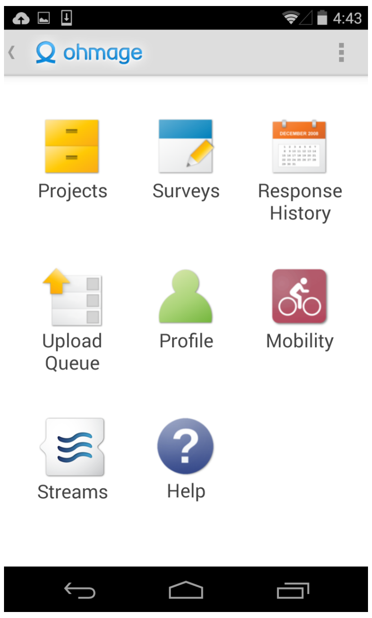
[41]. As a supporting tool for reporting health data, Ohmage MWF lacks some vital features that many of the first users of Ohmage complained about. We received a small list of feedback from the users on what they found were complicated when using the Ohmage MWF application:

1. It takes too long to register a survey.
2. I cannot see any surveys, how can I answer a survey?
3. Where can I find my own progress in the survey?
4. I would like to answer a survey, but I cannot remember it. A reminder function would be nice.

Android has two different versions of Ohmage, one of it is Ohmage MWF where it can only be used on mHealth's Ohmage back-end (see figure 3.11(a)). The other version is the "normal" version where you can choose which server you want to connect to (see figure 3.11(b)). Since we do not have credentials to access the MWF version, we will focus on the "normal" Ohmage version, and hopefully they have the same features. The first feature is the "response history" where the user can see when, what and how many surveys were sent each day. The application also has support for *Mobility* which supports collecting third party systems used to register objective data such as step counter, and GPS-tracking. These two features is missing on the iOS version, which makes it harder to support collecting objective data for users on both platforms. Meanwhile the iOS version is missing the features mentioned previously, but the iOS version has the "reminder" functionality (locally only), which the users wished for to be able to remember to answer surveys. In the sections below, we will go through all the main features of the Ohmage MWF application for both operative systems. The dashboard for both platforms is quite different from each other, not only the features but also the naming of the features on the dashboard (see figure 3.8).



(a) Ohmage (iOS)



(b) Ohmage MWF (Android)

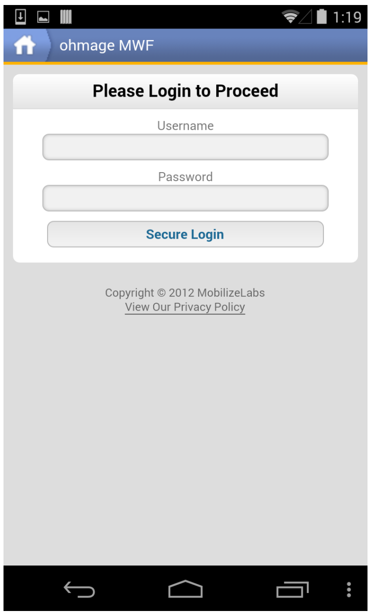
Figure 3.8: The difference between iOS and Android's dashboard

3.6.1 Login page

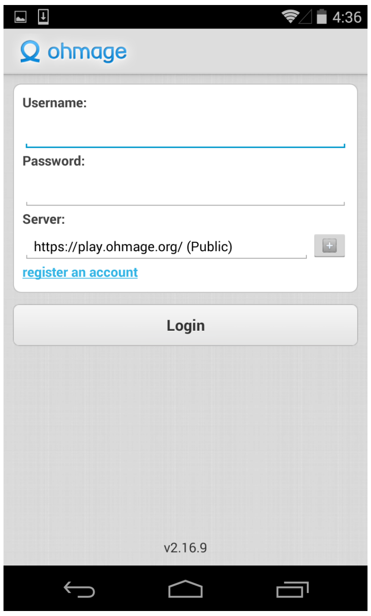
The very first page the user is presented is the login page. Every time the application has been updated, all data from the application is deleted. This would force the user to login again, if the username and password is difficult or complicated, or if it has been a while since the last time, then it might be difficult to log into the application again. Both the server information and user credentials have to be inserted in order to get access to the surveys.

Another big technical problem the application has is that it only support one format of the URL path for the back-end. The application did not support direct IP addresses, but it had to be in the format of: **http(s): serverURL.com**. It was not possible to change the path of the API since the URL to the back-end has been hardcoded into the application. It will always try to do requests against **serverURL.com/app**, and if the API is not installed under **/app**, as a result, it will never be able to authenticate with the back-end or do any requests. The login page will be reloading, and the system would be in a state where it is not possible to do anything. The only way to reset it is to close the application, and during the authentication no feedback about the system status is returned to the user.

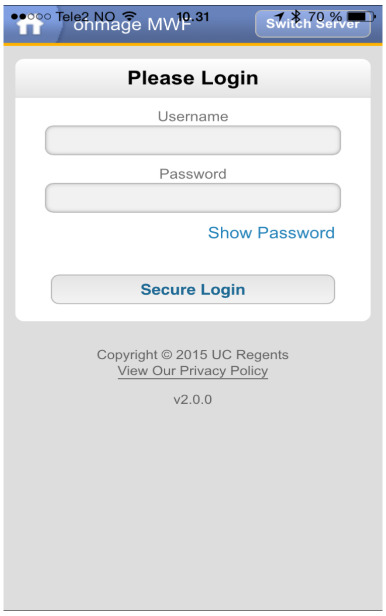
If the entered URL path is incomplete by missing a period symbol, then the user will be noticed by the application that the URL is wrong. If you change the URL after getting this error message (see figure 3.10(a)), you will be met with another error message even though it is the correct path (see figure 3.10(b)), and the fastest way to reset the state of the application is to simply change the server path to one of the default servers instead of the custom one. Furthermore, when you have entered the server URL, you will be greeted with the message that everything is alright (see figure 3.10(c)). It is important to note that there is no check if the server is actually valid, as long as the URL contains one period symbol (**.**) it is counted as valid (ex. ohmage.nktconsulting is valid!).


1:19
ohmage MWF
Please Login to Proceed
Username
Password
Secure Login
Copyright © 2012 MobilizeLabs
View Our Privacy Policy

(a) Ohmage MWF Login on Android

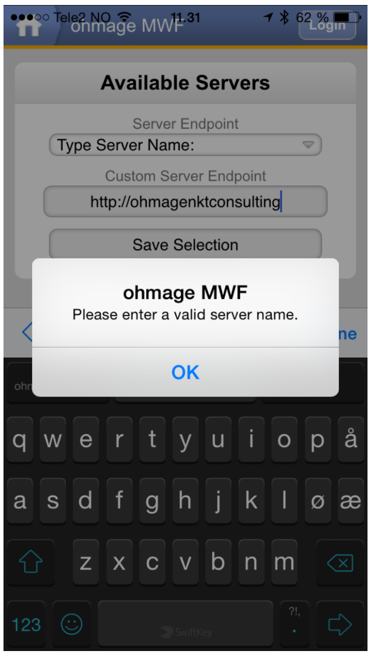

4:36
ohmage
Username:
Password:
Server:
https://play.ohmage.org/ (Public)
register an account
Login
v2.16.9

(b) Ohmage Login on Android

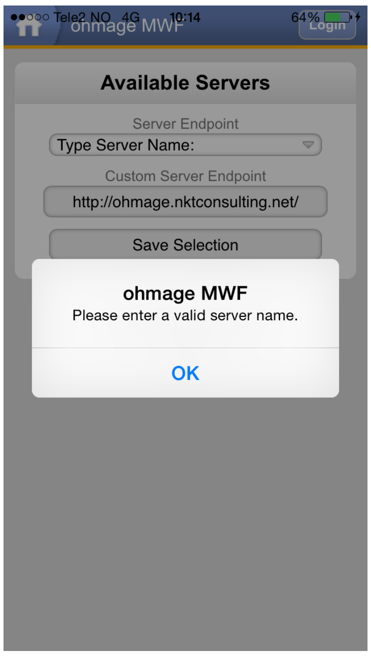

●●○○○ Tele2 NO 10.31 70 %
ohmage MWF
Switch Server
Please Login
Username
Password
Show Password
Secure Login
Copyright © 2015 UC Regents
View Our Privacy Policy
v2.0.0

(c) Ohmage MWF Login on iOS

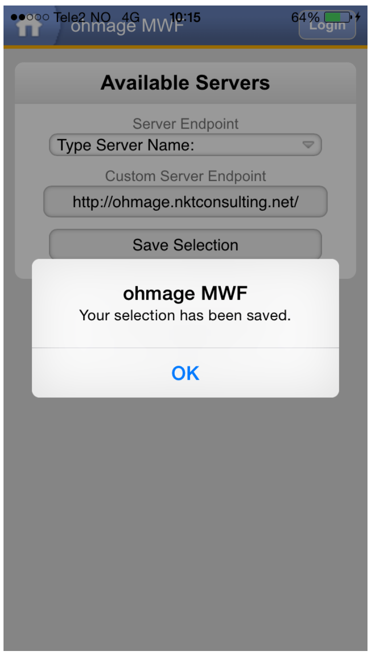
Figure 3.9: Overview of Ohmage Login and Server page


●●●○○ Tele2 NO 11.31 62 %
ohmage MWF
Login
Available Servers
Server Endpoint
Type Server Name:
Custom Server Endpoint
http://ohmagenktconsulting
Save Selection
ohmage MWF
Please enter a valid server name.
OK

(a) Wrong entered URL


●●○○○ Tele2 NO 4G 10:14 64%
ohmage MWF
Login
Available Servers
Server Endpoint
Type Server Name:
Custom Server Endpoint
http://ohmage.nktconsulting.net/
Save Selection
ohmage MWF
Please enter a valid server name.
OK

(b) Application bug

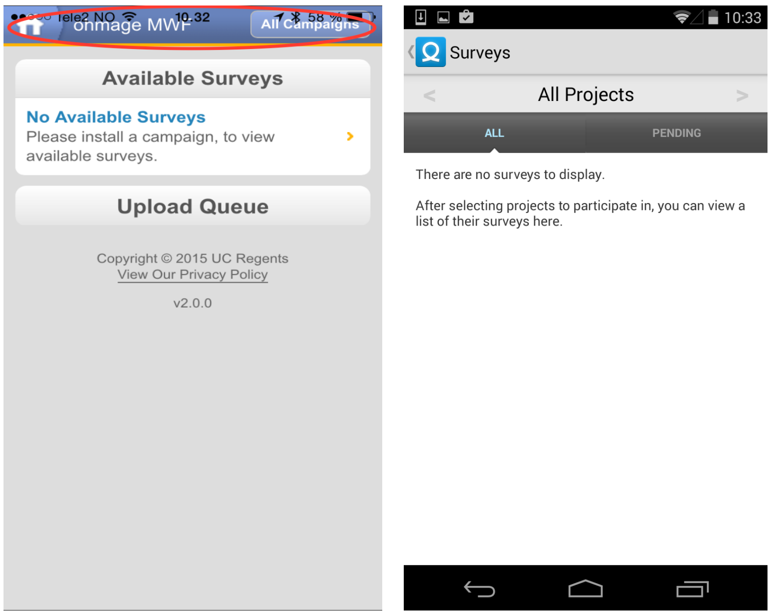

●●○○○ Tele2 NO 4G 10:15 64%
ohmage MWF
Login
Available Servers
Server Endpoint
Type Server Name:
Custom Server Endpoint
http://ohmage.nktconsulting.net/
Save Selection
ohmage MWF
Your selection has been saved.
OK

(c) Correct URL

Figure 3.10: Problems when entering Ohmage URL

3.6.2 User Interface

Once the user has successfully logged into the application, they will be redirected to the dashboard which will present all the features the application can provide (see figure 3.11). The most important feature for collecting data is the surveys. When the user click on the **Survey** button, they will see that there is no available survey (see figure 3.11(b)). In order to be able to do the surveys, the user has to download the campaign and add the surveys to their survey list before they can start answering. This forces the players to use extra time to set up before they can start register data, and may cause confusion the first time the player is doing this. Another problem with the UI, the top bar's margin is pushed lower than what the default iOS status bar is, which results in buttons being nearly impossible to click and use (see figure 3.11(a)).



Available Surveys

No Available Surveys
Please install a campaign, to view available surveys.

Upload Queue

Copyright © 2015 UC Regents
View Our Privacy Policy

v2.0.0

There are no surveys to display.

After selecting projects to participate in, you can view a list of their surveys here.

(a) Ohmage MWF surveylist on iOS (b) Ohmage surveylist on Android

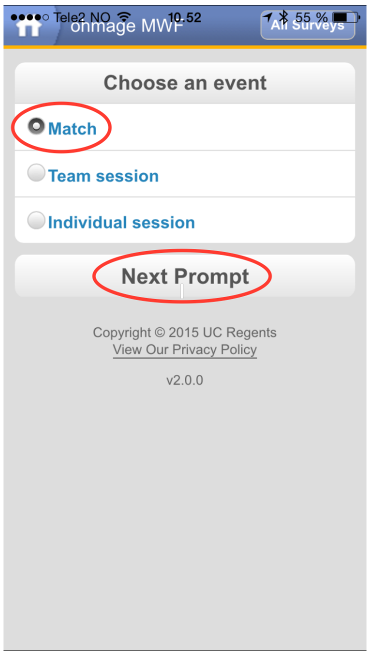
Figure 3.11: The difference between iOS and Android Ohmage MWF

3.6.3 Reporting workflow

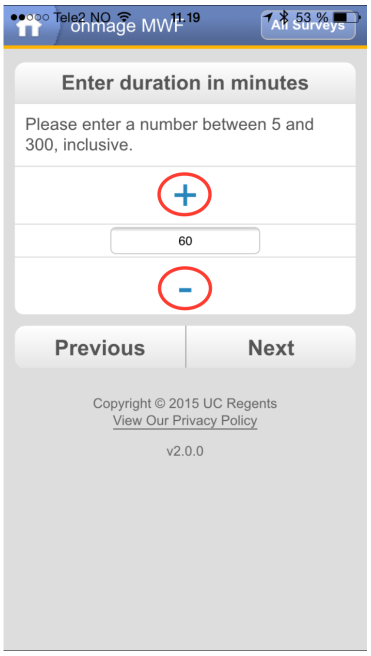
The current reporting workflow is not optimized for time efficiency, because of this the player is forced to use extra time to answer their surveys (Figure 3.12(a)). They have to choose an option, and then press *next* in order to go to the next question, this is an extra step that can be avoided. Another problem with the report process is when the player meets the prompt type **number** (see section 3.5), because each time the counter is clicked, the number is only increased/decreased by 1 each time. Unless the player is typing down the number themselves, changing from the standard number (set by the campaign settings) to the maximal number of 300 can take quite a lot of time (see figure 3.12(b)). When the player wishes to send the self-assessed response, the Ohmage application will ask for GPS coordinates, hence making the player easier to identify (for example when the player is registering the survey answer at home).

3.6.4 Survey queue

If the players do not have access to Wi-Fi or cellular network, they can still answer surveys. These surveys will be stored locally on the mobile device and be shown in the **Queue** tab (see figure 3.13(a)). If the player were to not have network connection right at the moment they send their survey, the answer is stored on the phone, however, uploading has to be done manually. The error message shows no indication that the survey has been stored locally, the chances of the player thinking that they have to answer later is quite large (see figure 3.13(b)).



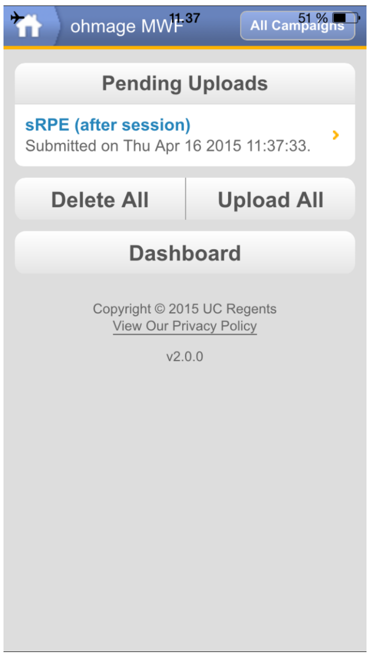
(a) Extra forced click per question


●●○○○ Tele2 NO 11.19 53 %
ohmage MWF
All Surveys
Enter duration in minutes
+
60
-
Previous
Next
Copyright © 2015 UC Regents
View Our Privacy Policy
v2.0.0

Please enter a number between 5 and 300, inclusive.

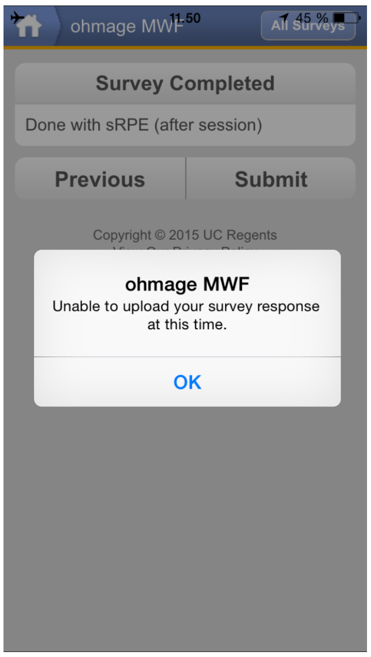
(b) Counter problem

Figure 3.12: Overview of heatmap and problems during answering surveys



sRPE (after session)
Submitted on Thu Apr 16 2015 11:37:33.

(a) Survey queue (survey sent with no network)



Done with sRPE (after session)

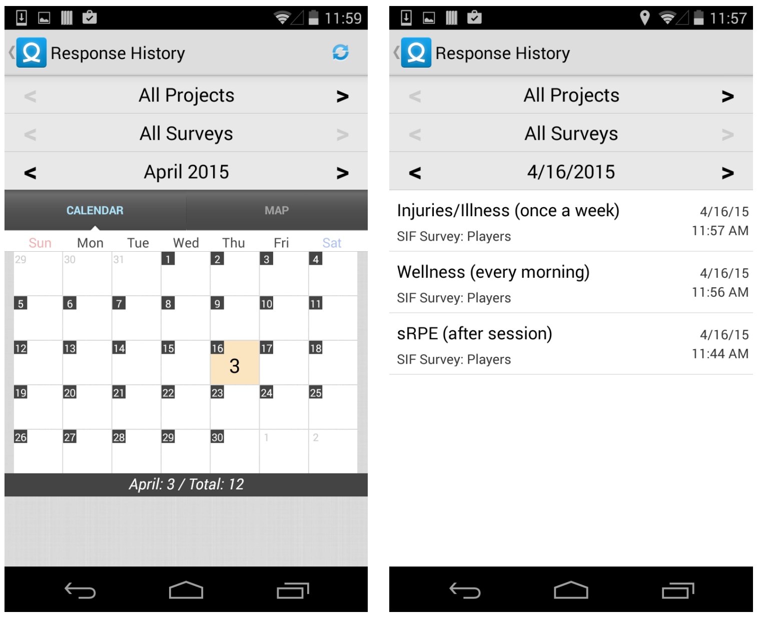
ohmage MWF
Unable to upload your survey response at this time.

(b) Error message when the user has no network

Figure 3.13: Overview of heatmap and problems during answering surveys

3.6.5 Survey history

Survey history is one of the best features Ohmage has to offer; the only problem with this feature is that it is only available on Android. This feature gives the player the opportunity to check if he has answered the surveys has they should, and this is a great feature to use when you want to know what you answered at a specific day.



Injuries/Illness (once a week) — SIF Survey: Players — 4/16/15 11:57 AM

Wellness (every morning) — SIF Survey: Players — 4/16/15 11:56 AM

sRPE (after session) — SIF Survey: Players — 4/16/15 11:44 AM

(a) Calender with a counter of survey answers (b) List of which surveys has been answered

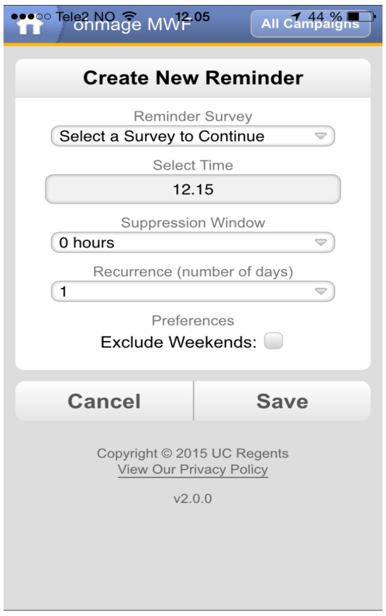
Figure 3.14: Overview of Ohmage's survey history feature

3.6.6 Notifications

It is possible to set up reminders in Ohmage, however, this feature exists on both platforms. The problem with these reminders is that they has to be setup by the player themselves, which takes time, but it is also optional. If the player has not set reminders for when to be reminded to answer surveys, then the player will not be reminded by any outside sources (like through push messages), but rather depend on themselves to remember in order to answer surveys. What this feature lacks is receiving notifications when the user is outside of the application (see figure 3.15(d)), and this is one of the biggest bugs the Ohmage MWF has on iOS. There is no point to have a reminder function if it cannot alert the user when the application is not active, and the reason is simple. The players wish to answer surveys quickly, and then do something else in their daily lives instead of keeping the Ohmage application in foreground at all time. The reminder feature on Android is hidden within the surveys, which makes it more difficult to find the feature compared to iOS.

3.6.7 Motivational factor

The most important feature and factor for usage of a mobile application to collect data, can be broken down to one simple question: **"What do I get back from using my free time to do this?"**. By using their own time to answer the surveys to give the coach data in order to be able to improve has its cost. It becomes a routine job that has to be followed in order for the quality of the data to be sufficient and useful for the system. If the player does not get any encouragement from using their own time on answering surveys, the chances for the players to give up answering surveys is much higher than for keep going on. Especially if they do not see any result over time from using a data collection tool.


●●●○○ Tele2 NO 12.05 44 %
ohmage MWF
All Campaigns

Create New Reminder

Reminder Survey
Select a Survey to Continue

Select Time
12.15

Suppression Window
0 hours

Recurrence (number of days)
1

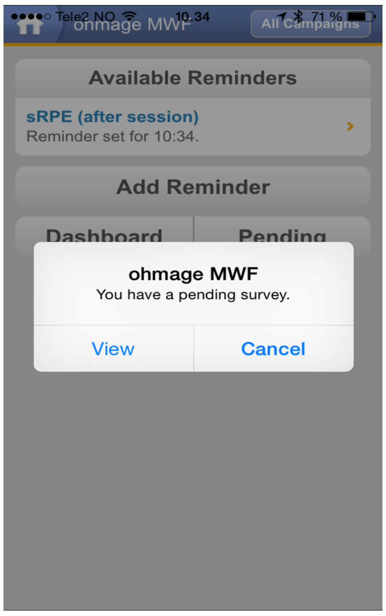
Preferences
Exclude Weekends:

Cancel | Save

Copyright © 2015 UC Regents
View Our Privacy Policy

v2.0.0

(a) Setup of a reminder in Ohmage


●●●●○ Tele2 NO 10.34 71 %
ohmage MWF
All Campaigns

Available Reminders

sRPE (after session)
Reminder set for 10:34.

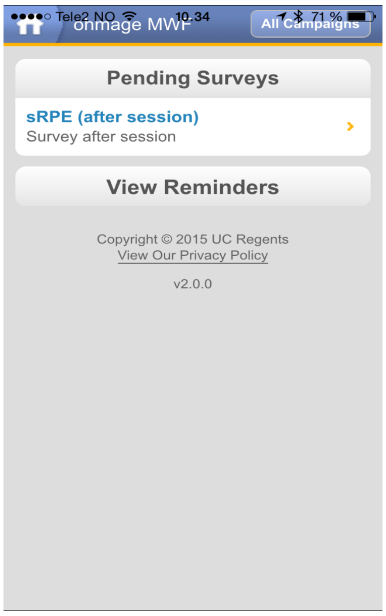
Add Reminder

Dashboard | Pending

ohmage MWF
You have a pending survey.

View | Cancel

(b) Notification while Ohmage is open


●●●●○ Tele2 NO 10.34 71 %
ohmage MWF
All Campaigns

Pending Surveys

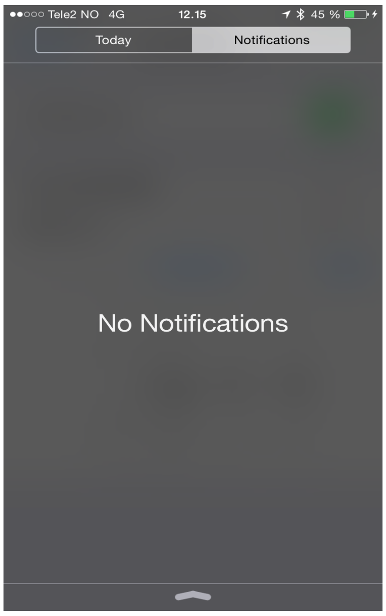
sRPE (after session)
Survey after session

View Reminders

Copyright © 2015 UC Regents
View Our Privacy Policy

v2.0.0

(c) Pending survey


●●○○○ Tele2 NO 4G 12.15 45 %

Today | Notifications

No Notifications

(d) Notification not working

Figure 3.15: Notifications in Ohmage

3.7 Summary

As mentioned in section 3.1, we use the frameworks we wanted to use, the only deviation from the original plan is the frameworks used in the rest of pmSys. This decision has proven to be the most important for the success of this master thesis, without this decision we probably would not have gotten as far as we have. pmSys is currently used by the Norwegian National team and some clubs in the Norwegian Premier league, and it's development and interest is growing.

In section 2.7, we came to the conclusion that Ohmage was the most fitting based on what pmSys needed.

Even though we have only used the back-end API, we could also use the visualization module integrated into Ohmage. The problem with this module and why pmSys does not use this module is because it is not stable enough. It is also problematic when using it, because OpenCPU has fixed parameters on how and what it accepts of data. The module practically accepted data from the system, and then sent the data to OpenCPU [42] where the data would be crunched in the cloud. With the data returned from OpenCPU, static images would be created and then returned to the user.

Ohmage and all its applications is perfect as a base system for beginners, since the implementation is so generic. Ohmage can be used to collect any type of data, but if the data has to be specific, Ohmage has to be modified to be able to collect and visualize the data more efficient. The Ohmage MWF mobile application clearly lacks some vital functionality, which makes it very difficult to be able to collect data efficient over time (section 3.6). In the next chapter, we will discuss how pmSys and all its systems and applications is using the Ohmage back-end to collect data and the functional requirements for pmSys's health monitoring.

Chapter 4

pmSys-design

In this chapter, we will present briefly all systems under the name *pmSys*. The goal of pmSys is to create an optimized digital monitoring system for football teams, with the Ohmage server as the back-end server. Hence, we need to define our functional and non-functional requirements pmSys has to fulfill. Then, we present briefly which systems will go under the name pmSys. Finally, we outline the configurations we had to do on the original Ohmage back-end to make it fulfill our needs as a back-end system.

4.1 System requirements

For a system such as pmSys to be successful as a health monitoring system, there are several functionality requirements. We distinguish between functional and non-functional requirements. These functionalities are critical for any type of system that wishes to monitor health data. That is why we chose to use the Ohmage back-end since it has already has some of these functional requirements implemented.

4.1.1 Functional requirements

The definition of a functional requirement is that the function a system and the components in the system. It may be calculations, technical details, data manipulation etc., but in this section, we will be describing the technical details of the functionality and why it is critical for pmSys that the functionality is fulfilled.

4.1.1.1 Data capture

The whole idea of monitoring health data also implies that the data has to be captured. There are many ways to capture health data, but the data is split into two groups; subjective data and objective data. pmSys currently only supports subjective data through answering a series of customized questions, and these questions are calibrated to provide the most correct data as possible when answered (as mentioned in section 2.1)). The Ohmage back-end supports objective data through third party systems, but the applications needed (Figure 2.6) to collect them is not available on neither Apple Store nor Google Play. Mobility has been removed, meanwhile AudioSens [43] and SystemSens [44] has their code repository open, but the projects has not been maintained for several years.

In order to be able to capture data through surveys presented in pmSys, there has to be some kind of motivation factor that keeps the football player interested in using the system. By providing the football player live feedback on their self-assesment throughout the season, the motivational factor increases for those who keep progressing in a positive manner. Meanwhile for the coaches, the motivational factor is to be able to monitor the physical state of the players. The coach has to enforce the system into the team's schedule in order to get a continuous data flow.

To be able to introduce new ways to capture objective data, it is important to follow the mainstream of *Internet of Things* [31] and the wearables that supports the type of data collecting that pmSys needs. More on objective data and integration in pmSys will be discussed in chapter 8.

4.1.1.2 Presentation and visualization of data

How the data is represented and visualized gives the user a feeling of whether the system being useful or not. By processing raw data from the surveys and from monitored data, and turning the same raw data into something meaningful in terms of graphs and bar charts make the system more approachable. As mentioned in the previous section, this functionality provides a lot of motivation for the user of the system. The idea of showing just enough relevant data to the user instead of overflowing the user with all kinds of data goes a long way.

4.1.1.3 System assistance for data quality

It is essential to keep a high level of quality of the captured data. In order for the data to be useable to forecast injuries and physical state, the data has to be registered every single day at the correct time. Without this meeting this criteria it is nearly impossible to predict if a player's physical state is getting close to injuring himself or if the player is already injured. Therefore it is important that the system provides the players and the coaches a way to be reminded when to register. By using the remote notification service provided by Apple and Google, it is possible to remind the player to register the days survey if the player has forgotten to answer.

4.1.1.4 Platform support

pmSys is available on the two most used mobile platforms, iOS and Android. Due to the nature of pmSys and the idea of being available for all types of mobile devices, it is also possible to use pmSys without the most advanced features through the web browser.

4.1.2 Non-functional requirements

4.1.2.1 Availability

The system has to be available to the user no matter what happens to the components or the server itself. If the server was to go down, an offline mode should detect that the server is down, and then take over storing the input data temporary until the server is back up. The user should not be able to detect that the server is down, or if the network coverage is not good enough in the area.

4.1.2.2 Usability

In section 3.6, we talked about the Ohmage MWF mobile application not being intuitive enough. This requirement is a highly prioritized function in any system that wishes to keep the user in focus. The idea is to be able to deploy the system and all the functionalities that comes with it, and without giving the user any prior training or instructions and yet be able to record quality data. For a system to become intuitive enough for anyone to use requires a lot of work, but it will also help create a critical mass of users to increase future user base, through its reputation and functionalities provided.

4.1.2.3 Scalability

For a health monitoring system like pmSys, it is important that the system can be up scaled or downscaled whenever the time fits. To be able to support an abnormal load of users in an instant, it is important that the system can support multiple parallel (concurrent) requests without showing any signs of an imminent system overload.

4.1.2.4 Privacy and Security measures

pmSys is collecting personal information from the users, and since it is hosted and maintained from Norway, we are forced to follow the rules when regarding how to manage personal information. "Personopplysningsloven" [45] is the law on how and what can be stored by the system, and it is created to protect individuals from having their private information abused. As a default in pmSys, all usernames have been created with a randomized algorithm where the letters and numbers have nothing in common with the actual user. All users of pmSys have to accept and give the system permission to collect, store, analyze and visualize their data in order to use the system. This is an important aspect of pmSys and its compliance to the Norwegian law.

```
Real user: Thuc Hoang
pmSys username: uio.2j39sh3j
```

Figure 4.1: The correlation between real name and username

In 2013, Edward Snowden [46] leaked top secret information that National Security Agency (NSA) [47] [48] were monitoring people from all over the world. Cryptography and security on social media were no longer trustable, this was because Snowden also leaked that some of the largest companies such as Facebook, Google, Microsoft, Skype and Apple had created backdoors specifically for NSA into their system so that they could monitor in real-time.

This created a massive storm about data stored in USA and on services where their headquartes were based in USA. As a result, people were looking for ways to store their data where they knew that it was going to be safe from eavesdropping.

In 2014, a huge bug that leaked information when a query was sent was discovered after being unknown since it was introduced in 2011. This open source implementation for Secure Socket Layer (SSL) is known as OpenSSL (Ohmage uses OpenSSL to secure SSL and Transmission Control Protocol (TCP) connections), where the bug was named Heartbleed [49]. The bug itself was quite simple in the way it was implemented, a length check of user input was not present, which meant that the user could extract more data from memory than what it was supposed to be allowed to do. By repeating enough the same query multiple times on different ranges, sensitive data could be returned and leaked without the knowledge of the administrator knowing about it [50].

What this means for pmSys is that the system needs to be secure enough to protect the users' data. Trusting third party libraries has a huge risk of leaking data, however, as long as precautions are made before using it, the system itself should be acceptable secured. Since we are dealing with health information, the collecting process is not top secret, but the users would definitely not want their data leaked to anyone without their consent. Therefore, before data collection can be made, the user has to give pmSys their consent to collect their data. All data in pmSys also has to be secured, and the data storage (databases) has to be protected from outside access, by restricting access based on IP addresses. It is also important that the data itself cannot be somehow linked to a player based on something related to the player, which meant that the username used has to be randomized (see figure 4.1). In the way Ohmage (section 3.5) works, it is also important that no teams can access data about each other. Team A should never be able to access data from Team B and vice versa.

4.1.2.5 Performance

There are several aspects of performance when it concerns this type of systems. It is important that the system reacts quickly when a user interacts with the system, and it is also expected that the system can handle new user creations without slowing down. Response time when requesting data from subsystems should be taken into consideration, the system should be reaching for the fastest possible processing speed when implemented.

4.1.2.6 Maintainability

For a system in the same size as pmSys, it is crucial that the code is maintainable. A requirement is that the code has to be well written, effective and commented to achieve the most optimal result, which will result in having the best chance of being easy to maintain for new developers in the future.

4.2 Architecture

pmSys consists of several subsystems and applications that provide users of the system everything from visualization, analyzation and message notifications (Figure 4.2). Each of the systems is vital to the existence of pmSys, and each of the systems fulfills a functionality pmSys provides the users of the system.

The pmSys-app is a optimized cross-platform mobile application the players use to register they survey responses. The mobile application also provides visualization of the players' own subjective data within the application, and push notification messages sent from the coach. There are only two systems the pmSys-app has contact with; the first is the pmSys-Push that the application subscribes itself with in order to retrieve push messages. The second system is the Ohmage back-end, where the campaigns and surveys are retrieved and shown in the mobile application.

pmSys-Trainer is a web portal developed with the coach and the medical staff in mind, where it allows the coach and the medical staff to analyze and interact with all the data the player has registered. The web portal provides team and single player visualization, which makes it easier for the coach and the medical staff to track a player's progression throughout a season. The web portal also allows the coach to send push notifications to players in the team, by sending the message through pmSys-Push.

pmSys-Push is the middleware-system between the pmSys-app and pmSys-Trainer which allows a one-way communication between the coach and the player. The system provides pmSys with access to Apple's and Google's push message system, and this enables to possibility to send mobile push notifications to both iOS and Android users. pmSys-Push also allows the coach to create cronjobs, which are automated tasks for when to send push messages to the players (for example a friendly reminder about registering RPE after their training).

Objective data will be uploaded and processed in pmSys-Tesseract, which is a dedicated system for just objective data from third party hardware and wearables. Algorithms process the input data, and data can be retrieved when pmSys-Trainer requests data from the system.

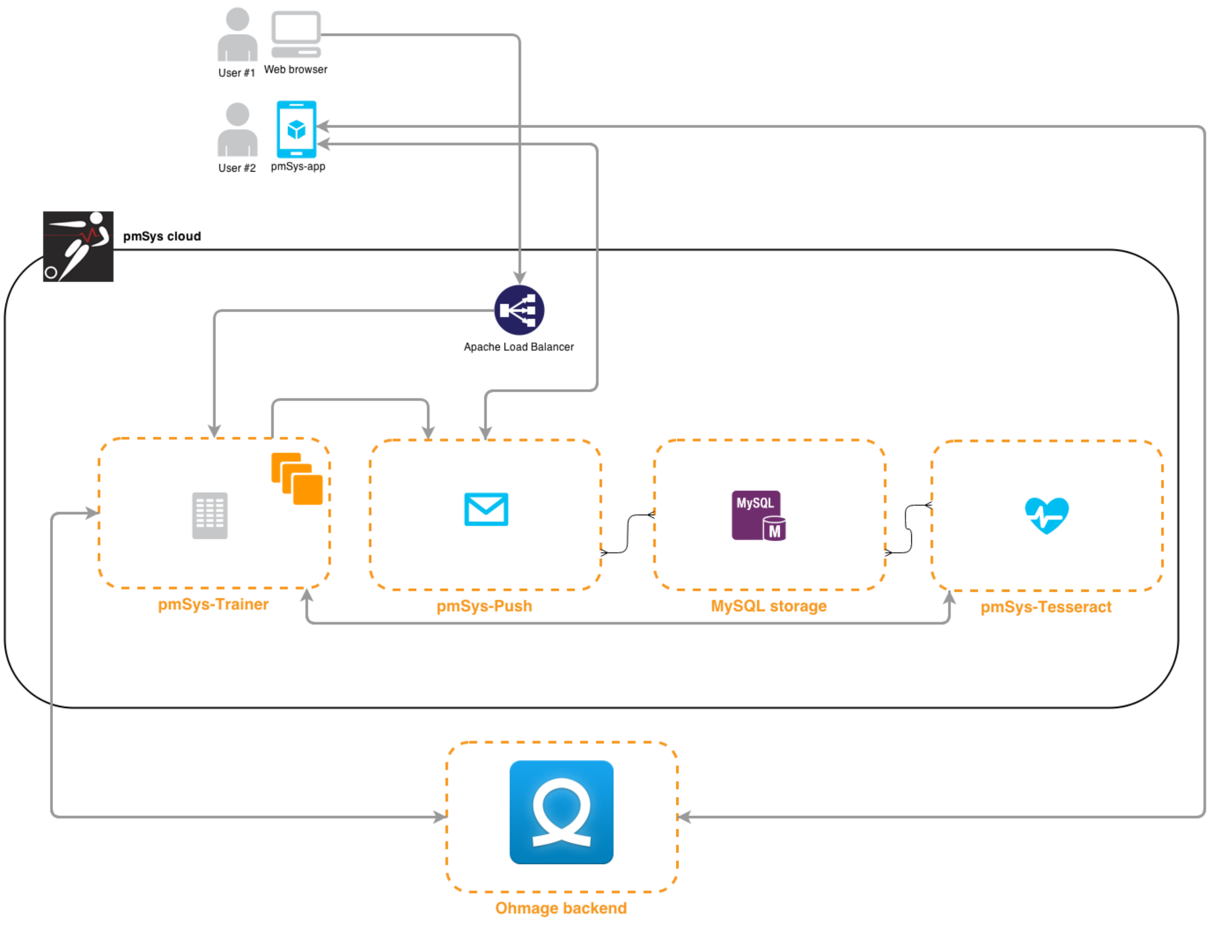


Figure 4.2: pmSys architecture

4.3 Configuration

In this section, we will define some configurations done to suit our needs with pmSys. In order to get the most optimal experience, we have found the best way to isolate the data to keep the secrecy of the users of pmSys securely and anonymous.

It is important to note that we have two servers with the same configurations, but different levels of security behind the database. The first server is our own hosted server, which we use as our test bench and stage-server before we push all our features to the main system. This has proven to be the most secure way of deploying our systems after thoroughly testing, which also has prevented a lot of bugs in the production environment (see figure 4.3).

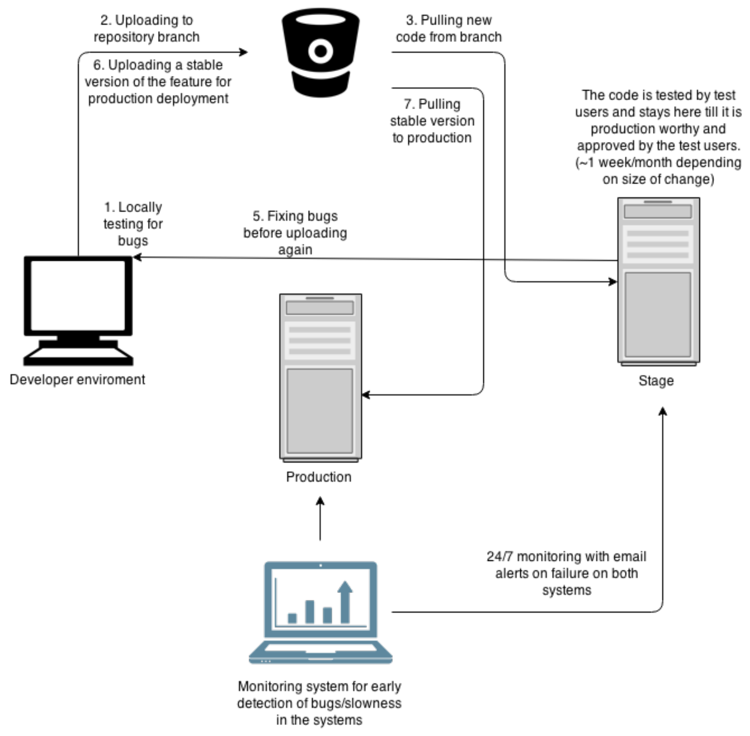
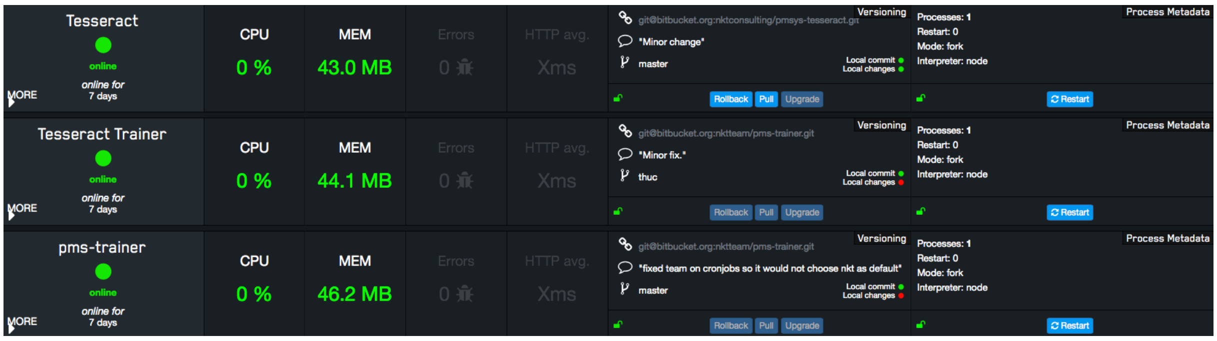

2. Uploading to repository branch
6. Uploading a stable version of the feature for production deployment
3. Pulling new code from branch
7. Pulling stable version to production
The code is tested by test users and stays here till it is production worthy and approved by the test users. (~1 week/month depending on size of change)
1. Locally testing for bugs
5. Fixing bugs before uploading again
Developer enviroment
Stage
Production
24/7 monitoring with email alerts on failure on both systems
Monitoring system for early detection of bugs/slowness in the systems

Figure 4.3: Deployment workflow for pmSys

We have also integrated a monitoring system, Keymetrics [51], which is designed specifically for real-time monitoring of NodeJS applications. We use this monitoring system to troubleshoot and detect if the system is having problems which has helped us prevent downtime on our systems, and it is also very useful when we want to pull a new source code from our repositories (see figure 4.4).



```
Tesseract            CPU    MEM       Errors  HTTP avg.  git@bitbucket.org:nktconsulting/pmsys-tesseract.git   Versioning  Processes: 1        Process Metadata
online               0 %    43.0 MB   0       Xms        "Minor change"                                                   Restart: 0
online for                                               master                                   Local commit ●          Mode: fork
7 days                                                                                            Local changes ●         Interpreter: node
MORE                                                     Rollback  Pull  Upgrade                                          Restart

Tesseract Trainer    CPU    MEM       Errors  HTTP avg.  git@bitbucket.org:nktteam/pms-trainer.git             Versioning  Processes: 1        Process Metadata
online               0 %    44.1 MB   0       Xms        "Minor fix."                                                     Restart: 0
online for                                               thuc                                     Local commit ●          Mode: fork
7 days                                                                                            Local changes ●         Interpreter: node
MORE                                                     Rollback  Pull  Upgrade                                          Restart

pms-trainer          CPU    MEM       Errors  HTTP avg.  git@bitbucket.org:nktteam/pms-trainer.git             Versioning  Processes: 1        Process Metadata
online               0 %    46.2 MB   0       Xms        "fixed team on cronjobs so it would not choose nkt as default"   Restart: 0
online for                                               master                                   Local commit ●          Mode: fork
7 days                                                                                            Local changes ●         Interpreter: node
MORE                                                     Rollback  Pull  Upgrade                                          Restart
```

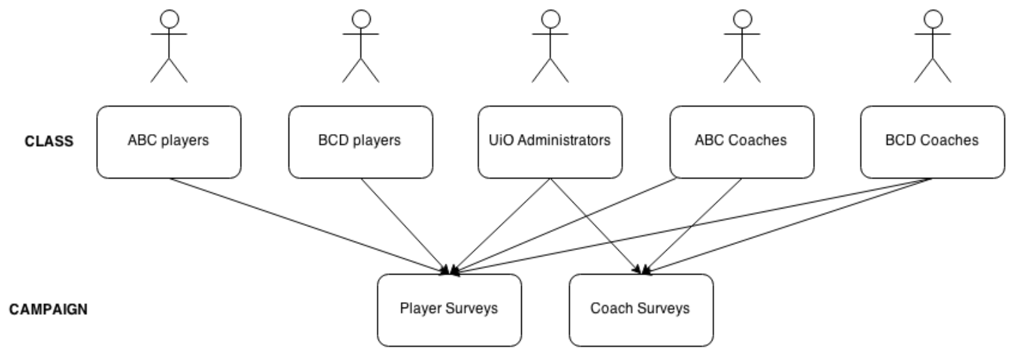
Figure 4.4: pmSys monitoring

4.3.1 User creation

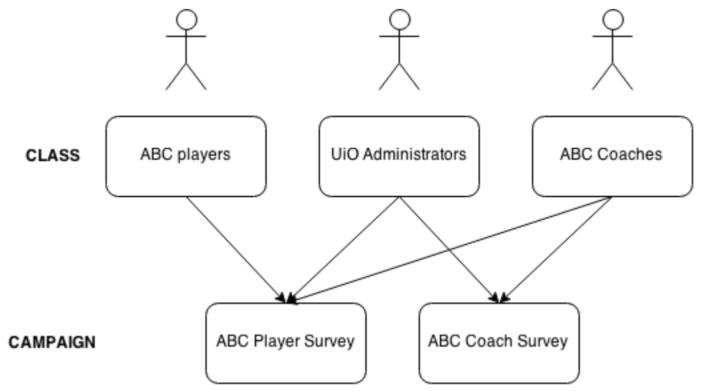
There are strict rules regarding access to personal data stored in pmSys. Which means that every time we wish to create new users on the production environment, we need to contact the system administrator appointed to create new users. There is only one person that has total access to the production server where both the web portal and the database is stored. This is to assure that we follow the rules set by the privacy laws in Norway [45] [52] [53], and that nobody from the outside knows how to access the system. Instead of using the API endpoints [54] to create the users, the system administrator has a Python script which writes directly to the database instead of going through the API.

4.3.2 Class roles

When the users have been created in the format defined in figure 4.1, we have to assign the users to the classes they belong to. In a way, we was treating classes as if it was a team, the class represents either players or coaches in a team. It is important that the players can only access their own data, and the coaches can access all their players' data. By using classes we can differentiate between the accesses given to a user. This way we can separate the access to all the data between players and coaches, and to do queries against the users of a class to create better analysis. In our first definition of the class roles (see figure 4.5(a)) all coaches had in reality access to both data of team X and team Y, but since the coaches had no access at all to the analysis tool, we did not have to account for data separation on the first batch of users of pmSys from the Norwegian Premier League. But a year later, the national team and new Tippeliga teams wanted to use pmSys as well, which forced us into finding out the best solution for data seperation meanwhile also being maintainable for any system administrator of the system. This became the current definition we have created (see figure 4.5(b)), which gives us the flexibility to do a query and simply only retrieving data from one team, instead of all the teams like in the old class definition. A trade off to this solution is that we have to create two new classes and two campaigns for each team. This makes it more difficult to keep control of all the classes, and the security of the survey data is more important then anything.



(a) Initial class definition



(b) Current class definition

Figure 4.5: Previous and current class roles in pmSys

4.4 Summary

To be able to create a system like pmSys, we had to define some requirements in section 4.1 which is vital for our system. Most of these requirements are more of less default requirements for any system, but in the case of pmSys, these are the minimum demands that we have set for our system. Now that we have defined the configurations with the Ohmage back-end, we can start to look at the implementations of pmSys applications. This includes the mobile application we have developed, internally known as pmSys-app, the mobile notification system that is known as pmSys-Push and the web portal for the coaches, pmSys-Trainer.

These systems present themselves as if it was one, therefore it is possible to consider pmSys (and all its applications) as a *distributed system*, since it is giving the users processed data from multiple systems, but presented as one. This gives the user a seamless image of one system serving them all the content, since it is either the mobile application that the players use, or the web portal where the coaches can analyze the data from their players.

In the next three chapters, we will present all the applications under the name **pmSys**, which is the mobile application, the web portal and the push notification service we have created for all users of pmSys.

Chapter 5

The pmSys-App

In this chapter, we present the mobile application that was made to replace the Ohmage MWF application. We will then present the architecture and frameworks used when developing the mobile application. Furthermore, we present the implementation of the application and its features. Finally, we will evaluate the pmSys-app versus Ohmage MWF, and present data from user studies done with objective and subjective users.

5.1 Motivation

In chapter 3, we discussed how Ohmage has functioned for The Norwegian Premier League "Tippeligaen" as their data collection system. In section 2.7, we came to the conclusion that Ohmage were very good at collecting data, but the system itself was to generic for our type of usage. Our focus-group is football players which meant the system had to be customized to be easy enough to make it worthwhile, that is why our first goals for the pmSys-App was a proof of concept. Our goals were to prove that the Ohmage mobile application could get better and more effective for our focus groups. The mobile application is one of the main sources where data can be collected, and in this case, it is subjective data that is collected through a series of survey questions. When our prototype was presented we showed the possibility of what a mobile application could do if it were customized, the project status for pmSys changed from "proof of concept" to development.

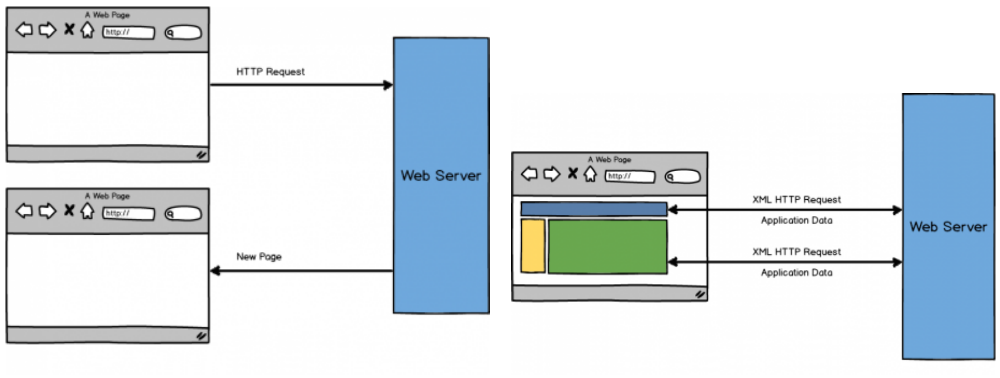
As discussed in section 2.7, the problems with the Ohmage mobile application was that it was not intuitive and hard to understand for first time users. In pmSys-app, our goal was pretty clear from the start, we wanted to make it a lot more intuitive, with more features than what Ohmage offered and make an application that were cross-platform where the GUI (Graphical User Interface) were more or less similar.

5.2 Architecture

5.2.1 Single Page Application

The old approach was to request a web page from a server, which then would be rendered on the client side (see figure 5.1(a)). This approach forced a lot of bandwidth usage, and the data on the web pages could only contain static data. The user experience was limited because there was no way to hide the latency between loading the pages, and the lack of *offline support* made the web application useless when there were no bandwidth. The second approach to creating web applications changed the way web developers was thinking, by introducing **Asynchronous JavaScript and XML** (AJAX) [55] to create more dynamic web applications.

Single Page Application (SPA) is a web application that is loaded once with the purpose of giving the user a more fluid user experience, as if it was an desktop application (see figure 5.1(b)). The definition of a SPA by Mikowski and Powell [56] can be defined as *"an application delivered to the browser that doesn't reload the page during use"*. In a SPA, web technologies such as HyperText Markup Language (HTML), Cascade Style Sheet (CSS) and JavaScript is loaded once the application is running. Interactions with a SPA often requires communication with a web server which serves the content the user requested, but this only applies to dynamic web applications that needs their content refreshed once in a while. Instead of retrieving server rendered webpages, this approach focuses on client side rendering which reduces both the usage of bandwidth over time and less detectable latency. To be able to do this, **Asynchronous JavaScript and XML** (AJAX) can be used. AJAX is not a web technology, but rather a collection of web technologies that makes web applications more powerful by giving features web applications previous did not possess, and by using HTML and CSS for rendering the view. The Document Object Model (DOM) for interaction and dynamic data display, XML to exchange data between the client and server, XMLHttpRequest for asynchronous data polling from server, and JavaScript to bind all these technologies together into one web application. In the later years, XML has been changed with JSON (JavaScript Object Notation) for data exchange since it is more read friendly than what XML is, and furthermore, it is also easier to use JSON then XML with JavaScript. Simply put, JSON can do the same as XML, but without the extensibility that XML allows, since it is not needed the way JSON is structured yet being just as powerful as XML.



(a) Traditional web application (b) Single Page Application

Figure 5.1: Overview of the history of web pages [57]

5.2.2 Model View Controller

The first Model-View-Controller (MVC) framework was created by Trygve Reenskaug and published in Xerox Parc 1979 [58]. This first implementation of MVC has influenced in most UI frameworks on how to split the application into the three parts; **model**, **view** and **controller** (see figure 5.2). This approach encourages reuse of code throughout the application, and it also makes the application easier to maintain and more scalable.

The model component represents all data in the application, and it includes everything from application states and content. If the data in the application were to change, the model can contain logic code to update the controller when change occurs, and the model can also access a database to store further data. The view component is the visual data representation of its model. A view is attached to each model component and when the model changes, the view will change to the data accordingly. The controller component is the link between a user and the application; its job is to route requests from the user to the correct model.

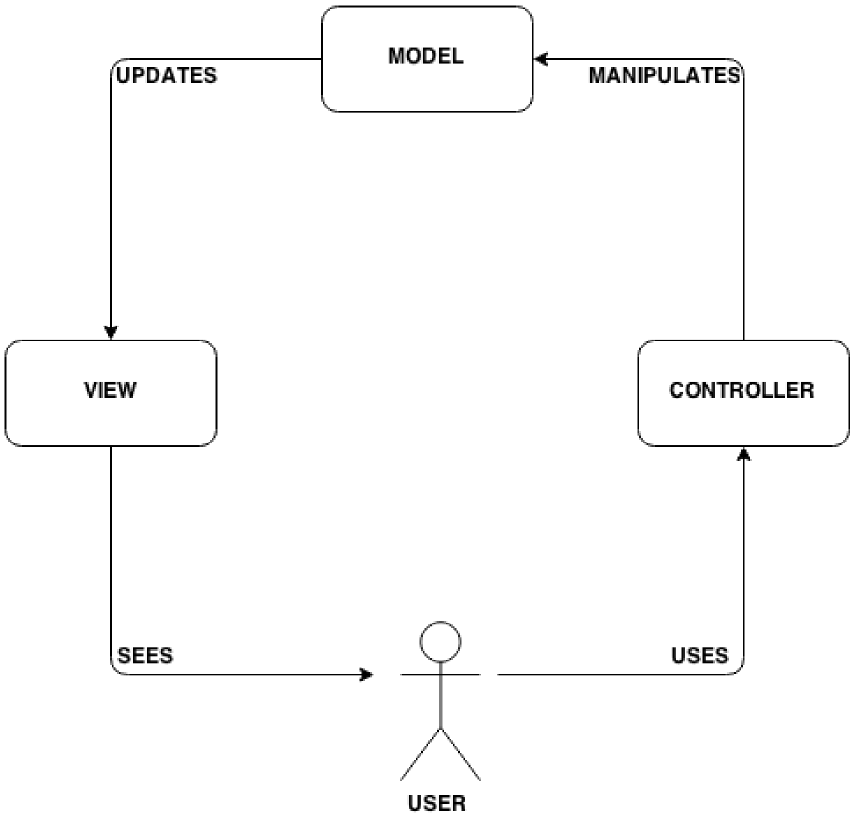


Figure 5.2: The Model-View-Controller model

5.2.3 Cordova

Cordova is a platform making it possible for hybrid mobile applications [59] to become native applications through wrapping it with Cordova. Cordova supports a set of device API's which can be used as if the application was native, and due to how Cordova works building an application for various platforms (iOS, Android, Windows Phone and Amazon Phone) they all would be similar except for the plugins which does not support certain platforms.

5.2.4 AngularJS

AngularJS [60] is a web application framework maintained by Google and several other individual developers. AngularJS provides the developer a set of tools to create single page applications, which renews how webpages and mobile applications is created compared to earlier (2000s). The goal of AngularJS is to make it easier to both develop and test code by providing a client-side Model View Controller architecture to implement the user interface. AngularJS previously used the MVC framework (see section 5.2.2), but now they follow the framework *Model-View-Whatever* (MVW) where *Whatever* stands for *Whatever works for you* [61]. The framework devides an application into 3 parts; **Controllers**, **Services** and **Views**. Features such as *Two-way data binding* gives the application the possibility to synchronize (in real-time) between the view (UI) and the model (data). This feature makes it very easy to create a responsive application without thinking of refreshing the view each time new data is loaded. Angular also provide a *HTML templating syntax* were the user can write expressions, as well as making a view inherit from another view.

Another feature Angular provides is *directives*. A directive is used to introduce a new syntax in the code (as a module), as a marker on a DOM element with a customized behavior. This enables web applications to have advanced features without using another framework for the same feature. For example a date picker, instead of using another framework or library, it is possible to just add a directive (open source or self-written) into the code to retrieve this feature.

```
<html ng-app>
 <!-- Body tag augmented with ngController directive -->
 <body ng-controller="MyController">
   <input ng-model="foo" value="bar">
   <!-- Button tag with ng-click directive, and
          string expression 'buttonText'
          wrapped in "{{ }}" markup -->
   <button ng-click="changeFoo()">{{buttonText}}</button>
   <script src="angular.js">
 </body>
</html>
```

Figure 5.3: AngularJS HTML template syntax

Even though we have listed a few features that make AngularJS excellent, there is one last feature worth mentioning. This feature is the most important and critical for any web application, and it is called *Dependency injection*. By "injecting" a service into the code, it allows an application to reuse previous code in order to give access for a section of the code. Services can be used as a feature that gives a view access to data, for example a service that stores user credentials. With a service like this, it can take care of all authentications between the user and the server it authenticates with. When injecting this service to other parts of the application, it enables the possibility to extract user credentials in other parts of the application (ex. visualization) without rewriting the similar piece of code another place. Which again reduces the size of the application and the need of duplicating snippets of code, thereof the idea of reusing previous written code.

5.2.4.1 Routes and states

When building a web application, the content is retrieved based on what URL (Uniform Resource Locator) you are requesting. There are two existing ways of routing with AngularJS. The first one is by defining the URL, of the view the application wants to show at a certain path (see figure 5.4). The other type is by combining routes and *states*, by using states makes it possible to predefine the paths to certain keywords, and then simply redirect the view there without linking to the correct URL, but instead linking to the state (see figure 5.5). Routing in AngularJS creates parameters in the URL when the user clicks on links within the application, and these parameters determines the content the application shows at all times (see figure 5.6). In the pmSys-app, we have decided to use states because of it is easier to keep track of states than the URLs when it reaches a certain point in the application for example when we have sub states.

5.2.5 Ionic

Ionic [62] is a full stack framework/platform created by Drifty Co for developing HTML5 mobile applications with AngularJS in the back-end. Ionic provides the front-end modules paired with AngularJS which takes care of how the application works (states, routes and views). Ionic does not only offer front-end but also some optimizations on the core AngularJS code where it lacks performance such as smooth scrolling [63].

```
var app = angular.module('app', ['ngRoute']);

// Route configuration
app.config(function($routeProvider) {
   $routeProvider.when('/', { // Main page
         templateUrl : 'pages/main.html',
         controller : 'mainController'
      }).when('/contact', { // Contact page
         templateUrl : 'pages/contact.html',
         controller : 'contactController'
      }).otherwise('/'); // Fallback plan in case of error
});

// Controller creation and inject Angular's $scope
app.controller('mainController', function($scope) {
   // Create a variable "Message" in the model with a message for the view
   $scope.message = 'This is the main page!';
});

app.controller('contactController', function($scope) {
   $scope.message = 'This is the contact page!';
});
```

Figure 5.4: AngularJS routeProvider example

```
.state('visualization', {
  url: '/visualization',
  templateUrl: 'app/visualization/visualization.html',
  controller: 'VisualizationController'
})
.state('visualization.srpe', {
  parent: 'visualization',
  params: ['urn'],
  templateUrl: 'app/visualization/visualizationSrpe.html',
  controller: 'VisualizationSrpeController'
})
.state('visualization.srpe.rpe', {
  parent: 'visualization.srpe',
  params: ['urn'],
  templateUrl: 'app/visualization/visualizationSrpeRpe.html',
  controller: 'VisualizationSrpeRpeController'
})
.state('visualization.srpe.load', {
  parent: 'visualization.srpe',
  params: ['urn'],
  templateUrl: 'app/visualization/visualizationSrpeLoad.html',
  controller: 'VisualizationSrpeLoadController'
})
```

Figure 5.5: The pmSys-app's visualization using state routing

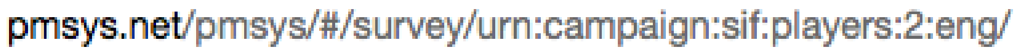

pmsys.net/pmsys/#/survey/urn:campaign:sif:players:2:eng/

Figure 5.6: URL example

5.3 Features

The goal of the pmSys-app is to improve the existing Ohmage MWF application, with emphasis on optimization of the most crucial features such as answering surveys and visualization. We have listed briefly the features the pmSys-app has to offer, and in the next sections, we will elaborate the features more detailed.

- Cross-platform (Web Browser - iOS - Android)
- Answer surveys
- Visualization
- Notification (local and remote)
- Reminders
- Change password
- Glossary
- Offline mode on iOS and Android

Functions	Answer survey	Visualisation	Push Notification	Reminder	Change password	Glossary	Offline mode
Ohmage app	Yes	No	No	Yes	Yes	No	Yes
pmSys app	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Table 5.1: Detailed list of features available in Ohmage compared to pmSys

5.4 Implementation

Everything in the pmSys-app is modulized, where each of us has been responsible for developing and maintaining if and whenever bugs occur. This is so that the application can be easier to maintain both for current developers but also for future developers. Creating large files for the whole application creates a lot of headache when coding and using revision control such as Bitbucket and Github. The pmSys-app works exactly the same on iOS and Android, but while on web browser there is no push notification, reminders or offline mode due to the nature of browsers.

For back-end services, we are using the Ohmage server (see chapter 3) as our data storage and how the users authenticate in the mobile application. The Ohmage server has a REST (Representational State Transfer) API, which allows us to do HTTP (HyperText Transfer Protocol) calls to the server for operations such as authentication and retrieving data from the system in a timely manner.

5.4.1 Authentication

pmSys-app is using both *Stateless Authentication* (often used for mobile applications) and *Stateful Authentication*. The reason for this is due to how the server is built. Out of the box it is supposed to allow the user to use either Stateless or Stateful, but in reality, some REST calls do not support Stateless.

Stateless Authentication allows the mobile application for a one-time authenticated hashed password. The hashed password is created by hashing the username, password and a salt string together and then returned back to the HTTP call.

Stateful Authentication generated a token that lasts for 15 minutes by default; this default has been hardcoded into the server (the documentation states otherwise). The time limit set on the token is too small to be usable in a mobile application where it needs to fetch data from certain HTTP calls, which does not support Stateless Authentication (Table 5.2).

Method	Authentication	User manipulation	Classes	Campaigns and surveys
Stateless (hashed password)	Yes	Yes	No	Yes
Stateful (time limited token)	Yes	Yes	Yes	Yes

Table 5.2: Authentication type table

5.4.2 Login

The Ohmage MWF application had problems with adding custom server URLs, and the problem raised in section 3.6.1 were that Ohmage did not support malformed URLs, and IP addresses. Another problem the application had were that once the path to the back-end were wrongly entered, the application had to be reset in order to be able to re-enter the path again. This has been fixed in the pmSys-app, where the data is persistent even after a update of the mobile application.

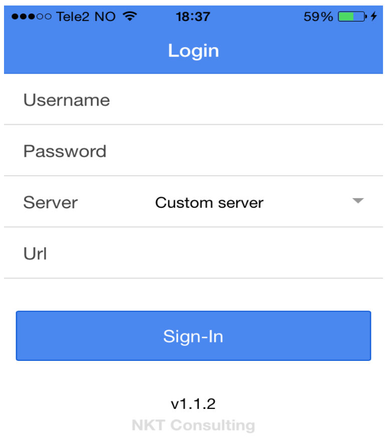


Figure 5.7: Login page with custom server enabled

5.4.3 User categories

pmSys-app categorises all users, this is similar to how Ohmage categorize their users (section 3.3.2). Priviliged users in Ohmage are coaches on pmSys in general, while Restricted are the football players.

5.4.4 Campaigns and Surveys

While on Ohmage's official mobile application you need to download and add the campaign (that has already been allocated to your account), the pmSys-app immediately retrieves the surveys in a campaign you have access to displayed in your mobile application as soon as you log on. This was a feature that helped the deployment process go easier, and saved both the coaches and the players a lot of time when they needed to refresh the program list.

When the campaign data is retrieved from the Ohmage back-end, the data is returned in XML-format. To make the data easier to work with and to be usable, we have to convert it into a JSON object. To do this we have used the library, x2js [64]. Access to campaigns is decided through the use of classes (see section 3.4), and that is why it is important that the users has been added into the correct classes for any campaign to show up in the mobile application.

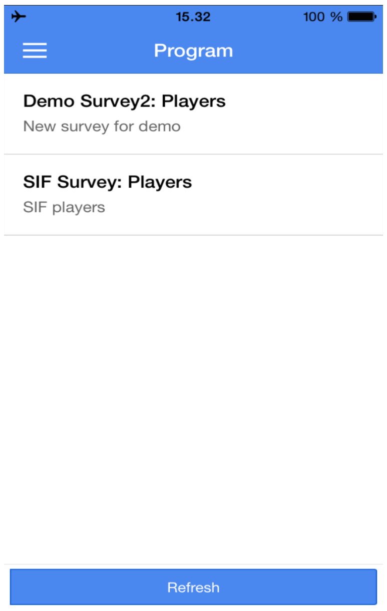


Figure 5.8: Programlist when logged into pmSys

5.4.4.1 Prompts

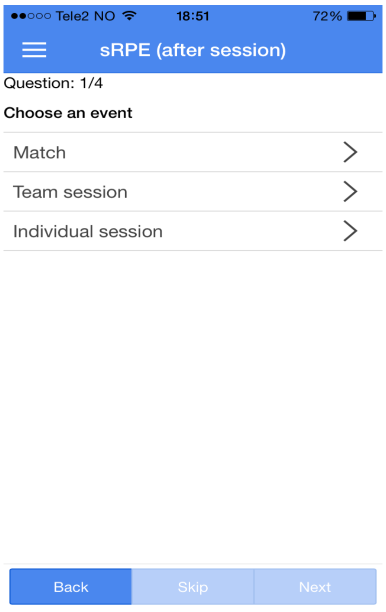
In the pmSys-app we are only using single choice, number and timestamp prompts, with only these three prompt types we can cover all our needs when it comes to creating surveys in pmSys. When we tested the Ohmage mobile application there was an extra step (section 3.6.3) for each question, which means only seconds extra.

Seconds can turn into a lot more over time, and our solution to fix this nuisance is to remove that step on the single choice questions in order to save that extra step. We have also optimized the number prompt, from having a interval of one per click, we have increased the interval to five since training sessions are usually rounded up to the closest five or ten (ex. 55 minutes). The results from the optimizations of the survey answer process has decreased the time needed when answering by nearly 50% (see section 5.6.5) with the pmSys-app.

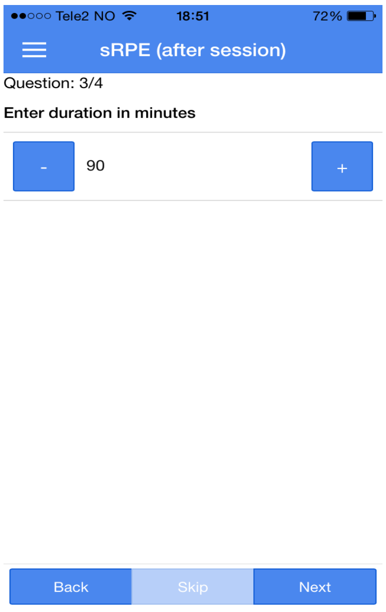
5.4.5 Survey response registration

The whole goal of the pmSys-app is to optimize the process of reporting after training sessions, by answering self-assessment surveys. The reporting process has been optimized to the point of avoiding unnecessary steps. Hence, the player can answer surveys in matter of seconds and focus more on developing their football skills.

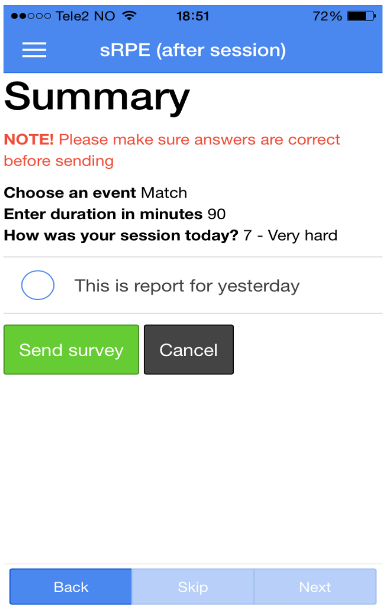
A survey is defined in a campaign, where one campaign can contain multiple surveys (see section 3.5). A campaign with surveys must be defined and uploaded in a XML-format (see figure 3.7) to the Ohmage back-end, and then the system administrator has to attach classes to the campaign in order to see the surveys in the pmSys-app. A normal player would only require access one campaign since they can only belong to one team. But it is possible to be a member of multiple campaigns if the player is added, for example if the player has been picked out to play for the National team.



(a) Single promt



(b) Number promt (interval set at 5)



Summary

NOTE! Please make sure answers are correct before sending

Choose an event Match
Enter duration in minutes 90
How was your session today? 7 - Very hard

This is report for yesterday

(c) Summary page

Figure 5.9: Example of a survey in the pmSys-app

In figure 5.9, the most vital part of the survey is shown. There is an extra feature on the summary page, where the user can register for the previous day if the player has forgotten to answer the survey. The pmSys-app currently supports number, timestamp and single choice, even though Ohmage has support for many others (see section 3.5). During the survey registration, a JSON object with the syntax Ohmage requires [65] is created. When the player has reached the end, the summary page is shown (figure 5.9(c)) where the content is the parsed data of the JSON object (see figure 5.10). When the player presses on "Send survey", the JSON object is sent to the Ohmage back-end and registered as a survey response for the specific campaign. Compared to the Ohmage MWF application (see section 3.6.3), the pmSys-app will **never** store or try to localize the player through GPS, and the reason for this is the privacy rules in Norway and the requirements set in chapter 4.

```
{
  "survey_key":"8146f9e9-8136-40d3-98e5-bd2e93bdcf41",
  "timezone":"GMT+02:00",
  "location_status":"unavailable",
  "survey_id":"srpe",
  "privacy_state":"private",
  "responses":[{
    "prompt_id":"srpeType1",
    "value":"1"},
    {"prompt_id":"srpeType2",
    "value":"0"},
    {"prompt_id":"srpeLength",
    "value":90},
    {"prompt_id":"srpeWorkload",
    "value":"1"
  }]
}
```

Figure 5.10: An example of the JSON object in required syntax

5.4.6 Visualization

The Visualization module does data presentation in the pmSys-app. For the graph plotting, we use an open source JavaScript library called NVD3 [66]. This module fetches data from the Ohmage API, and then presents it in a lucid way as shown in figure 5.11. The visualization module is our solution to the *motivational factor*-problem discussed in section 3.6.7. In order to motivate the players to use their own personal time to register their surveys, we have created a range of different graphs with analysis of their subjective data.

Due to the nature of the data returned after fetching from the Ohmage API, the data has to be sorted and parsed (step 2 in figure 5.11) before it can become useable with the NVD3 module. Firstly, the data has to be matched to the player. Secondly, the date of the survey answer has to be appended to the value the user answered, or else the data can become inapplicable for visualizing (faulty sorting). In order to use the data with the NVD3 library, the data has to be sorted into an array.

The presentation of the subjective data is presented with graphs with the terms RPE load, monotony and strain discussed in section 2.1. The graphs gives the player a simple (figure 5.12), but detailed analysis of what their perceived intensity of a sessions throughout a day. This motivates and allows the players to monitor their own progress, instead of just submitting reports.

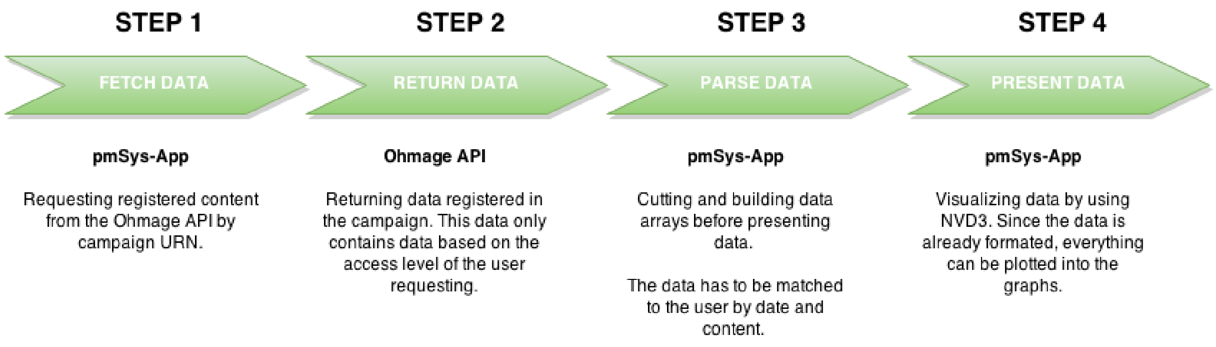
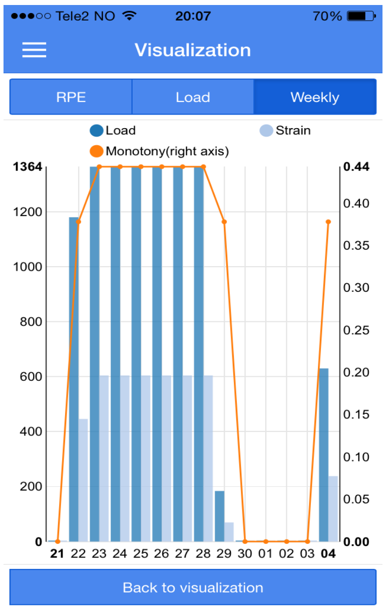

STEP 1
FETCH DATA
pmSys-App
Requesting registered content from the Ohmage API by campaign URN.
STEP 2
RETURN DATA
Ohmage API
Returning data registered in the campaign. This data only contains data based on the access level of the user requesting.
STEP 3
PARSE DATA
pmSys-App
Cutting and building data arrays before presenting data.
The data has to be matched to the user by date and content.
STEP 4
PRESENT DATA
pmSys-App
Visualizing data by using NVD3. Since the data is already formated, everything can be plotted into the graphs.

Figure 5.11: The workflow of the Visualization module

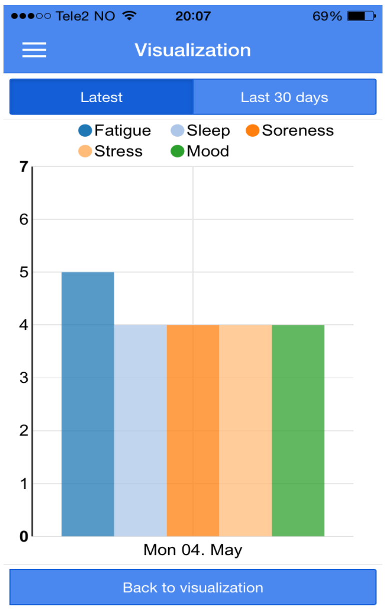
5.4.7 Notifications

The pmSys-app receives remote push notifications from pmSys-Push (see chapter 7) where the coaches can send messages from our webportal; pmSys-Trainer. The push message is delivered instantly after being sent to the player (see figure 5.14), and the requirement for receiving remote push notifications is that the players has to allow push messages, and that their mobile devices has an internet connection. If not the message will be stored for two hours in the notification platforms provided by Apple and Google, before being deleted permanently.

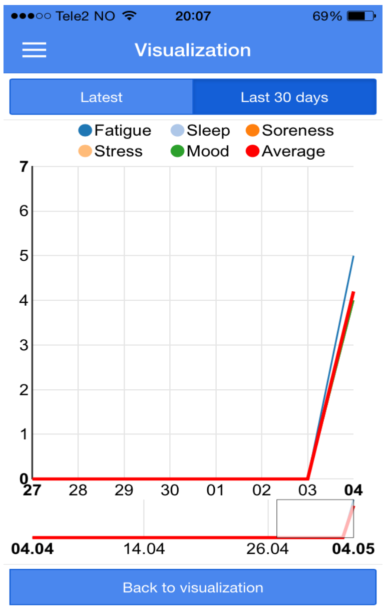
pmSys-app also supports local notifications [67]. On the mobile application on both platform it's known as Reminders (see figure 5.13). Reminders is using the plugin which imitates how the native calendar behaves, the only way to remove a reminder is to either restart the mobile phone by power shutdown, or canceling through the mobile application. Every time the mobile application starts up, the application will subscribe itself with APNS, and then the returned data (token) is sent to our push service in order to keep the device token refreshed all the time.



(a) A week analysis of RPE load, monotony and strain

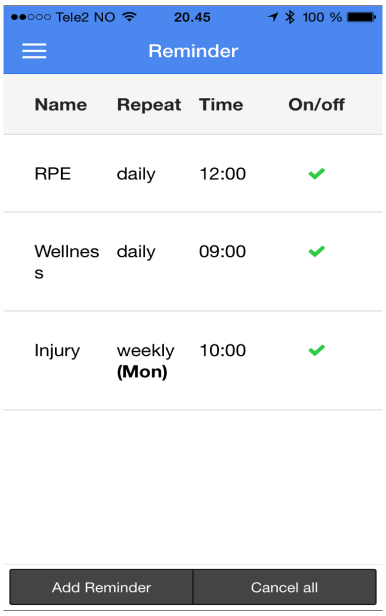


(b) Latest registrated wellness data



(c) Wellness for a whole month (scrollable)

Figure 5.12: Visualization of survey responses in the pmSys-app

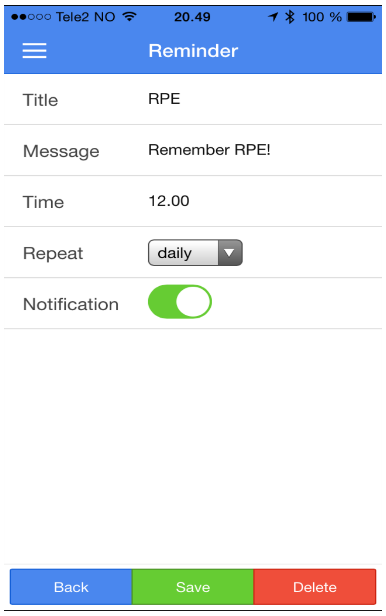

●●○○○ Tele2 NO 20.45 100 %

Reminder

Name	Repeat	Time	On/off
RPE	daily	12:00	✓
Wellnes s	daily	09:00	✓
Injury	weekly **(Mon)**	10:00	✓

Add Reminder | Cancel all

(a) List of notifications


●●○○○ Tele2 NO 20.49 100 %

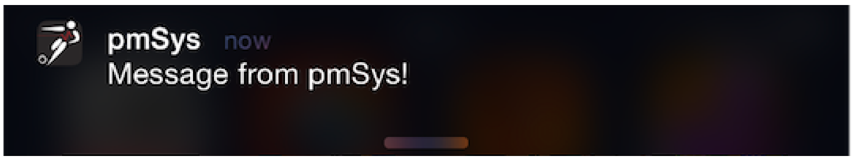
Reminder

Title	RPE
Message	Remember RPE!
Time	12.00
Repeat	daily
Notification	On

Back | Save | Delete

(b) Sample of a notification

Figure 5.13: Overview of the (local) notification feature


pmSys now

Message from pmSys!

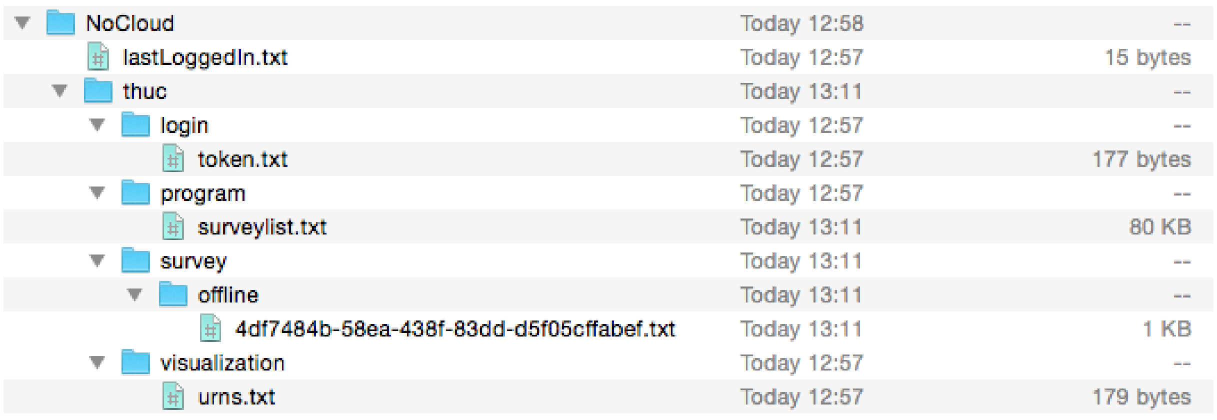
Figure 5.14: Receiving push message on iOS

5.4.8 Cordova Filesystem

pmSys-app is using Cordova's JavaScript implementation [68] of the native device's file system through their own plugin: file. This plugin provides read and write operations towards specific file system paths on all operative systems, but for pmSys-app that means iOS and Android. The supported functions by the plugins are quite wide and are easy to use once you get the hang of it. This allows us as developers to provide the same JavaScript code for both platforms even though they point to different file system paths on the native device:

```
function readFromFileSystem() {
  window.resolveLocalFileSystemURL(cordova.file.dataDirectory,
     function(fileSystem) {
    // iOS = /var/mobile/Applications/<UUID>/Library/NoCloud
    // Android = file:///android_asset/files
    // Data returned in variable fileSystem contains all files and folders
       in the native's filesystem
  }, callback);
}
```

pmSys-app creates files on the mobile system in order to be able to perform the features it supports such as offline mode (see figure 5.15), and saving the need for polling data from the back-end (battery and bandwidth draining). There are other ways to store data on mobile devices, more on that is discussed in section 5.6.



```
▼ NoCloud                                                Today 12:58        --
    lastLoggedIn.txt                                     Today 12:57  15 bytes
  ▼ thuc                                                 Today 13:11        --
    ▼ login                                              Today 12:57        --
        token.txt                                        Today 12:57 177 bytes
    ▼ program                                            Today 12:57        --
        surveylist.txt                                   Today 13:11     80 KB
    ▼ survey                                             Today 13:11        --
      ▼ offline                                          Today 13:11        --
          4df7484b-58ea-438f-83dd-d5f05cffabef.txt       Today 13:11      1 KB
    ▼ visualization                                      Today 12:57        --
        urns.txt                                         Today 12:57 179 bytes
```

Figure 5.15: iOS file system

5.4.9 Offline support

The pmSys-app has support for offline usage of the application. And by providing the users offline support, they can still answer and store their answers locally on their phone until the mobile application has network again. The reason for this feature is due to the network coverage available in Norway, there is no guarantee that there will be good network or any network wherever the user may be at the moment they answer their survey. Once the mobile application detects that network coverage is missing, the application immediately disables all the features that requires network in order to work. This includes being able to change their password, visualization, fetching new surveys and updating campaign information.

By standard programmed behavior, surveys will be stored in a file locally on the mobile device in case network is missing, that way it is possible to load the surveys the user need to answer the daily surveys. When the mobile application detects that the network is back meanwhile the application is turned on or resumed, it will be checking the file system for unsent survey answers. If any survey is found in the folder *survey* (see figure 5.15), then it will send all the survey answers in the folder before recursively deleting the files after successfully sent.

5.4.10 Bypass functionality on startup of application

When running the mobile application, the variables are filled with data. Global variables needed has values when logging on for the first time, however, once the mobile application is suspended or turned off, all the data will be removed. The next time the application is turned back on; it will simply restart the application. pmSys has randomized usernames and passwords, however, if the user were to be forced to log into each time, it could take a lot of time before they can answer a survey. As a result, this also removes the motivation to use pmSys altogether. Without the players, no data can be gathered to provide visual analysis throughout the system.

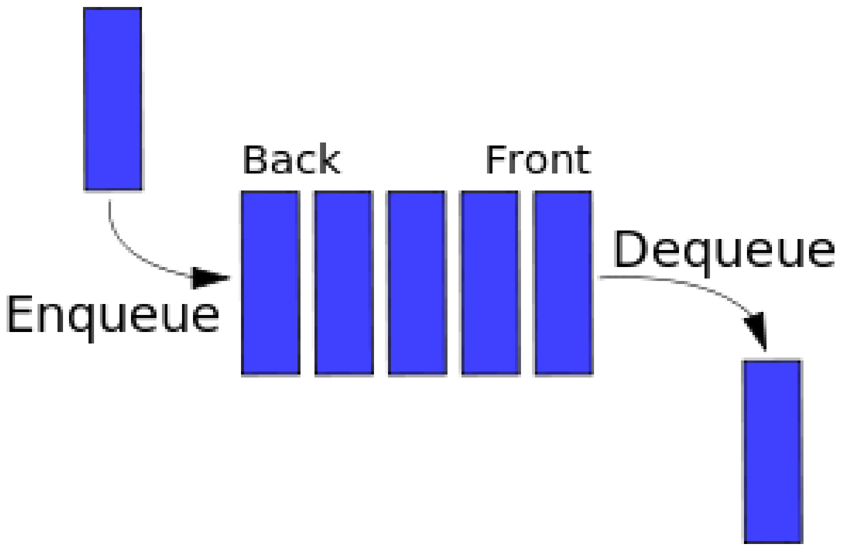


Figure 5.16: Memory stack on iOS for background applications

On iOS, the application is suspended when it is kept in the background of the operative system. All functions is put into sleep mode, and there is possible to wake up the application by sending an interactive push message which can force the application to download new data from a URL. When the mobile device is running out of internal memory, the operation system will automatically remove the first program in the list to free memory [69] [70](see figure 5.16). For a long time we thought that the application timed out since we were dealing with *Stateless authentication* (section 3.3.1), the way we routed in the application and the possibility of the state of the application and AngularJS did not go well together. Since our biggest user base were using iOS and we managed to reproduce this problem every day, and after a deeper and more thorough debugging the bug were discovered. This is why we have created a bypass function that practically fills out all the global variables that we need to make the mobile application work.

When the user opens the pmSys-app (see figure 5.17), it checks if whether the application has credentials saved (global variables and login information stored on file), if the application is resumed it will redirect the player to the "Program" page. If the application started from scratch, the application will fill out the global variables with the saved data on file. Once this has been done, it will redirect the user to the "Program" page. If there is no credentials available, it will redirect the user to the "Login" page.

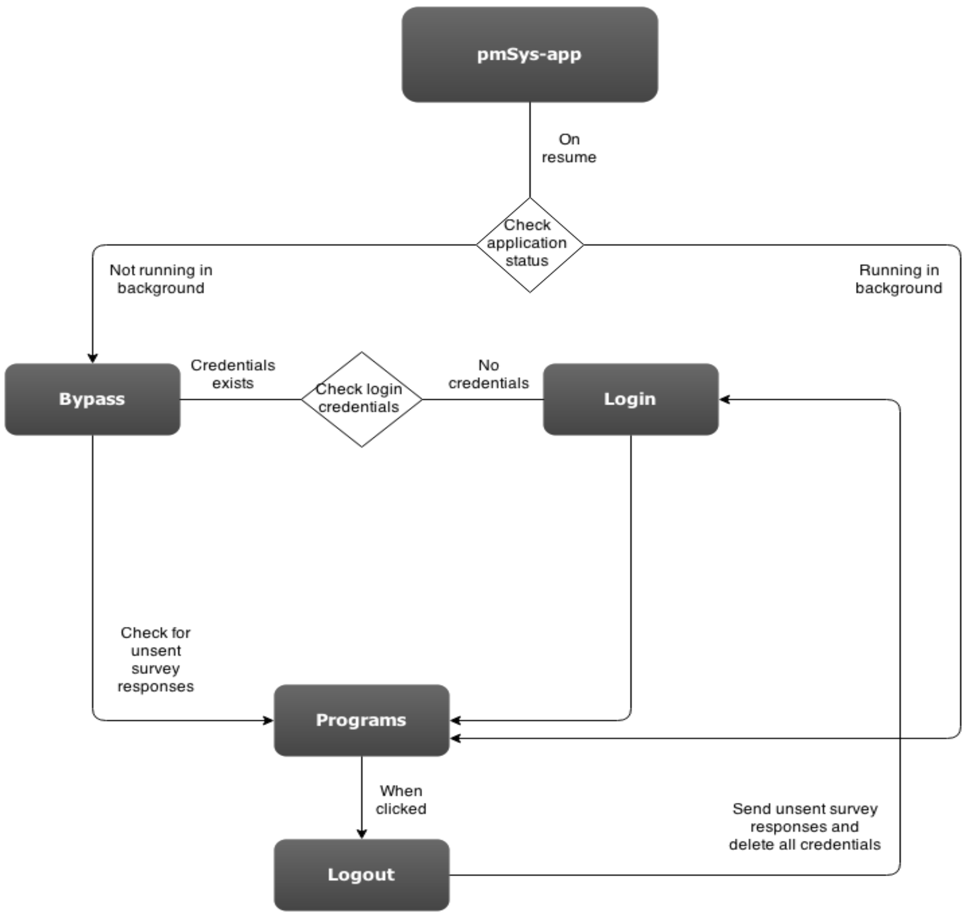


Figure 5.17: Bypass workflow

5.4.11 Glossary

A glossary feature is added into the pmSys-app with glossaries for the terms used in the application are explained. The glossary is available in Norwegian and English even though the application is in English. Some football players is better at reading and understanding Norwegian, hence the glossary covers both languages while explaining difficult terms and why a specific survey is important.

5.5 Deployment

5.5.1 App Store (iOS)

The deployment routine for iOS is one of the most difficult tasks to do for first time mobile application developers [71]. There are many guidelines provided by Apple which every mobile application developers has to follow, for our case we also needed to know how long it would take to deploy an application to App Store both for synchronization with Google Play and for predictions when to roll out the latest versions. Apple requires all applications to be signed with a valid Apple Developer account (100$/year) which will be responsible for all faulty and bugs associated with the application. It's impossible to falsify the signature associated with an account since each signing certificate must be issued by Apple to become valid in xCode [72].

When we deployed pmSys-app for the first time (September 22th 2014), it took us 14 days (included the weekend) to get our application approved on our first try. The first version was deployed to App Store October 6th 2014, but the way to send it for review was a nightmare. Apple was right in the middle of a big change on both OS versions and how to deploy applications, which resulted in outdated guides on how to do this properly. Due to this, it took us four days to prepare the application and the description before sending it for review at Apple.

To our surprise, it would take up to 14 days for pmSys-app to change application status from "In Review" to "Approved", and then another 6 hours for the application to propagate to all App Store app storages throughout the world for download. Which means if we were to deploy a fault version of the application or simply quick fix it, it would in the worst case take us two weeks to fix the application for the users. Two weeks with the planned schedule of answering surveys meant 28 surveys lost per user in pmSys-app if our app were to be deployed with a faulty function where Apple could not find it when they have approved it. Because there is always a chance for a double glitch in both the automation Apple has and the UX developer Apple allocates for review. This meant that every deployment has to be of high-level quality before pushing it to Apple, the kickback from deploying a faulty version could mean quite a lot for the reputation and the data collection for pmSys.

For pmSys, a normal developer account has been purchased (99$/year) to deploy the application. There is possible to pay for an "Enterprise" certificate which costs 299$/year, which makes it possible to deploy new updates as fast it is possible to update all the CDN's Apple uses [73].

5.5.2 Google Play (Android)

Google Play [74] has a fully automated system for all new mobile applications, which means that there is no further quality check than the system itself. This is also one of the problems with Google Play that nearly all types of mobile applications can get accepted, which makes it harder to be taken serious once a name has been abused. Before we tried to deploy to Android we tried to find out how long it would take to deploy to Google Play, however, there was no distinct answer. When we actually tried it, it us took less than 15 minutes to go from "In Review" to "Approved", but it took 4 hours before starting to show up on the global Google Play lists.

Due to the nature of how Google Play works quickfixes can be pushed out extremely fast, but the problem is that the functionality syncronization between Android and iOS users could be vastly different (which again makes it harder to give a general support on both platforms). Updates of the application takes less than two hours to be avaiable for all Android users, which are extemely fast compared to how Apple operates. In a way it is more liberating to have that power to push new versions so quickly, but "with great power comes great responsibility" [75].

5.6 Evaluation & Discussion

pmSys-app is a hybrid HTML5 application which allows the developers to create a simple mobile application with little to no knowledge of memory pointers or how to prevent memory leaks, which is needed if we were to build any application with native code. We decided to use Cordova due to how it packed our web application so it could run on the phone. By using Cordova, we can easily say that we have shorten the time used to develop by tenfold, the reason is simple; We do not have to create the same application in different languages. But only focus on one application for all platforms we wish to support.

For iOS you can either create applications by writing it in Objective-C or Swift, while for Android you has to write the application in Java. It's obvious that the languages does not have the same semantics and they do not have the same function names and such, which would result in a lot of time learning these languages for each developer to maintain either focus on one platform or both platforms while trying to create a fix for bugs in a timely manner.

5.6.1 Hybrid versus Web versus Native application

There are endless of discussions of why a native mobile application is better than a web or hybrid application. It simply breaks down to the fact that it has access and advantage of all the device features [76]. Even though the arguments may have not changed for why to build native applications, the reasons why a developer should be building a web or hybrid application has.

Mobile OS type	Programming language required
Apple iOS	C, Objective C, Swift
Google Android	Java
Windows 7, 8 Phone	.NET

Table 5.3: OS and their programming language

A native application is an application written in the operative systems language (see table 5.3), it does not need any third party frameworks to wrap the code around in order to make it work on the platform. Since it is written in the OS's language, it has full control of all the features a mobile device can offer. Camera, GPS, accelerometer, compass or contact list access is just a few of the many functionalities a mobile device has. It has to be installed on the mobile device before it can be run, which again limits the portability of the application compared to a web application.

A web application is different from a native application. A web app is in reality websites created to look and feel as if it were an application, by using HTML and JavaScript any website can become a web application. Rather than installing the application on the mobile device, it can be run through a web browser, that way it can be accessed on any device (also desktops) without problems. The limitations of a web application and any website for that matter, is the fact that it cannot access all the functionalities a mobile device has to offer. It is impossible for a website to access the contact list on the device, which makes the usage of such applications limited and more often used for desktops only.

Meanwhile a hybrid application is a mix of both native and web application. It is a web application encapsulated in a native wrapper that loads the web application in the web browser at runtime. This way the application gives the user a native feeling since it can be installed through the app stores. The access to a mobile device's functionality is limited to the plugins created for a specific platform.

In the case of the pmSys-app, we chose to create a hybrid application, this was due to several reasons listed in table 5.4. The best part of building a hybrid application is that it is possible to write one web application wrapped into Cordova, and then deploy it to many other operative systems, this also includes deploying it as a website.

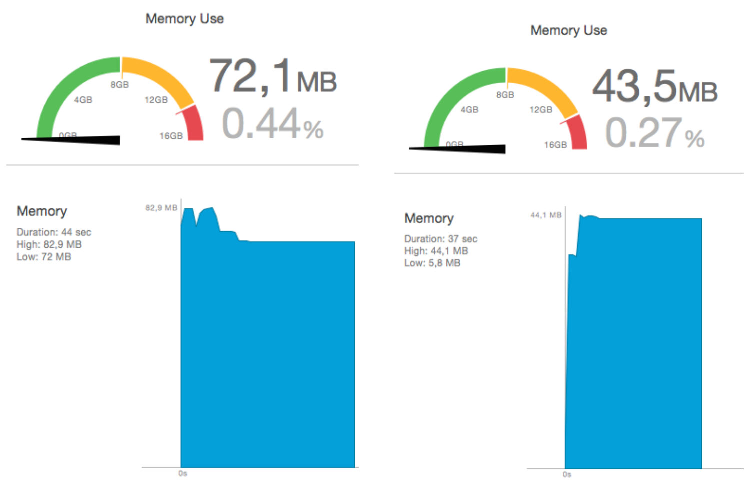
The biggest reason why we created a hybrid application is due to our limited knowledge of Objective-C (iOS) when we started this thesis. To create an application as native takes time, and if we were to code native, we had to create the same across both platforms we wanted to support, which is iOS and Android. By creating a hybrid application, we saved a lot of time while sacrificing little to nothing due to the size of our application. If our application was a big application with limited resources, we would have considered building it as a native application. Facebook invested a lot of time into creating an hybrid application, which costed them a lot of wasted invested time [77]. In defense, this was in 2012, the frameworks has been updated since then and this statement might not even be entirely true anymore. Optimization of the frameworks has been done, and in our case, Ionic has focused a lot on optimization

Pros	Cons
Maintainability	Not suitable for complex mobile applications [77]
Plugins that support needed functionality	Performance issues on a few features with the current framework [63]
Same programming language for all platforms	Less control of device native functionality
Build **once**, deploy **everywhere**	
No memory leaks, Cordova takes care of optimalization	

Table 5.4: Pros and cons of pmSys-app as a hybrid application

and fixing the problems web applications had before with sluggish loading. The reason for this sudden boost of performance is not entirely entitled to the frameworks, but also to the OS'es. In Apple's iOS8 that was released in 2014 made a simple change where hybrid applications could finally use the same JavaScript engine as Apple's native applications.

The engine for all applications below iOS8 uses *UIWebView* [78], limits the processing power of the application. The reason for this is that Apple uses a JavaScript library, Nitro (previously known as SquirrelFish), which boosts the JavaScript operations in Safari (mobile version only). This was due to security measures that had to be taken because of the way Nitro is built, JavaScript code is compiled and run as JIT (Just In Time) [79] compilation. This meant that the JavaScript were not compiled and translated before the application started, but during the application runs, in real-time. This meant that if Nitro was exposed to malicious code, the mobile device could do a lot of harm since it would then have direct access to memory. With the new access granted in IOS8, any application has access to *WKWebView* [80] with a major boost in performance. A bug in iOS8 made Cordova developers fall behind the native applications since Cordova could not utilize the newly acquired JavaScript interpreter engine. As a temporary fix to this bug, a Cordova plugin has been created to fix this problem until iOS8 has fixed the problem [81]. What this means for the pmSys-app is that the new WKWebView makes hybrid application just as fast as native applications, especially when iOS fixes the bug. This does not mean that hybrid applications is the best solution for all types of applications, but for applications where the functionality is not too dependent to the mobile devices functionality, by create hybrid application you might save a lot of time.



(a) UIWebView (b) WKWebView

Figure 5.18: Overview of the memory usage of pmSys

5.6.2 Data storage

We decided to use the filesystem API provided by Cordova, and the reason for this was the great support of the API itself. Everything was well documented and easy to test manually since the files are visible if you have access to the file system. Even though the file system API is simple to use, it is also possible to use other data storage frameworks like SQLite [82].

Instead of creating numerous files like how the pmSys-app is developed, you simply do normal SQL queries against the SQLite database, which can contain a lot of data through BLOBs (Binary Large Object). BLOBS is a collection of binary data stored into a single field in a database management system. At the time this feature was implemented, SQLite was not prioritized nor considered due to our own bias towards Cordova and all the plugins the framework had to offer. Therefore, when looking back at how storage of data is currently handled, SQLite might be a better suit for the pmSys-app, even though the current way work seamlessly.

5.6.3 User case: Which application is preferred

In this section, we present and then discuss the results from our user studies. The groups is split in half, whereas one half is with users that has never seen or used Ohmage or pmSys and will be categorized as objective users. The second group is the two most active teams on pmSys, Lørenskog Idrettsforening (LIF) and Strømsgodset (SIF), that has used pmSys and they will be rating the application after their own subjective user experience.

All the questions has these possible values, which is inspired by the Likert-Scale [12], to choose from unless it is a Yes/No-question:

- 1 - Poor
- 2 - OK
- 3 - Good
- 4 - Very good
- 5 - Excellent

5.6.3.1 User study 1: Objective users

We selected 27 independent and objective students from multiple schools in Oslo, such as University of Oslo, University College of applied studies and the Norwegian Business School. This was to make sure that the results from the application could be as wide and objective as possible, especially with people with little to no knowledge on mobile application development. From this user study, we retrieved a lot of useful information about our application versus Ohmage. Most users liked in general everything pmSys had significantly more than what Ohmage could offer, which only means that as a first impression, pmSys did pretty well at being a intuitive application with little to no prior experience or knowledge.

One of the comments we got were; *"pmSys is a much easier and more detailed system to use. It seems more modernized with features that makes it easier to use the application. The visualization is very good, and I liked that I could see my activity statistics for the last 30 days."*. Another one said *"The surveys in pmSys is better constructed. It is easier to select an answer and get through the survey. I liked the summary page. Very nice with graphs and tracking the data."*. The comments is mentioning the whole motivation factor with the pmSys-app, and this shows that the visualization is a feature that makes the user more interested in using it over time. In the figures from 5.19 to 5.22, we can see that pmSys is scoring relatively higher than Ohmage on all categories. One of the most important category and biggest gap between pmSys and Ohmage scores is the usability category. This is a very good sign, since that means that pmSys has improved the user experience by a lot on the platforms compared to what Ohmage is scored, which reflects in the data we have retrieved from the user study.

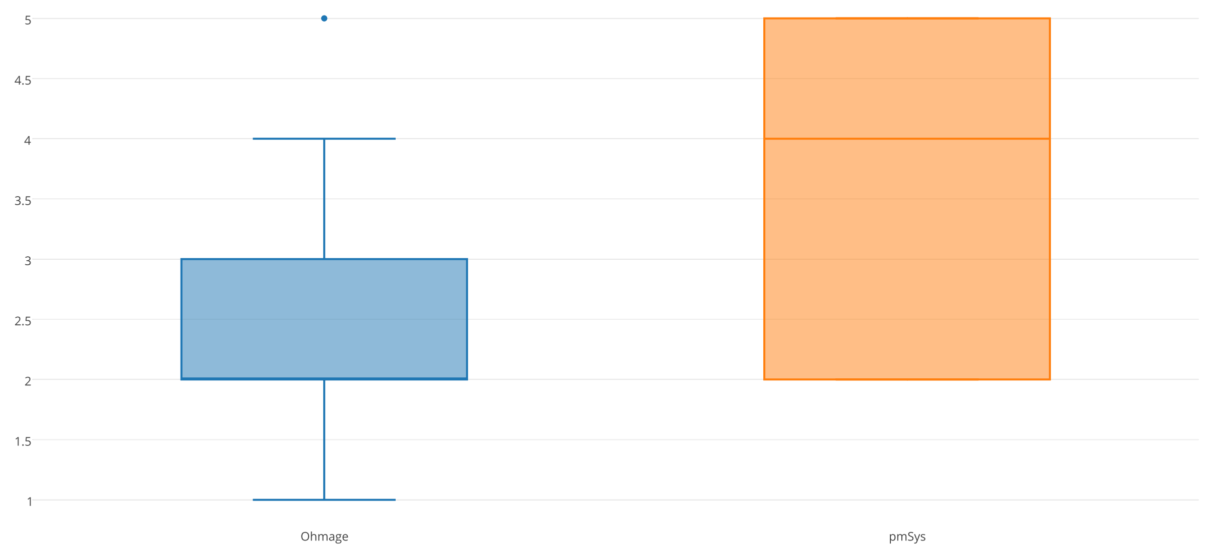


Figure 5.19: Usability

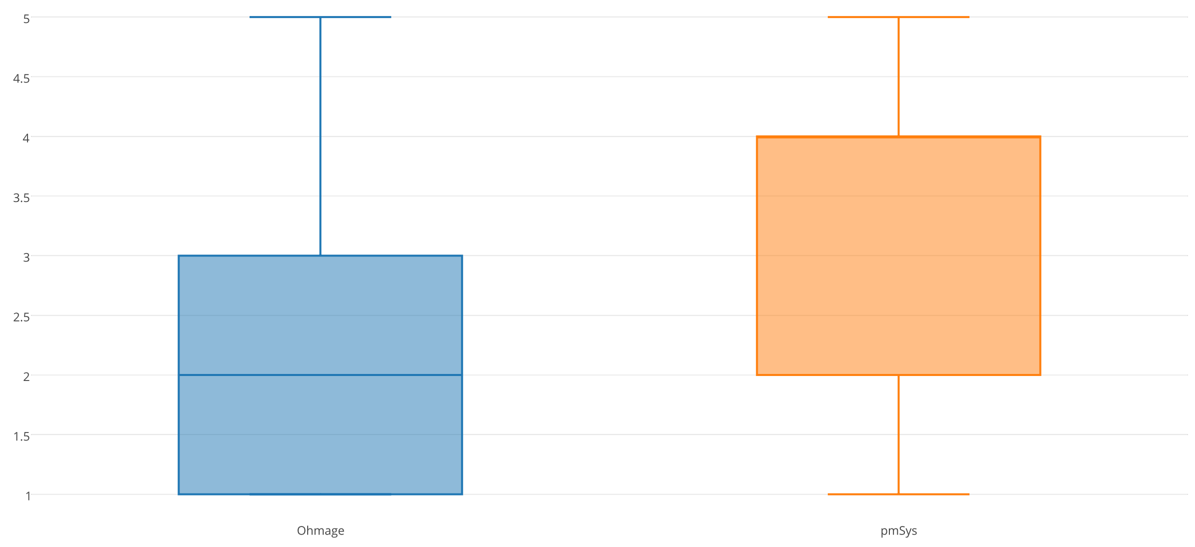


Figure 5.20: Design

5.6.3.2 User study 2: Subjective users

Even though the objective data is giving us an image of how the new users perceive the mobile applications, it is the actual users of the pmSys-app that can really tell the story. In figure 5.23, we can see that pmSys overall is scoring above half the scale (positive feedback), which proves that the pmSys-app has done something right after improving the features Ohmage had to offer, but it is also important to emphasize that there are still areas with potential to improve even further. Comments such as *"The pmSys-app works phenomenally for me, and I'm very pleased of the application's impact on my life (positive)."* and *"Very good application. Saves me a lot of time!"* gives us an indication that the application helps with saving a lot of time when answering surveys. Most of the players that answered that they previously had answered surveys by using excel or pen and paper. The fact that the pmSys-app can help them save a lot of time means that we have reached one of our goals.

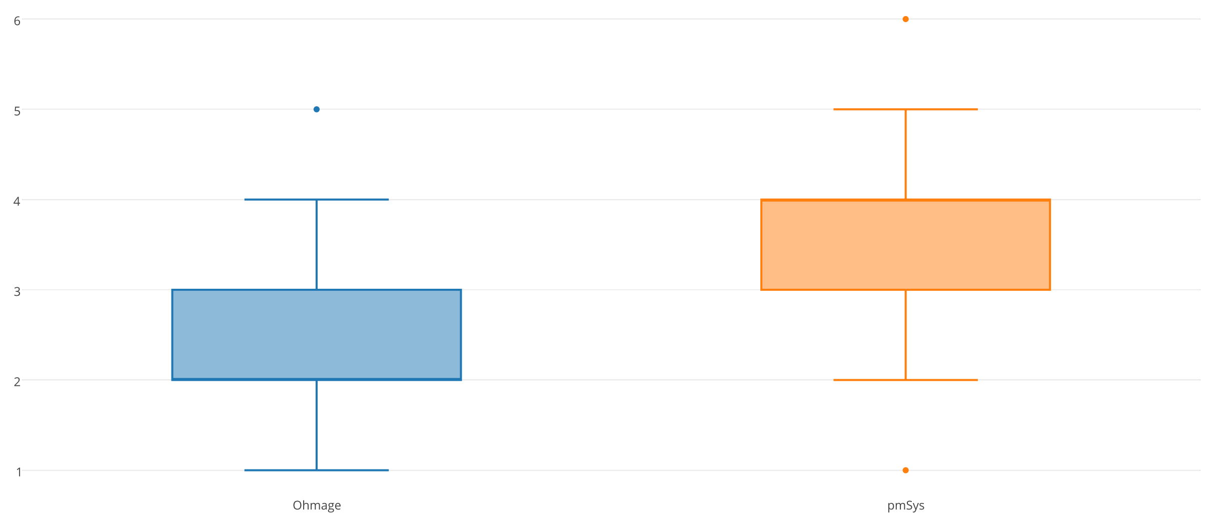


Figure 5.21: Navigation

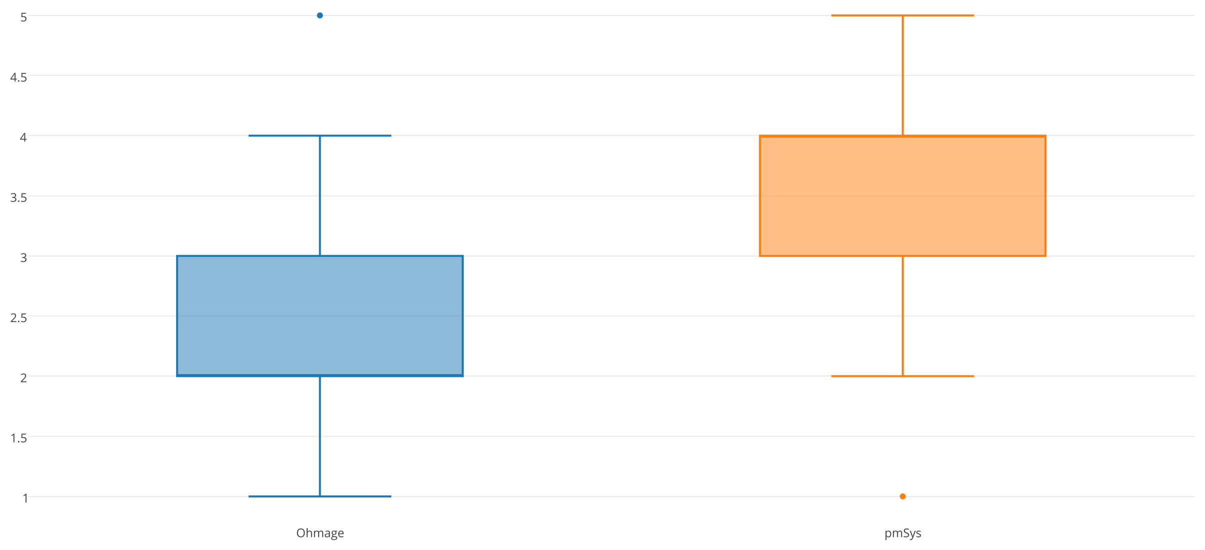


Figure 5.22: Content presentation

5.6.4 Answer rate between Ohmage versus pmSys

In 2014, we had three teams from the Norwegian Premier league that used our system with the Ohmage mobile application, which resulted in a total of 467 survey answers for **all** the players in pmSys (see table 5.5), which were 37 active players from tree different teams. If we assume that they equally answered surveys, the average survey answered the whole year would be 12,6 surveys per player. The reason for why the total amount of answered surveys is so low is not entirely known, but some of the reasons might have been raised in the bullet list in section 3.6. With a total count of 467 surveys divided on three teams is outrageous. For a system that is supposed to help the teams with detecting injuries, and to help the coaches balance their training so that players do not over train themselves and become injured for a period of time.

Year	# of players	Total amount of answered surveys	Average surveys answered per player	Mobile client used for reporting
2014	37	467	12,6	Ohmage
2015	27	2593	96	pmSys

Table 5.5: Survey statistics for 2014 and 2015

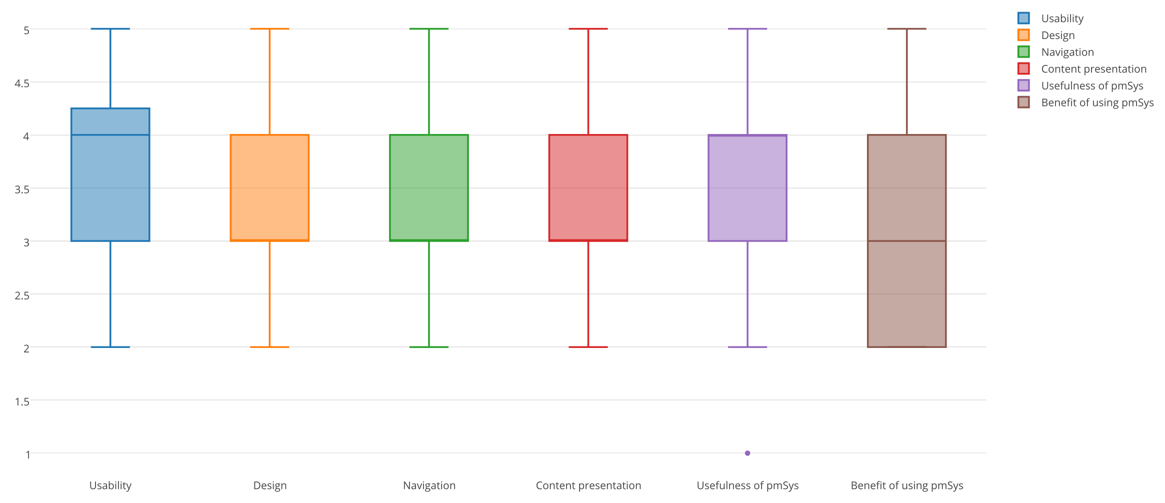


Figure 5.23: Statistics from the user study conducted at LIF and SIF

Even though the numbers from 2014 is quite low, the numbers in 2015 is incredible compared to last year (see table 5.5). There has been 2593 survey answers divided on 27 active football players from Strømsgodset in the timeframe from January 1st to April 30th. That is a **555% increase of answer rate**, and that is just data for four months for the same team! Even though the numbers is undeniable impressive, we have also seen a higher rate of participation among the players, and more resources allocated by the teams into using pmSys as a part of their training schedule.

5.6.5 Performance test of Ohmage versus pmSys

Two users on two different mobile devices performed this performance test, and because of this the data presented may only be giving an indication on where the problem lies on each platform. The first user had some differences compared to the second user, because every time the first user wished to submit the survey data, a message prompt would appear and use 1 second extra. For the second user, this only appeared once in a while; therefore the data is not entirely correct. It is also important to point out that the performance test is testing three areas:

- Normal RPE routine (from application bootup) - See figure 5.24(a)
- RPE routine (application already active) - See figure 5.24(b)
- Injury routine[1] - See figure 5.24(c)

It is also important to emphasize the difference between the two applications. Ohmage has an extra step for each question, which takes more time, however, it does not have a summary page which speeds up the process compared to the pmSys-app. The splash screen on Ohmage is relative to how long it takes to prepare the application to run, meanwhile on the pmSys-app it has a hardcoded three second delay before showing anything at all. The idea of the pmSys-app is to optimize the reporting process for the players, and this can be seen in figure 5.24 by the recorded time spent answering surveys the lower the number, the better the result.

[1] All the last options has been chosen to loop through all 11 questions in the survey

From the figures you can see that due to the self-inflicted penalty of hardcoded splash screen and summary page, the time it actually takes to answer is more or less equal to the Ohmage application. This statement is only true for small surveys such as RPE, but for longer and more advanced surveys such as injury, the time needed to finish this one takes a lot more time. The rest of the tests is ran by having the application in the foreground, which speeds up the reporting process a lot. Without the splash screen timer delay, the time needed to answer a survey is a lot faster on pmSys than Ohmage. Something we also need to consider is that these performance on time was performed right after each other, which makes muscle memory a factor in the time used to answer a survey. The more we repeated ourselves during the test, the less time it took to finish the survey. And this is what happens when a player enters the same answers every time, they start memorizing the answers, and the time it would take them to answer is minimal.

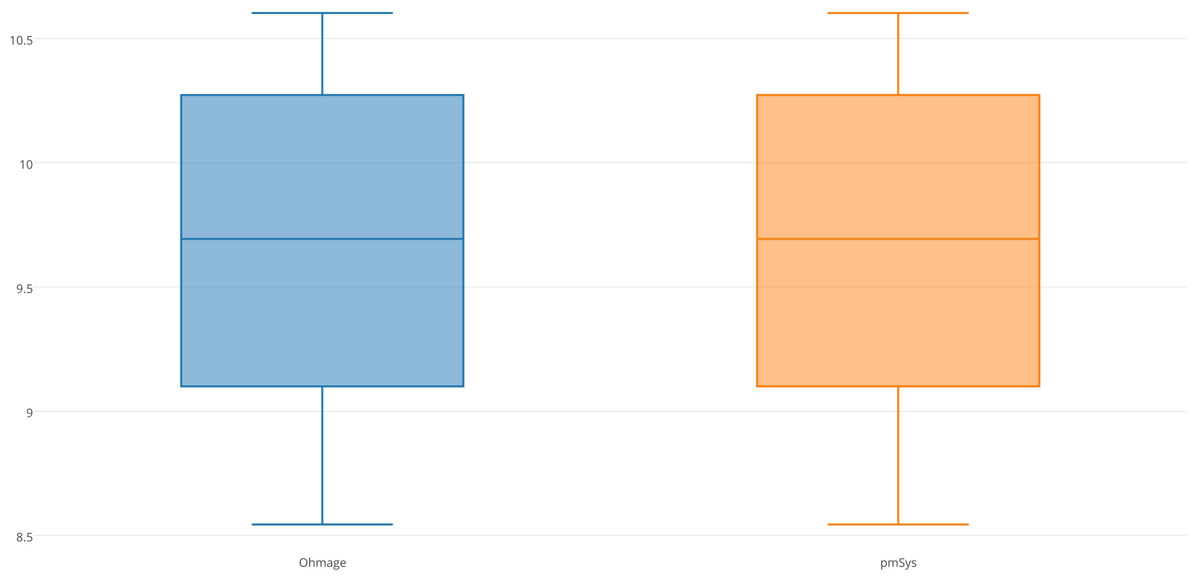
5.7 Summary

In this chapter, we have presented the data collection tool that has replaced the Ohmage MWF application, the pmSys-app. By following the two mobile development approaches, SPA and Hybrid, has made it possible for us to create a mobile application for both Android and iOS with the same source code. We have used Cordova as a native-wrapper around our code to make our web application run as if it were a native mobile application. In section 5.6.5, we evaluated the performance of the Ohmage MWF mobile application versus the pmSys-app, based on how long it would take to answer the most answered surveys. If the mobile application were opened from a cold-boot, then the time elapsed to answer a simple RPE survey the same. But if it were a repetitive process where multiple surveys were going to be answered, or if there were a long survey (like injury), then the time needed to answer with the pmSys-app were almost 20% faster than on the Ohmage MWF application.

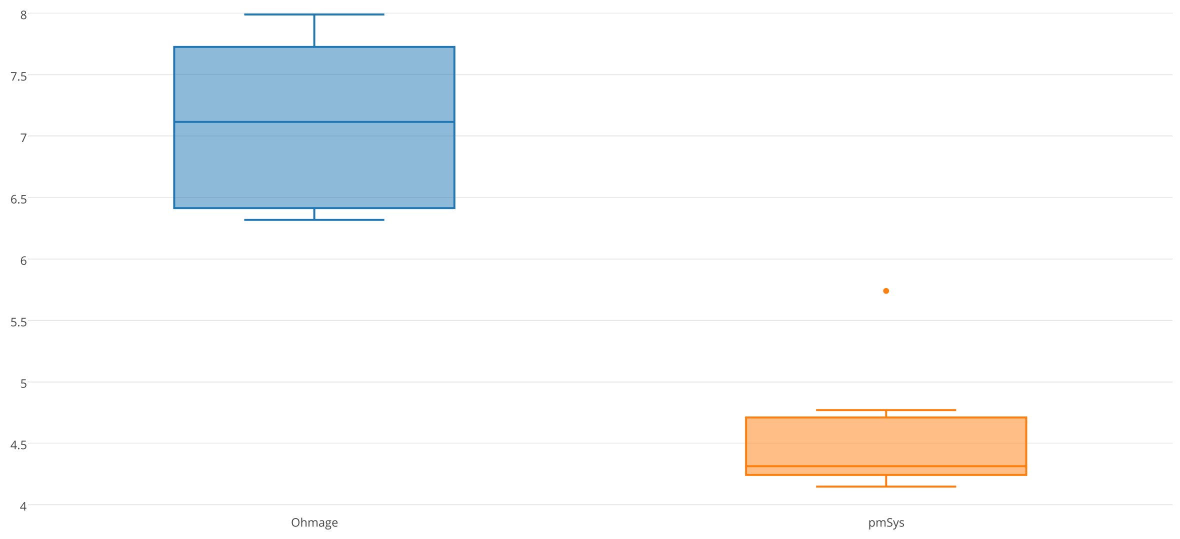
An objective user study was conducted at the University of Oslo, where the results between the Ohmage MWF application and the pmSys-app were quite different. The pmSys-app were scored higher than the Ohmage MWF application in all categories we wanted to conduct tests on. Even though the pmSys-app is scored higher than the Ohmage MWF application does not mean that it is better, the score were not on the top. Hence, there is more room for improvements in all categories before the application is "perfect" for everybody. The subjective user study conducted with two of the most active teams that currently use pmSys in their teams supports the statistics from the objective user study. The results is more alike than what we expected before conducting the user studies, but the results shows that the two groups agree with each other that there is more improvements to be done to the application.

It is safe to say that the pmSys-app is an improvement of the Ohmage MWF application and also a much better solution than pen and paper. Therefore, we will focus on our second focus group i.e., the coach and the medical staff. Now that the data collection tool has proven to be a success with a 555% increased answer rate in just four months, the coach needs a tool to gather all the data where it is presented neatly and intuitively with graphs. It is important that the size of the data samples has no effect on the effectivity of the tool; therefore, it is critical that the coach can get an overview of the teams perceived training intensity for each day within a timeframe. The coach also needs a detailed overview of a single player's data if the coach wishes to do a more thorough analysis.

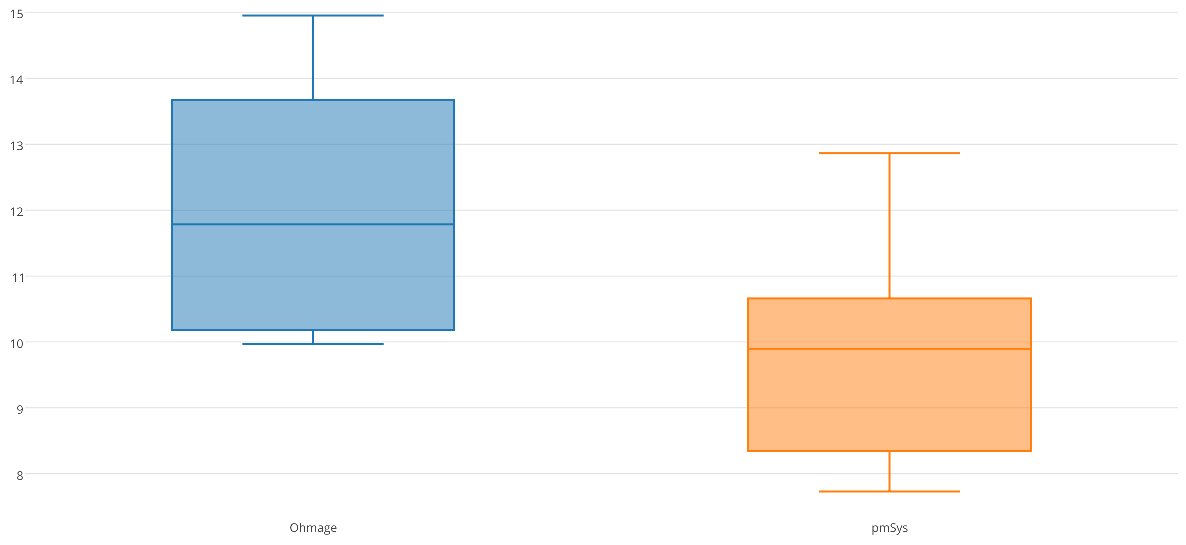
In the next chapter, we will look at the web portal we have created customized for the coach and the medical staff, pmSys-Trainer.



(a) Normal bootup + Normal RPE routine



(b) Normal RPE routine



(c) Normal injury routine

Figure 5.24: Performance tests between Ohmage versus pmSys

Chapter 6

pmSys-Trainer

In this chapter, we present the web portal, pmSys-Trainer. The web portal is the tool created specifically for the coaches and the medical staff for easier data analysis from collected survey data. In order to be useful for the coach, the data has to be presented in a way that can help the coach and the staff to detect injuries before it actually happens. Even though the main focus of the web portal is to monitor and analyze the players, we have also integrated a communication channel between the web portal and the pmSys-app. By enabling push notification messages, the coaches can remind the players about forgotten survey registrations. We will present briefly the Ohmage front-end that the first version of pmSys was deployed with. We then present the architecture, technology and frameworks used to develop this web portal. Finally, we evaluate the web portal compared to the Ohmage front-end, and we also discuss the choices done throughout the development process of pmSys-Trainer.

6.1 Motivation

pmSys-Trainer is the web portal we have created for the coaches to analyze the health data for their team, individual players and for sending push notifications to the players. The problem with Ohmage (as a system) is that the system itself is superb at collecting data, but the difficult part is to get the data out as you would like it. To analyze data without the visualization module integrated working meant that users had to extract all the raw data, format it so that his algorithms worked properly and then use the programs he needed to generate the graphs by hand. This manual labor was a strong motivation factor for creating pmSys-Trainer, there are no such thing as a coach with enough time to analyze the data themselves several times a week and still be able to do his work properly. Even though Ohmage is equipped with limited HTTP calls where the parameters could be decided we have managed to make it work seamlessly for the users, by combining the current technologies to do real-time analysis on each team and players whenever the coach wants it.

6.2 Features

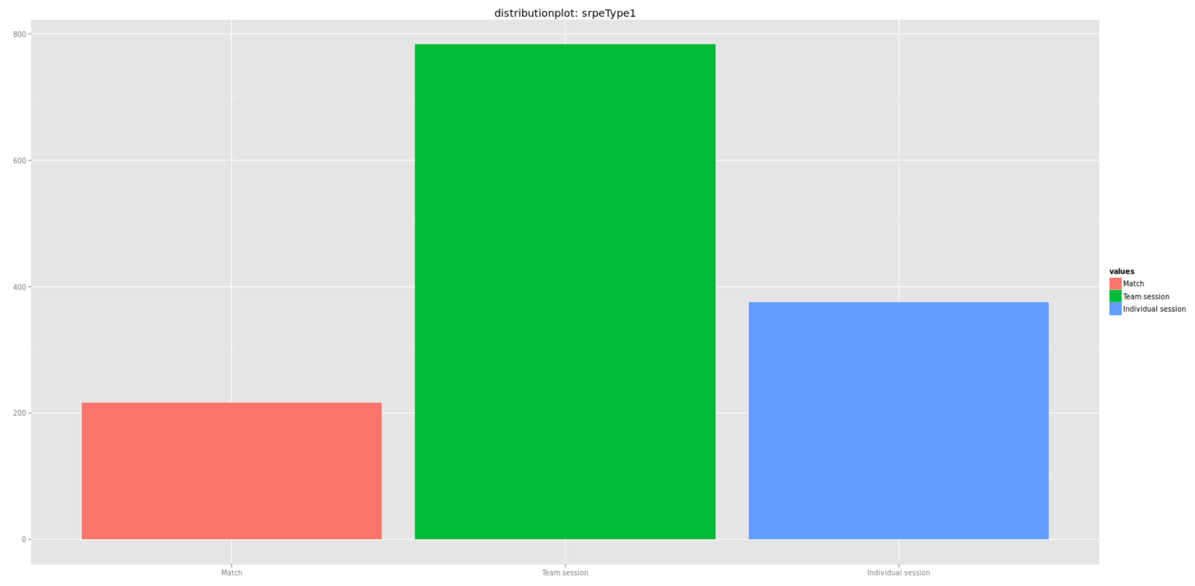
Since the focus of the web portal is data analysis and monitoring of the data, therefore, we wish to add features that enhance the user experience and also the workflow of the coaches. In the list below, the implemented features is either optimization or new features to the web portal (compared to the Ohmage front-end).

- Detailed visualization of team and player's health data

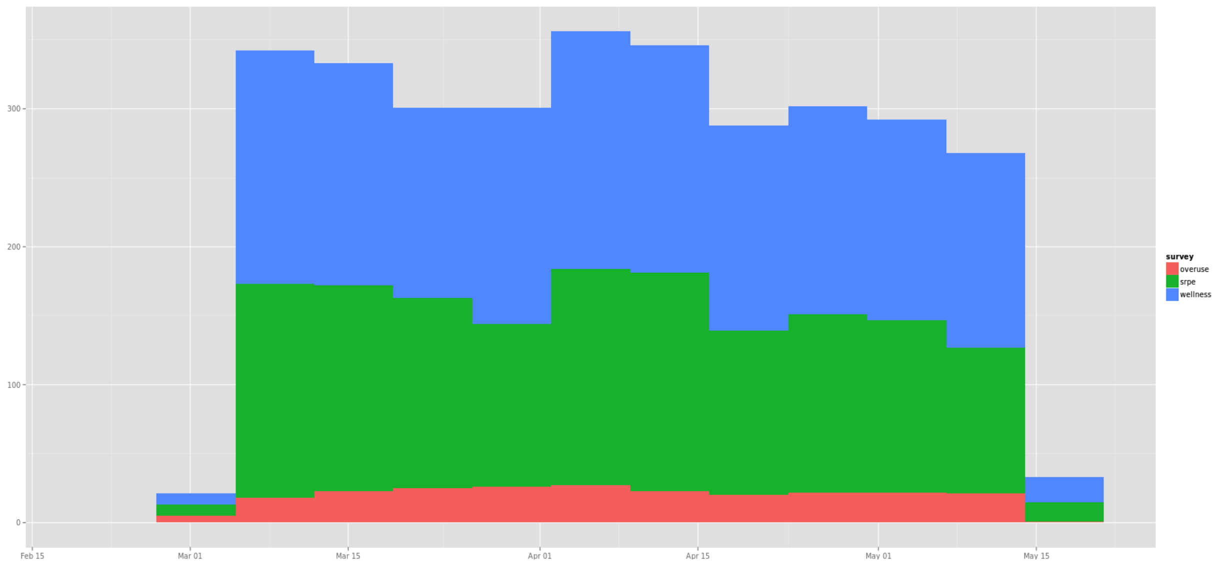
- Sending push notifications
- Create cronjobs combined with pmSys-Push

6.3 Related work: Ohmage front-end

As stated in the previous section the limitations of the current Ohmage dashboard is the sole reason why pmSys-Trainer was created. As a dashboard it has all the advanced features needed for an administrator, but for coaches it is difficult to try to understand all the features and use it effective enough. This is solely because of the visualization module that Ohmage created, which uses OpenCPU's web API [42] to visualize the data. The graphs OpenCPU returns to Ohmage has limited features, such as data interaction (see figure 6.1), because of this the data can be different from what the coaches would want and expect in order to do better analysis. An evaluation of pmSys versus Ohmage will be discussed in section 6.6.1.



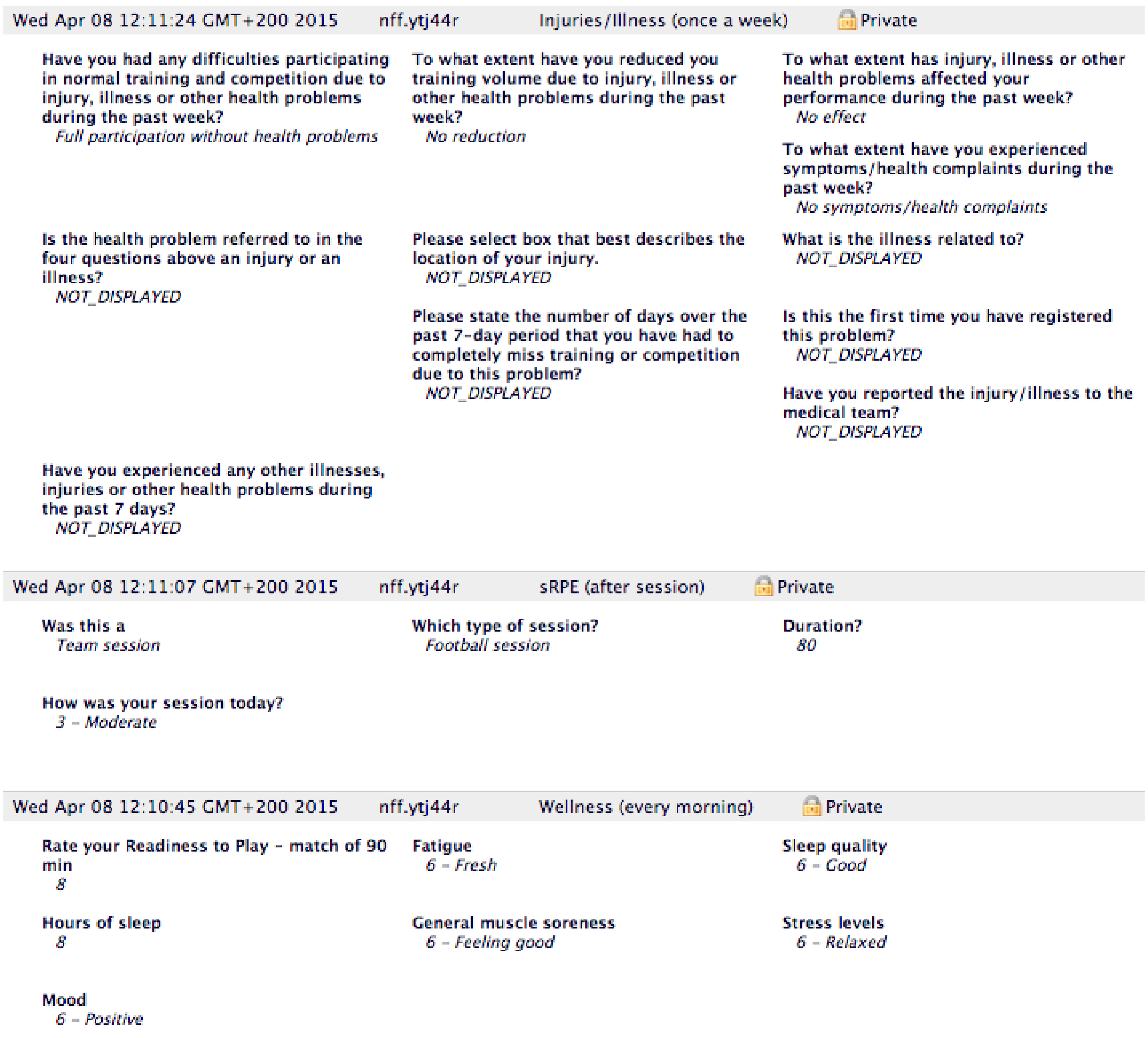
(a) Promt distribution of RPE types



(b) Survey response count of survey type

Figure 6.1: Two examples of what type of graphs OpenCPU returns to Ohmage

In order to do content analysis, the user has to export all the raw data for a specific campaign in CSV [83] format, filter the data into the correct format, and then plot it into a graph to visualize the data. The data has to be worked with in the state it is exported in, therefore, it would require a lot of manual labor in order to make the data useful. When the manual analysis is done, the visualization of the data is exported in PDF-format and then attached to a email. This approach does not allow the user to interact with the data, hence also not providing the best nor the fastest type of analysis.



Wed Apr 08 12:11:24 GMT+200 2015 nff.ytj44r Injuries/Illness (once a week) Private

Have you had any difficulties participating in normal training and competition due to injury, illness or other health problems during the past week?
Full participation without health problems

To what extent have you reduced you training volume due to injury, illness or other health problems during the past week?
No reduction

To what extent has injury, illness or other health problems affected your performance during the past week?
No effect

To what extent have you experienced symptoms/health complaints during the past week?
No symptoms/health complaints

Is the health problem referred to in the four questions above an injury or an illness?
NOT_DISPLAYED

Please select box that best describes the location of your injury.
NOT_DISPLAYED

Please state the number of days over the past 7-day period that you have had to completely miss training or competition due to this problem?
NOT_DISPLAYED

What is the illness related to?
NOT_DISPLAYED

Is this the first time you have registered this problem?
NOT_DISPLAYED

Have you reported the injury/illness to the medical team?
NOT_DISPLAYED

Have you experienced any other illnesses, injuries or other health problems during the past 7 days?
NOT_DISPLAYED

Wed Apr 08 12:11:07 GMT+200 2015 nff.ytj44r sRPE (after session) Private

Was this a
Team session

Which type of session?
Football session

Duration?
80

How was your session today?
3 - Moderate

Wed Apr 08 12:10:45 GMT+200 2015 nff.ytj44r Wellness (every morning) Private

Rate your Readiness to Play - match of 90 min
8

Fatigue
6 - Fresh

Sleep quality
6 - Good

Hours of sleep
8

General muscle soreness
6 - Feeling good

Stress levels
6 - Relaxed

Mood
6 - Positive

Figure 6.2: Ohmage dashboard responses

It is important to note, that the data from OpenCPU is only **statistical analysis**. The data we get in return is simple a counter of survey type and total survey registered within a period. For a coach, the statistical analysis helps nothing in detecting pre-injuries or over trained players. The visualization module also only allow single player analysis of the same type (statistical analysis), therefore, it is difficult to use the Ohmage front-end to do effective analysis and monitoring of a players physical state. The Ohmage front-end do not offer a team visualization, but only a single players response data (see figure 6.2). The data cannot be visualized in a way that makes analysis easier, but it is simply retrieved and presented as questions and answers. This is also the only form for "content analysis" the Ohmage front-end can offer the user by using their visualization module.

6.4 Architecture

The pmSys-Trainer architecture consists of NodeJS [34], Express [84] and Nunjucks [85], where NodeJS is the framework we use for both our client and server side. To keep the system as simple and easy to maintain, we have selected the best and most suitable front-end framework, Nunjucks with a syntax that is close to what AngularJS offers.

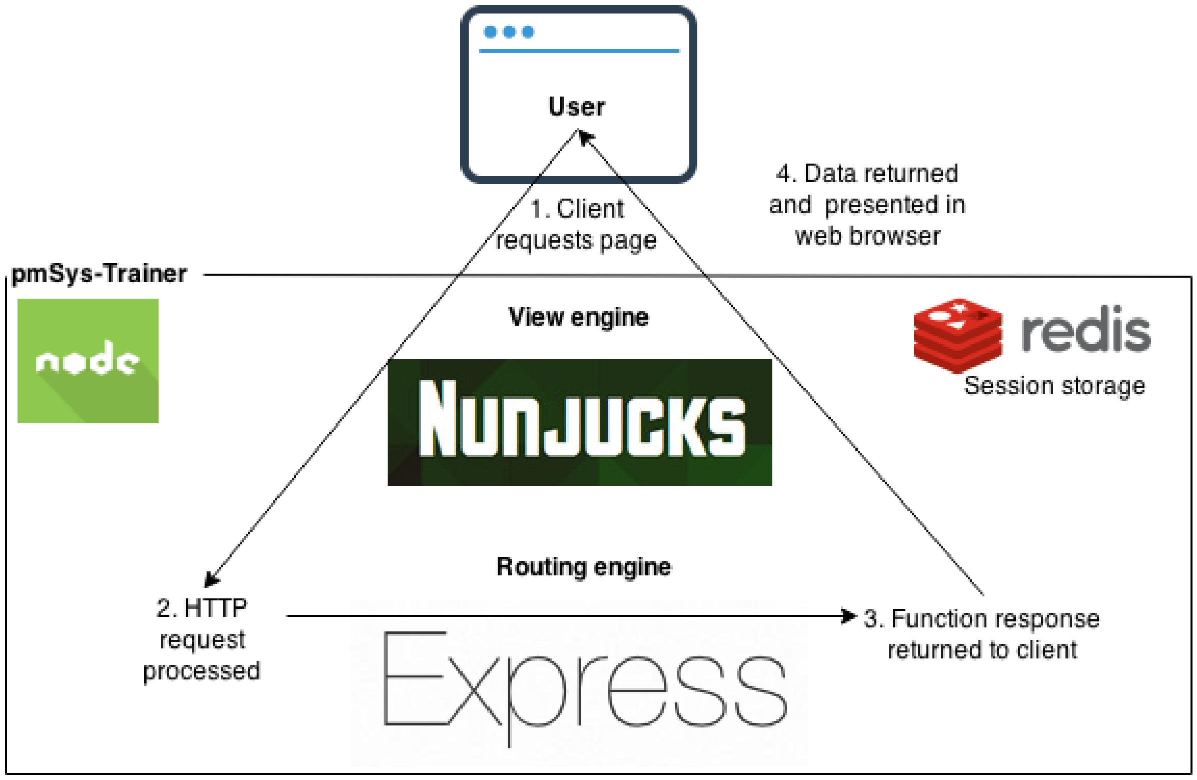

User
1. Client requests page
4. Data returned and presented in web browser
pmSys-Trainer
node
View engine
NUNJUCKS
redis
Session storage
Routing engine
2. HTTP request processed
3. Function response returned to client
Express

Figure 6.3: pmSys-Trainer architecture

6.4.1 NodeJS with Express and Nunjucks

In section 3.1, we wanted to try out new technology when developing pmSys. The idea was to use a lightweight framework for creating a server side system. NodeJS [34]. Traditional solutions would require us to create a server with features, and then create a web application designed to retrieve data from the server.

Express is a web framework with feature support for server side, such as routing and session cookies. Express adds rich functionality to the NodeJS server, by simplifying the complex routines for data transferring between client and server sides. For example, when uploading a file to a NodeJS server, the file has to be encoded before it gets uploaded. Express will automatically recreate the file on the server side when the upload is finished, hence saving a lot of time and programming logic. We use Express as our internal routing table to serve or retrieve webpages for the client machines, this way we can focus purely on creating functions in JavaScript to fulfill a HTTP request instead of network protocols or error handling (see figure 6.4).

In figure 6.4, the code says *response.render(. . .)*, which means it will use the defined view engine. A view engine is the front-end engine installed to render webpages, and in pmSys-Trainer, we use Nunjucks [85]. What Nunjucks offers is the possibility to inherit or extend webpage layouts and the syntax is close to

```
router.get('/upload', function(request, response) {
  if(request.header['secret'] == null) {
    response.render('error.html'); // Uses the view engine
    response.end(404, "Secret not found!"); // Returns HTTP request with
        error code and content in plain text
  }
  else {
    response.render('upload.html');
  }
});
```

Figure 6.4: Sample code of Express routing with error handling

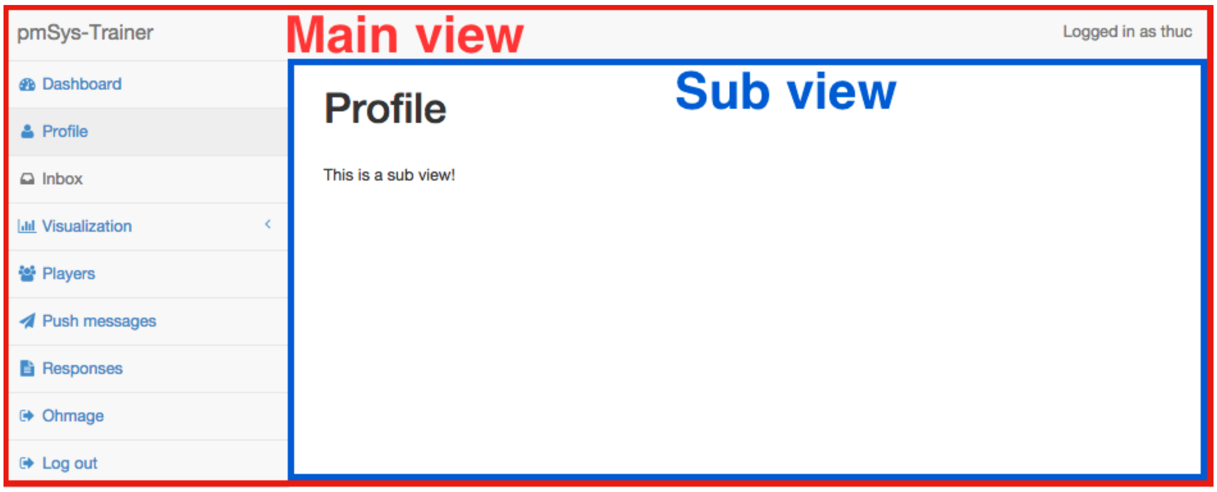
```
// MainLayout.html
<html>
  <body>
    {% block head %}
    {% endblock %}
  </body>
</html>

// Profile.html
{% extends "MainLayout.html" %}

{% block body %}
  <h1>Profile</h1><br />
  This is a sub view!
{% endblock %}
```

Figure 6.5: Sample code for the views in Nunjucks

HTML (see figure 6.5), and with this we can create unlimited webpages with the same main view, but with different content (see figure 6.6).



Profile

This is a sub view!

Figure 6.6: Extending the main layout in Nunjucks

6.4.2 Redis

Redis is an open source key-value database which can contain strings, hashes, lists, sets, sorted sets, bitmaps and hyperloglogs [86]. As a default Redis support 10.000 concurrent clients (configurable) at the same time, which makes it perfect for a system like pmSys. Companies such as Twitter, Snapchat, Stack Overflow and Github are using Redis as a key-value session storage. This has been confirmed by Github that their response time on their Github webpages went from 500ms to below 100ms in average after changing their architecture in 2009 [87], Redis was used as a key-value routing server to speed up searching for data on their data disks. What separates Redis from other NoSQL-databases is that Redis is a single threaded data store, which stores data in-memory but it also provides presistence of data the system were to go down (if configured), by creating snapshots stored locally on the host machine and it is a key-value storage.

6.4.3 Bootstrap

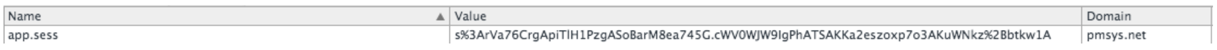
The pmSys-Trainer web portal is responsive, which means that the web application will be automatically resized if opened in a web browser on a mobile device (see figure). Bootstrap [88] is an open-source front-end framework, and it is created by Twitter. It has become one of the most popular and "go to" front-end framework for web developers world wide, and it is very simple to use since the project has an amazing documentation and support. The specific theme layout that we have used on the web portal is SB-Admin [89], and it has all the UI-components that we need for a web portal.

6.5 Implementation

6.5.1 Session key storage

When a user have successfully logged into pmSys-Trainer, a hashed password will be stored in the Redis server as a session storage, which makes it possible for coaches to use the web portal without signing in each time. The hashed password will not change before the coach changes his password, which would force the coach to re-enter his credentials to the web portal to be able to use it, but as long as the password is not changed the coach could log on from the same computer.

Once logged in the user stores a cookie with a session key which is used each time the page loads with the Redis server to validate the user's credentials.



Name ▲	Value	Domain
app.sess	s%3ArVa76CrgApiTlH1PzgASoBarM8ea745G.cWV0WJW9IgPhATSAKKa2eszoxp7o3AKuWNkz%2Bbtkw1A	pmsys.net

Figure 6.7: Session cookie on pmSys-Trainer

6.5.2 Visualization

The visualization module is a tool created to provide the coach and the medical staff with visualization of response data from the players. The visualization module supports two types of graphs, team and single player visualization. The process of how the data is processed and visualized is like how it was done in section 5.4.6, all the request results has to be concatenated before we use C3.js [90] compared to NVD3 as we did in the pmSys-app. The libraries are almost the same, but C3 offers more features within data interaction and multiple graphs on top of each other. The data format C3 requires is a bit different fron

NVD3, by forcing us to use two arrays in order to create one for the dates and one for the response values (see figure 6.8). Furthermore, it is important to note, that it is not needed to create a date object for each response, the C3 library will automatically create when generating the SVG canvas with the response data.

```
var date = [date_{1}...date_{n}]
var response = [response_{1}...response_{n}]

bind = {
  // y- : x-axis
  response : date
}
```

Figure 6.8: Example of the required data format for C3

All the data is fetched by multiple API calls to the back-end, and then concatenated together on the client side. The web browser has limited resources when it comes down to heavy data processing, therefore, the coach and the medical staff will not notice the latency when they wish to visualize the response data at the scale of a single month. But the data from a whole year could slow down the visualization, due to the fact that the data is processed on the client side and not on the server side before visualization (see section 6.6.2).

6.5.2.1 Team visualization

When a coach looks at the team visualization, he will see graphs showing the answers of the team that has registered data over time. All data from RPE, Wellness and Injury will be plotted here for the coach to analyze. The graphs are interactive where it is possible to select which players the coach wants to compare their data. By presenting the data this way, it is possible to get an overview of who has reported higher than the rest of the team for a day. Instead of checking out one and one player at the time, it is possible to just filter out visually which player that is near the warning zones (yellow/red) which shows how hard the sessions is each day. All the graphs has a scrollable bar (see figure 6.9) below the visualization, where the zoom can be adjusted in order to see data more clearly if the responses is too dense in an area. In figure 6.10, we have decided to make the visualization easier to read off by creating categories. The categories will show the survey responses from each player, which makes it a lot easier to focus on only one category at a time. One of the arguments on why the Ohmage front-end's visualization is not suitable for what the coach needs, is the possibility of interaction with the graphs. In the pmSys web portal (see figure 6.11), the graphs offers interaction to find out who is injuried based on a score, where the higher score value there is, the more injured is the player.

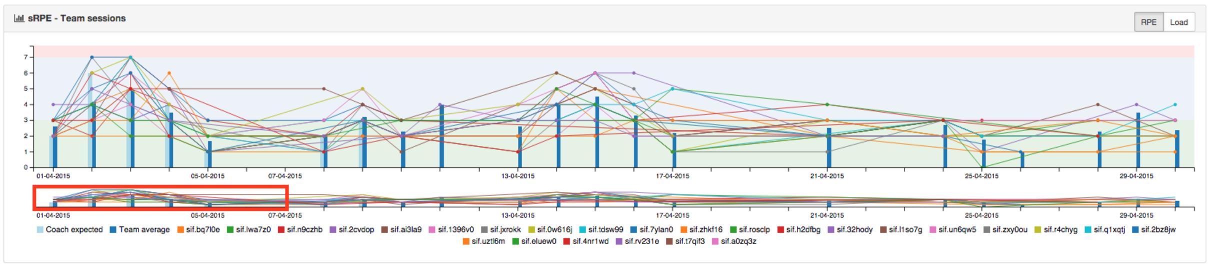


Figure 6.9: RPE visualization

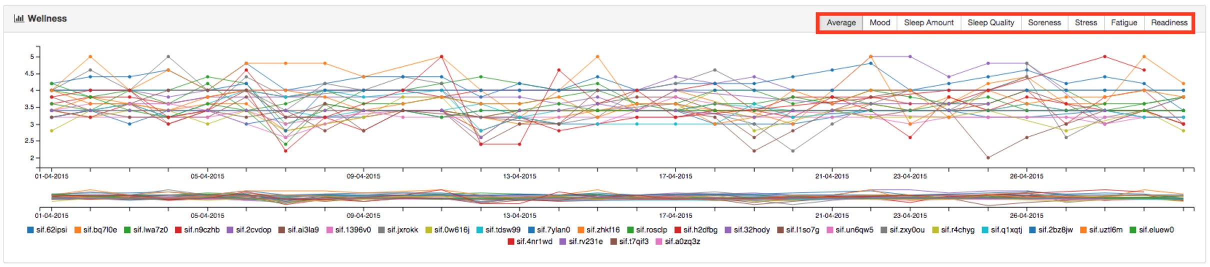
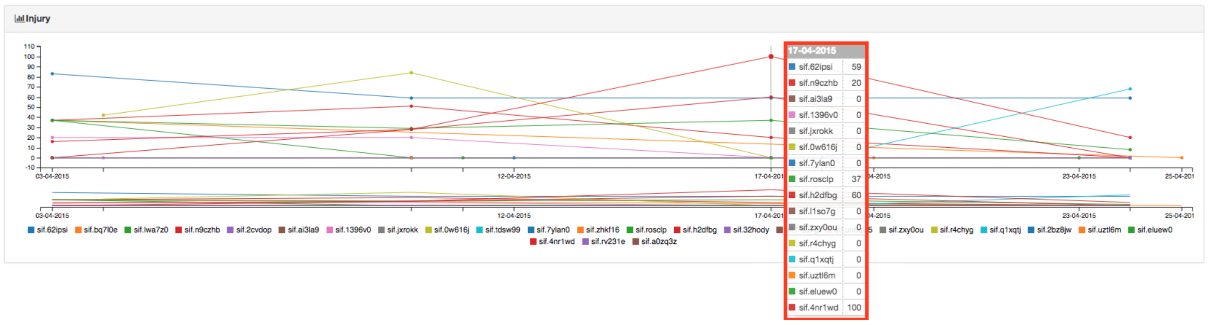


Figure 6.10: Wellness visualization



17-04-2015	
🟦 sif.62ipsi	59
🟥 sif.n9czhb	20
🟫 sif.ai3la9	0
🟪 sif.1396v0	0
⬛ sif.jxrokk	0
🟨 sif.0w616j	0
🟦 sif.7ylan0	0
🟩 sif.rosclp	37
🟥 sif.h2dfbg	60
🟫 sif.l1so7g	0
⬛ sif.zxy0ou	0
🟨 sif.r4chyg	0
🟦 sif.q1xqtj	0
🟧 sif.uztl6m	0
🟩 sif.eluew0	0
🟥 sif.4nr1wd	100

Figure 6.11: Injury visualization

6.5.2.2 Player visualization

On the contrary of the team visualization which may have too much information to be useful in the long run, a single player visualization gives the coach a more cleaner and detailed visualization of the data the player has reported (see figure 6.12). The single player visualization does not support the zoom feature like in the team visualization. The reason for this is that the data is not as dense as it is on the team visualization, and the visualization can give the coach a more detailed analysis of a player if the player has abnormal values over time. The data that is shown here is subjective data, more about objective data will be discussed further in chapter 8.

6.5.3 Push Notification

The push notification UI can do more than just sending push messages, it also gives an overview of who has answered RPE and wellness. All users that is listed in this feature is users with either iOS or Android device token registered, this ensures that the users that has unsubscribed themselves from the push service at logout will not receive push messages over time. See chapter 7 to see more about the push system.

It is also possible to create automated push messages for when to send push messages to the users (see figure 6.13), as of now it will send to all users on the list above. It is not possible to select the users you want to send push messages to, but more on this topic can be found in chapter 8.6. The automated push messages saves time for the coaches, since they no longer have to log into pmSys-Trainer in order to send out push messages. Before this feature was created the coaches had to send push messages manually, which meant that when the push messages was sent to the players, the time the players received the push notification was not consistent. One day the notification could be sent 9AM, another day were sent 11AM, by creating this feature the time the messages is sent can be predicted and expected.

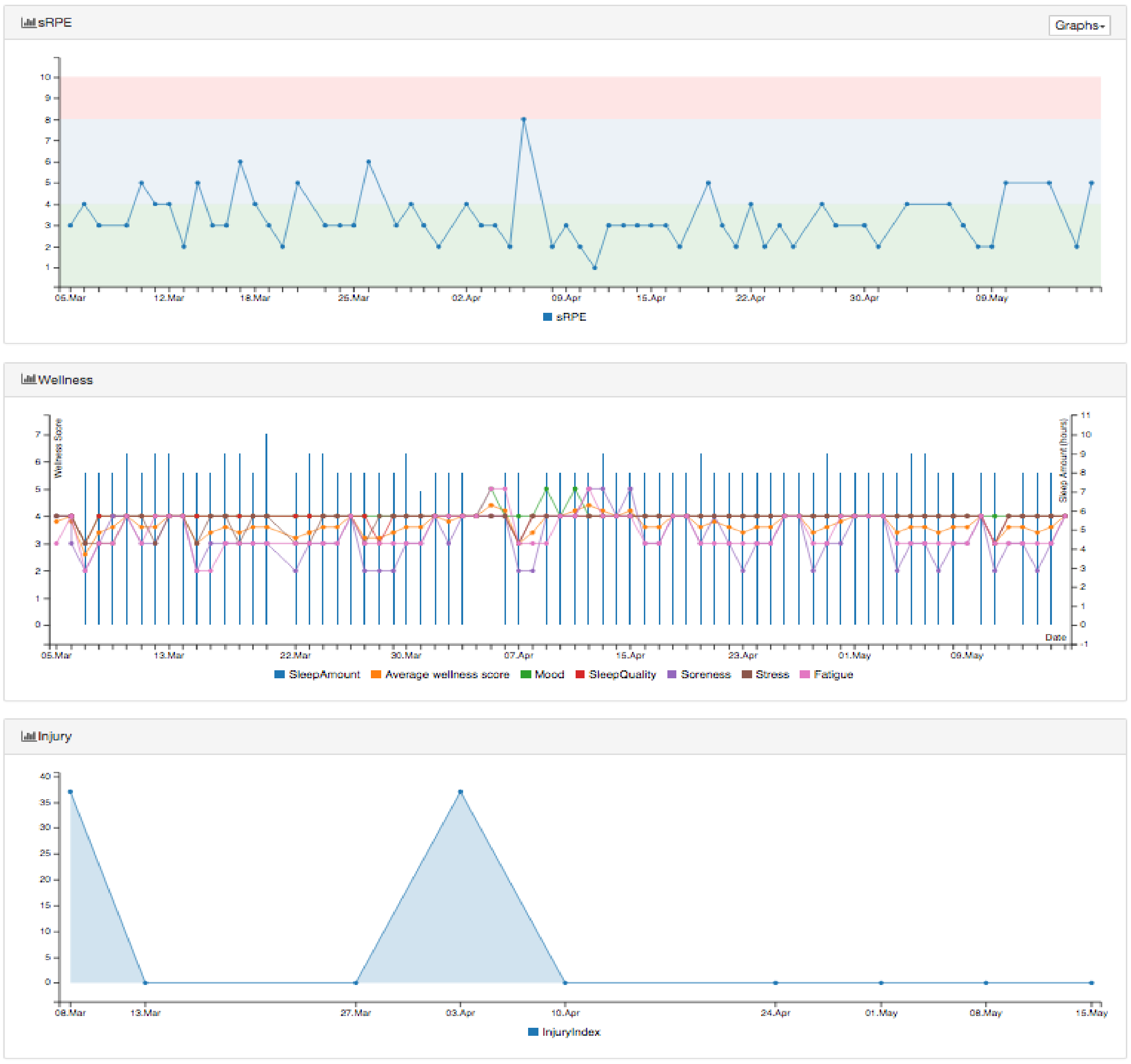
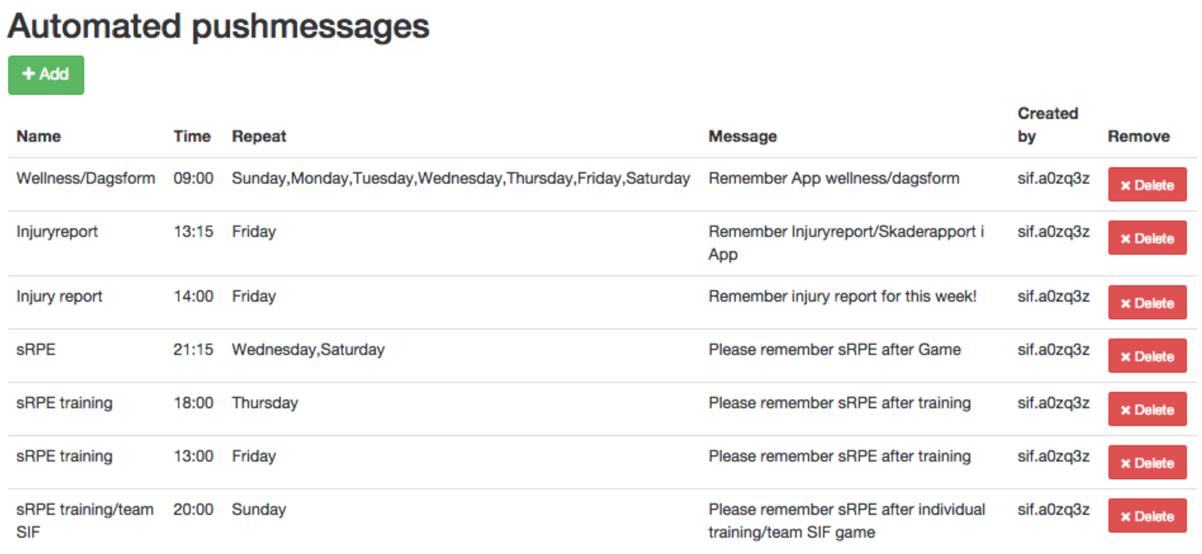


Figure 6.12: Player visualization

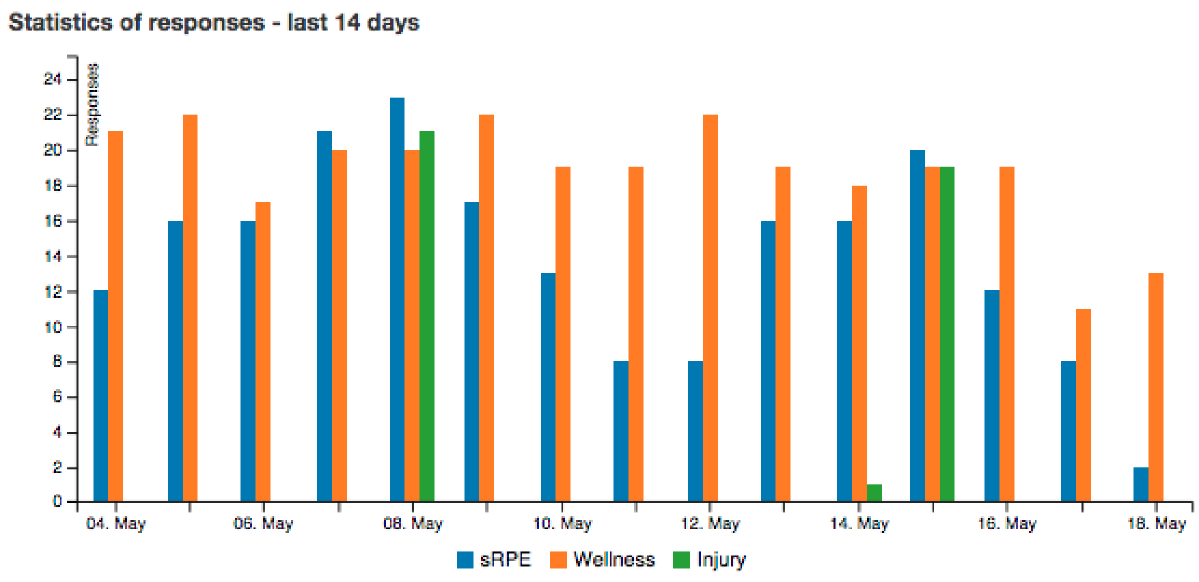

Automated pushmessages

Name	Time	Repeat	Message	Created by	Remove
Wellness/Dagsform	09:00	Sunday,Monday,Tuesday,Wednesday,Thursday,Friday,Saturday	Remember App wellness/dagsform	sif.a0zq3z	× Delete
Injuryreport	13:15	Friday	Remember Injuryreport/Skaderapport i App	sif.a0zq3z	× Delete
Injury report	14:00	Friday	Remember injury report for this week!	sif.a0zq3z	× Delete
sRPE	21:15	Wednesday,Saturday	Please remember sRPE after Game	sif.a0zq3z	× Delete
sRPE training	18:00	Thursday	Please remember sRPE after training	sif.a0zq3z	× Delete
sRPE training	13:00	Friday	Please remember sRPE after training	sif.a0zq3z	× Delete
sRPE training/team SIF	20:00	Sunday	Please remember sRPE after individual training/team SIF game	sif.a0zq3z	× Delete

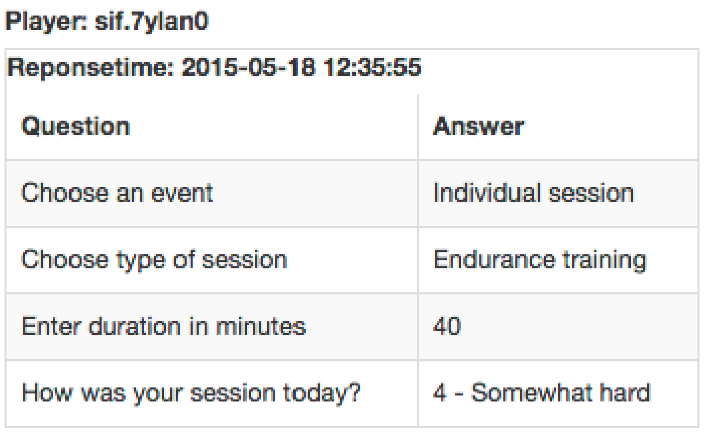
Figure 6.13: Automated push messages

6.5.4 Survey responses

This module allows the coach to read the survey response from each player. The coach and the medical staff is the only users defined as privileged (see table 3.2), therefore, they can access all responses in the campaigns they participates in. As a default, a statistical graph is presented to show which survey and how many has answered in the last 14 days (see figure 6.14(a)). The coach has to select a date in order to retrieve a list of who has answered surveys of a specific date. The list is divided into two groups; responded and not responded. The name of those who has responded is hyperlinked, and by clicking on a player's name an overview of the responses is shown (see figure 6.14(b)).



(a)



Player: sif.7ylan0

Reponsetime: 2015-05-18 12:35:55

Question	Answer
Choose an event	Individual session
Choose type of session	Endurance training
Enter duration in minutes	40
How was your session today?	4 - Somewhat hard

(b)

Figure 6.14: The response-module in pmSys-Trainer

6.6 Evaluation & Discussion

In this section, we will evaluate some key choices taken when creating the web portal the coaches' uses. As a tool it is important that the tool is making the analysis more effective, and not more time consuming.

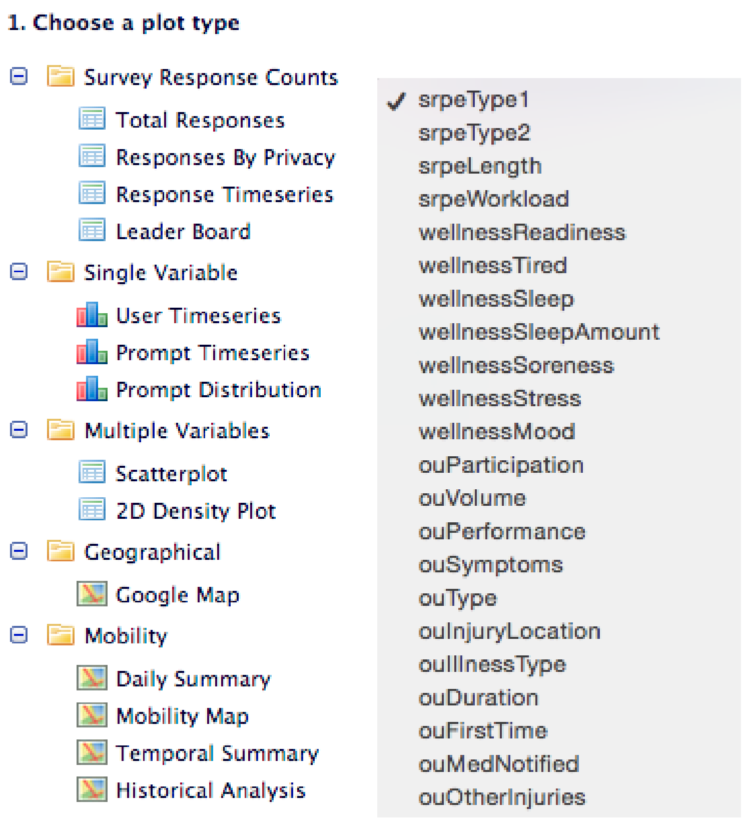
6.6.1 User study: pmSys front-end vs Ohmage front-end

In chapter 3, we evaluated Ohmage and all the systems under the name of Ohmage. In this section we will evaluate Ohmage's front-end versus pmSys front-end, since its features is vastly different from each other. The front-end Ohmage currently uses is meant for both the coach and system administrators, what it lacks is the features the coaches needs, in this case it is visualization that works correctly when deployed. Instead of replacing everything Ohmage has to offer, we have replaced the mobile application and the front-end (pmSys-Trainer) to suit our users better.

Systems	User manipulation	Class manipulation	Campaign manipulation	Survey manipulation	Visualization	Push notifications
Ohmage	Yes	Yes	Yes	Yes	Yes	No
pmSys	No	No	No	No	Yes	Yes

Table 6.1: Ohmage versus pmSys features

The reality is that where pmSys lacks features, Ohmage fulfills the missing functionalities (see table 6.1). This is especially true for the administration features, since it is not possible to add users or remove users from campaigns with pmSys. It is also not possible to create new campaigns, this is why the Ohmage front-end is just as important for system administrators as well at the pmSys front-end is for the coaches. In a way, both front-ends cannot be compared in the normal way where you analyze the features against each other. Since Ohmage has created a web portal for both the coaches and system administrators, whereas pmSys has solely focused on the coaches with analytic capabilities, which makes the requirements for each system vastly different.



(a) Visualization settings (b) Prompt types

Figure 6.15: Ohmage's visualization module

If we were to evaluate features between Ohmage and pmSys, only the visualization module would be comparable in this case. In comparison between Ohmage and pmSys visualization of data, pmSys has a vastly superior visualization of the data, by giving the users the ability to interact with the data. Another important aspect of the visualization between pmSys and Ohmage is that Ohmage only offers **statistical**

analysis of the survey answers, meanwhile pmSys offers **content analysis**. It also requires less steps to show different types of data parameters, which again can be translated into userability and motivational factor to use pmSys if you are a coach. Another difference between the visualization of data is the team visualization, meanwhile pmSys provides this feature, Ohmage does not have this feature at all. It is not possible to do this without doing the same queries for all the players and then open all the images it returns in order to see what happens. Even then it would make it impossible to compare the players against each other, this is why it is safe to say that pmSys is the best web portal of the two when it comes to team analysis, whereas Ohmage is the best front-end for system administrations.

6.6.2 Client side versus server side processing

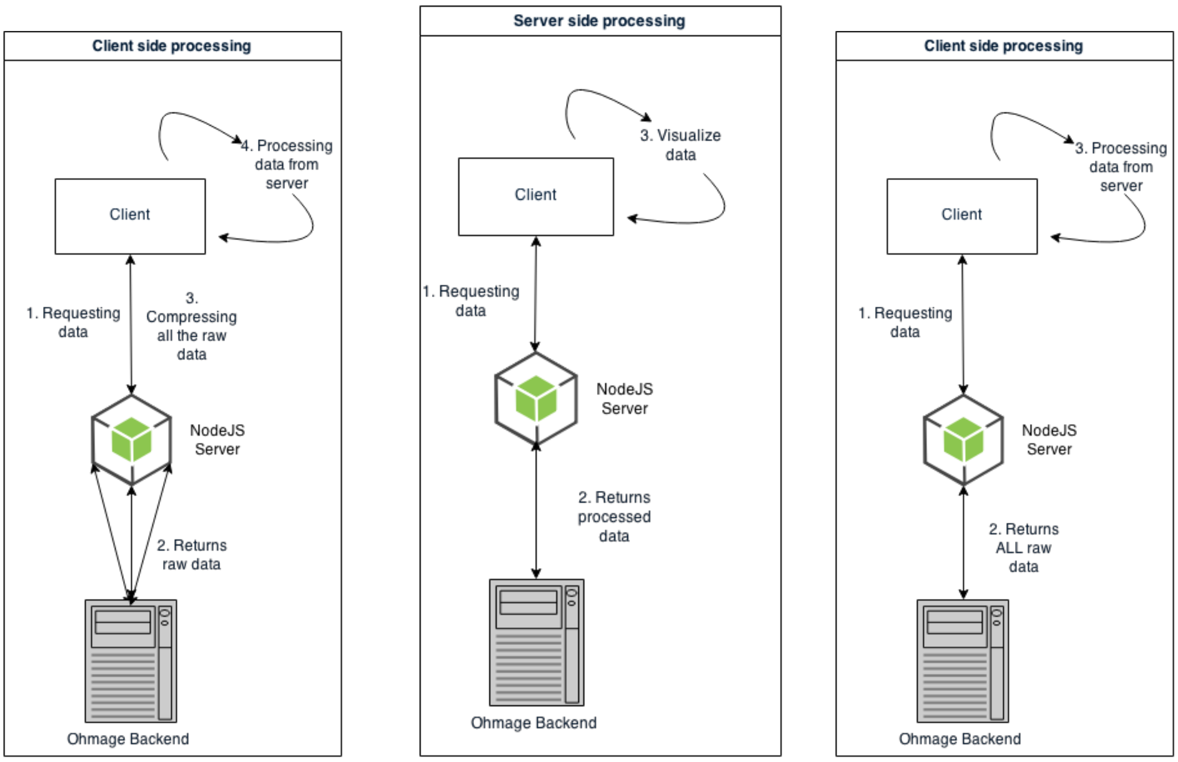
pmSys has systems that has client side and server side processing of data, the difference between them is small, however, the outcome of performance may be quite large over time. To understand what we mean by client side and server side processing, we need to define them. *Client side* is the side where the UI is viewed, the interaction on a web page, the machine the user is using to go into web page X. In the latest years it has been popular with what you can do with the client machines, now that the machines are more powerful and cheaper than before, the means needed to process large amount of data is getting reduced. In our case, we use JavaScript to process the visualization data on the client side, because of this reason.

Now that the machines are getting faster and more efficient, there is a lot to gain by moving the data processing from the server side (back-end) to client side. We thought a lot of data had to exist before the visualization would slow down the whole process, but in reality, it was only after 2 months of real data when the visualization started to slow down. The problem lies with the library we use to create the graphs, since it has trouble when there is more than 500 data points on a graph [91]. The problem also lies in the way the Ohmage back-end serves the data. Instead of one API call to get all data, we need to do multiple calls, then match the results from all the calls into 1 JavaScript object before creating the graph (see figure 6.16(a)). This increases the latency for the user even though it is barely noticeable since client machines are pretty powerful these days. However, with large sets of data, it becomes a lot of load on the web browser to process all the data.

There are several ways the processing process can be fixed, the first way is to process everything on the server side (see figure 6.16(b)), and then send it to the client side for simply visualize the data. This would be the easiest solution for big data sets, but there is also another way (see figure 6.16(c)) that requires some more work because then the Ohmage back-end has to be modified, to pre-fetch all the data together (unprocessed) and then returned. The difference between figure B and figure C is that one of them returns the data pre-processed processed which means the client can simply visualize the data, and the other way simply gathers all the data the client needs unprocessed, and instead of X calls the data is returned in one and then it is processed on the client side.

6.6.3 Token storage

In pmSys-Trainer, we chose to use Redis as the session storage, and some of the reasons has been discussed in section 6.4.2. The reasons listed were due to the speed and the high amount of supported concurrent connections in Redis, but another reason is that pmSys-Trainer can potentially use a lot of memory on sessions. Instead of leaving the sessions in memory without a timeout limit (current configured on pmSys-Trainer), it is possible to let Redis store this in a file to save memory. This way we can reduce the memory usage instead of using up all the memory needed on session tokens. The idea of using this technology is to force the coach to login once onto the web portal, after that the coach should be automatically logged in on to one specific computer. If the sessions were set to be invalid after a couple of minutes or hours the coaches had to use time to log into the portal just to use it.



(a) Client side (b) Server side (c) Client side v2

Figure 6.16: Client side vs Server side with pmSys-Trainer

6.7 Summary

The goal of the web portal was to create a better data analysis tool, which could assist the coach and the medical staff with analyzing and monitoring their players. An issue that appeared after a couple of months is the way we processed the data we got from the Ohmage back-end. By processing it on the client side, we reduced the workload on our own server, however, side effects of the performance on the client side were much higher than what we anticipated in the early phase of the development process. The pmSys-web portal is optimized for data analysis and with the coaches in mind, meanwhile the Ohmage front-end is a general platform for both the coaches and the system administrator. There is no secret that the web portal is a big improvement of the old analysis methods, such as pen and paper and the export process in Ohmage. But the web portal lacks supports for system administrators, therefore, it is impossible to use pmSys without the pmSys-web portal and the Ohmage front-end. There is also a lot of improvement potential for the web portal, and we have a long list of features that is requested, and features we want to improve.

We asked two medical analysts that have used the pmSys web portal and analyzed the Tippeliga teams since the beginning of this project. When the first version of pmSys only offered the Ohmage front-end, they had to export, fix the data for gaps due to missing survey registrations, plot it into a graph library, and then be able analyze it. The process was tiresome and it was easy to calculate the values for the graph wrong. Analyzing the survey responses for a whole team once a month, could take days to complete. And to avoid injuries, the analysis had to calculate and finish a lot faster than what Ohmage and manual labor could offer, hence, the motivation for the pmSys web portal. Now that the process has been optimized, and the web portal does all this automatically, they no longer need to this themselves. The visualization graphs will be refreshed instantly right after the player has registered their data, which is one of the most crucial and important feature the web portal can offer a coach. When we asked them how they felt about using pmSys, and if it has been an useful tool, their response were: *"The system is very useful as a monitoring tool. This is also the feedback we get from the (Tippeliga) teams. Furthermore, there is a big potential to make it even better!"*. We will present the our ideas and areas with potential to improve in the future work section in chapter 9.

In the next chapter, we present the system that allows the coaches to open a one-way communication message with the players, pmSys-Push.

Chapter 7

pmSys-Push

In this chapter, we present the middleware system that creates a one-way communication channel between the coach and the player, pmSys-Push. Firstly, we present known providers and their pricing for their services. We will then present the implementation of pmSys-Push, all its features and the services it uses to be able to send notification messages to the mobile devices with Android and iOS. Finally, we evaluate the result of integrating push notifications into pmSys with supporting claims.

7.1 Motivation

One of the problems the football players in the Norwegian Premier League had with Ohmage was that it had reminders that would have to be set up by the users themselves. The trainers had to call them personally if they wanted contact, it was also challenging and costly to remind the users to remember to register by text messages. This created an urge by the coaches in the Norwegian Premier teams where they wanted an easier way to remind their players without creating the reminders on each players phone. The player have time restrictions on their surveys where the time for the most accurate data collection is most effective, early in the morning when they wake up for wellness, right after their training sessions and once a week to report if they have injuries. With a tight schedule they might not remember to do the survey, which makes it very crucial for the monitoring that the players report at the correct time.

This is one of the goals of pmSys-push, to relieve the time coaches need to use to remind their players of registering a survey. Sending push notifications to players can be done manually or use the integrated cronjob function with pmSys-push, and because of this function, the coaches can rest assured that the push notifications will be sent when they have set it to run.

7.2 Related work: pmSys-push compared to other well-known providers

Push notification messages has given application developers a new of reaching out to their users. It has given the developers the possibility to gently tap the users on their shoulders to remind them, but it is also possible to be more intrusive if they really wanted. Most application developers work with one platform at a time, only iOS or only Android. But for those who wishes to send push notification messages to both iOS and Android at the same time, has to either build or use existing providers (see table 7.1). The prices and their features may vary, but what all the providers have in common is the free model. It is possible to test their system with limited features and limited push messages before upgrading which costs money. All the providers except for pmSys-Push have one thing in common. They all require that the application developer integrate their software development kit (SDK) to be able to send push messages natively. In pmSys-Push, there is no need for this since we also support hybrid applications. The developers only need to change two things on the push server; the database path where to store the callback tokens and the Google Cloud Messaging key needed to send push notifications to Android users.

Service	Initial cost	Limitations	Comments
pmSys	FREE	Need to maintain the server yourself	Unlimited push notifications. Free notification scheduling. Easy to scale. Zero costs.
Amazon	FREE	64 KB counts as 1 push notification.	0.5$ per 1M after the first 1M free push notifications. They also sell text message (0,75$/100) / email (2$/100.000) services.
Appoxee	FREE	Up to 250K users.	Unlimited push notifications. For automation it costs 500$. No text message costs.
Moblico	FREE	For 100 users, 250$ for 101 to 1500 users.	Unlimited push notifications. 0.03$ per text message and 0.01$ per email sent.
Parse	FREE	Up to 1M push notifications/m before costing money. Up to 1M API requests/m.	Their next plan costs 100$/m for + 20 requests/s and 2 concurrent jobs. 0.05$ per 1K extra push messages.
Pushwizard	FREE	Max 8 mobile applications. Max 4 push messages every month per device. For 200$/m 100 messages per device can be sent.	Their next plan costs 57$/m for 10 applications and more features. 299$/m (minimum) for unlimited push notifications.
Pushwoosh	FREE	Max 5 mobile applications. Scheduling costs 79$/m.	Unlimited push notifications on all plans. Their next plan costs 49$/m for 10 mobile applications. 749,95$/m for scheduling messages (max at 25 presets).
App Sales	FREE	Up to 100K push notifications/m before costing money.	Their next plan costs 49€/m for 2M push notifications. Unlimited push notifications costs 99€/m. No scheduling.
Urbanairship	FREE	Up to 1M push notifications/m before costing money.	0.001$/push after 1M free push notifications.
PushApps	FREE	Up to 1M push notifications/m before costing money. Max 5 applications.	Their next plan costs 19,99$/m for unlimited and scheduled notifications.
Xtreme Push	FREE	Max 5K devices. Max 2 mobile applications.	The free version comes with scheduled notifications. For unlimited applications and 100K devices it costs 299$/m.

Table 7.1: Examples of some push notification providers and their costs

7.3 Features

To be able to send push notifications, the system has to support both platforms. When sending a push message to a player, the system will automatically find out which OS the player uses before sending the push notification to the respective push message provider. A extra feature we have added after receiving constant feedback since push notifications were released, is automated push notifications (see section 7.4.5.

- Push messages to iOS devices
- Push messages to Android devices
- Cronjobs for automatic push messages
- Cold-boot for cronjobs

7.4 Implementation

7.4.1 API endpoints

pmSys-Push has multiple endpoints that is used frequently to provide the rest of the pmSys ecosystem push notification features, by using these endpoints it is possible access features that this system provides. The endpoints are used by both by the pmSys-app for subscription of the mobile device, and by pmSys-Trainer to send push notifications to the users and to setup automated push notifications.

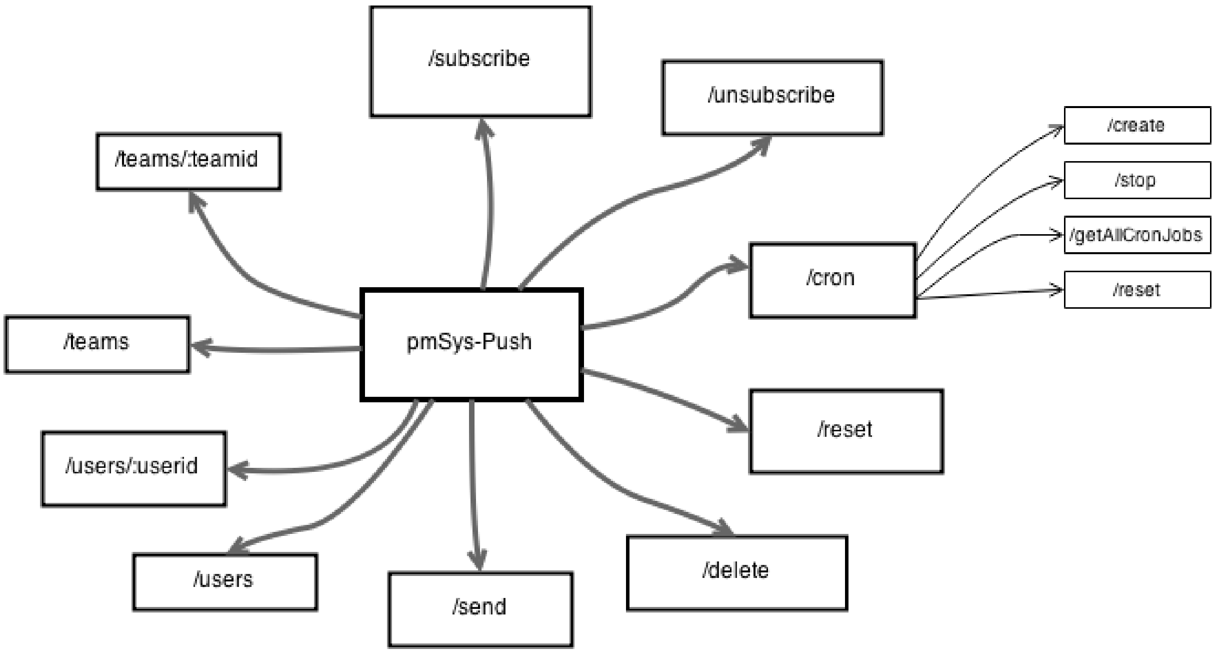


Figure 7.1: pmSys-Push endpoint map

7.4.2 Apple Push Notification service

Apple Push Notification Service (APNS) [92] is the service Apple Inc. provides developers to send push notifications to devices by using tokens generated by APNS spesifically for the device registered. Apple has a throughput on 9.000 notifications per second and for all iOS versions below 8.0, the maximum limit for each push message is 256 bytes, meanwhile for iOS 8+ the payload limit has increased to 2048 bytes due to the introduction of a new type of push messages that allow VOIP [93].

In pmSys, we use the NodeJS module *node-apn* [94], which is a JavaScript implementation which enables us to send push messages to APNS. This module is of such a high quality that Microsoft is using it on their Azure service to provide users the possibility to send push messages [95]. To be able to send out push messages to all devices the push server itself has to get a SSL Certificate issued by Apple, and the certificate is limited to a single application based on the application's bundle ID. Apple also has two development environments, Development and Production. The names are self explanatory, one is for when the application are still under development (usually used for testing purposes), when the application is uploaded to the App store the device token will automatically switch to the production environment. The development and production tokens are unique, which means that the tokens are not valid on cross of the environments.

7.4.3 Google Cloud Messaging service

Google Cloud Messaging (GCM) [96] is the push service Google provides their developers to send push messages to Android devices. To send push messages through GCM from pmSys, we use *node-gcm* [97] which replaced the old Google push network, C2DM [98]. As with APNs the device tokens is generated uniquely for each device registered with GCM with a special "senders key" which allows a push server with "senders key" to push messages to the application. This key is unique for a device; the token itself is bound to the device and not to an application. Which means that to send push messages to a device X you need an approved senders key Y. The senders key will be registered for the application's bundle ID which will show when a push message is received (see figure 7.2). The only way to get the senders key is to create a project through Google's Developer Console, the key is unique for each project (mobile application) and cannot be duplicated.

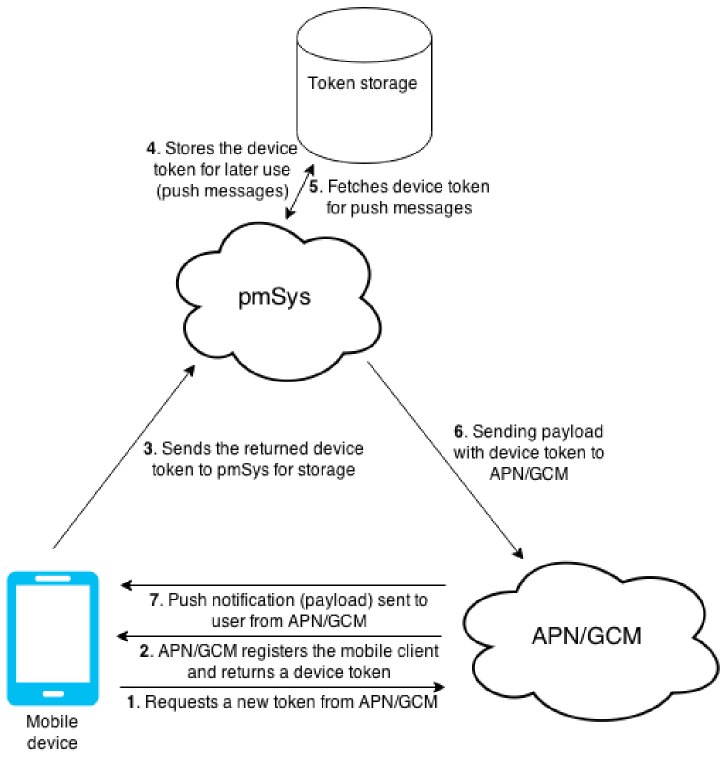

Token storage
4. Stores the device token for later use (push messages)
5. Fetches device token for push messages
pmSys
3. Sends the returned device token to pmSys for storage
6. Sending payload with device token to APN/GCM
7. Push notification (payload) sent to user from APN/GCM
2. APN/GCM registers the mobile client and returns a device token
1. Requests a new token from APN/GCM
APN/GCM
Mobile device

Figure 7.2: Overview of Apple Push Notification service and Google Cloud Messaging service

7.4.4 Token storage

We use MySQL [99] as device token storage, the main idea of using MySQL instead of other open source databases were due to the fact that Ohmage is using MySQL and the idea of merging these two databases together to make it easier to maintain 1 database. pmSys is only storing data that is vital to be able to send push notifications, data such as which team the user is a part of, device token for their device(s) and some extra parameters in order to provide sufficient data separation and security.

7.4.5 Cron

Cron is run by a crontab (a table of scheduled jobs), which are what UNIX-like operating systems are using to schedule shell commands at a specific time [100]. For node we are using *node-cron* [101] to do the same job as a cron would do on UNIX OS'es, but instead of using the commands crontab has we use a node-wrapper (in-memory) that runs native cronjobs (which are persistence on UNIX). To prevent it from disappearing when the server shuts down we have implemented our own cold-boot function that practically stores everything in a file, and the cronjobs will be loaded once the push server restarts. In this way no cronjob will be forgotten or lost due to shortage or OS crash, the file would still exist on the file system.

While on UNIX-based OS'es the command for running a script at midnight would be (see figure 7.3), but in node-cron it is possible to point to a JavaScript function instead, so for our use case it is easier to use this to send a push notification when we can just refer to a method we have created (see figure 7.4).

```
# * * * * * <command to execute>
# | | | | |
# | | | | |
# | | | | ------ day of week (0 - 6) (0 to 6 are Sunday to Saturday)
# | | | ----------- month (1 - 12)
# | | ---------------- day of month (1 - 31)
# | --------------------- hour (0 - 23)
# -------------------------- min (0 - 59)
```

Figure 7.3: Crontab on UNIX

```
var cron = new Cronjob({
  cronTime: '00 00 00 * * *',
  onTick: function() {
      // Everything in this are will run when it's midnight!
      // sendNotification(sendTo, message)
    },
    start: true
  });
```

Figure 7.4: Node-cron with NodeJS

7.5 Evaluation & Discussion

In chapter 5, we discussed the motivation on why pmSys-app was created. It was simply because the Ohmage MWF application was not suitable for our focus group, but it also took too much time to set up and the features were not consistent on all platforms. The pmSys app has several similar features as Ohmage, however, pmSys have added other features as well. The extra features made it possible for the coaches to send push messages through pmSys-trainer, push notifications with simple messages about they need to answer the surveys after each training session or every morning. Even though the features works seamlessly for the coaches, there is still problems hidden behind the implementation of push messages that we will discuss. There is a limitation of how much data you can send with each push message (see table 7.2), which makes it harder to send large push messages to the users. There two main types of push messages that both iOS and Android can accept; standard push messages and Silent (system) push messages. With a silent push message, it is possible to send a large chunk of data where the mobile application can fetch the data to update itself (possibility of force update of mobile application content). In pmSys, we are only using the standard push messages, because it is not needed for interactive content replacement on the client side.

OS	Payload limit
iOS < 8.0	256 bytes
iOS > 8.0	2048 bytes
Android	4096 bytes

Table 7.2: Payload limit on iOS and Android

The way pmSys-Push can offer to enable the possibility to use one service to send push messages to both iOS and Android could be a game changer. It is arguable that hosting the server yourself could be a hassle, but if you were to have a massive user base, pmSys-Push could save you for quite a lot of money compared to the other services listed in table 7.1. All of the services that provide push messages also requires that you implement and add their software development kit (SDK) to your application, this is to be able to use all the services that pmSys provides. The main thing that differentiates between pmSys-app and any other applications that uses SDK's is that pmSys-app is a hybrid application. There is no need for SDK to be able to accept push messages, all you need is the senders key for GCM. As a result, deploying an application with push functionality takes nearly no time, since all you need to change is the "senderID" (Figure 7.5) in the source code and the application would be receiving push notifications.

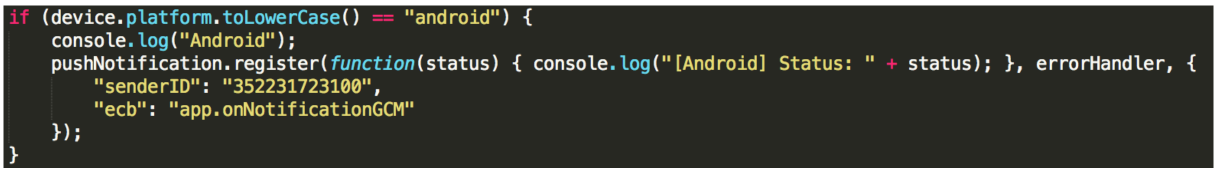

```
if (device.platform.toLowerCase() == "android") {
    console.log("Android");
    pushNotification.register(function(status) { console.log("[Android] Status: " + status); }, errorHandler, {
        "senderID": "352231723100",
        "ecb": "app.onNotificationGCM"
    });
}
```

Figure 7.5: SenderID for GCM

When cronjobs was implemented and activated, the Norwegian National football team the answer rate increased rapidly in matter of days (see figure 7.6). The reason for the increased answer rate is not entirely because of the automated push notifications, but it might have a supporting role for the specific timeframe. We received instant feedback that automated push notifications were a long awaited feature, because they wanted to send notifications at the time every day and that it is difficult to remember to do so themselves.

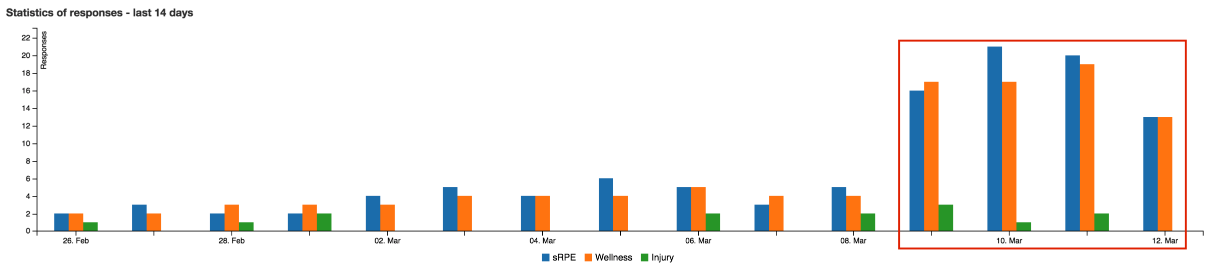


Figure 7.6: The Norwegian National football team answer rate after cron was introduced

7.6 Summary

In this chapter, we introduced pmSys-Push where it enabled the possibility to open a one-way communication channel between the coach and the players. The Ohmage application did not have any support for push notifications, which made it difficult to remind the players to answer their surveys. With the introduction of remote push notifications, the answer rate instantly increased, and the coaches could be 100% sure that the push messages would be sent at the specific time defined.

The pmSys-Push system allows anyone to set up their own push notification service with support for both Android and iOS. The negative thing about the system is that the users has to maintain the server themselves, if a problem with the system comes up, the users has to troubleshoot themselves or wait for support. Meanwhile with service providers, they takes care of everything and also notify if something were to happen.

In the next chapter, we present a system for collecting, storing and processing objective data before it is presented for the coaches in the web portal. The pmSys-Tesseract inherits all the functionality requirements defined in chapter 4, and with it, new challenges within security, storage and processing power comes with it.

Chapter 8

Objective data: Integration of third party hardware

In this chapter, we implement and integrate third party hardware and wearables into pmSys, and this system has been chosen to be named, **pmSys-Tesseract**. The idea behind the name comes from the Marvels Universe, where the Tesseract cube is a vessel for unlimited energy [102]. In the case of pmSys, the unlimited energy is a metaphor for the objective data that can provide endless possibilities. Even though subjective data has been one of the main building blocks of the pmSys system, the collected subjective data could easily be tampered by the football player. Therefore, objective data could be the confirmation bit of a football player's physical state. First, we discuss what is needed to integrate third party hardware and the issues that arise when implementing the support for third party hardware, then we will present how it has been done to make the whole integration work seamlessly with the rest of pmSys.

8.1 Motivation

Researches has been using accelerometers to study a player's physical state since 1980s [103], but it has been a limit of what the technology could collect. By going from collecting accelerometer data once a minute into multiple values in a second enables a new way of monitoring physical activity. This also opens up the possibility of using multiple wearables and systems to analyze even better, because the more detailed and different types of data a system can have, the higher is the monitored precision of the actual physical state of a player. This is what Milan Labs has done for a very long time (see section 2.6), and their results have been undeniable positive. Tromsø IL has also been using accelerometers and other systems for a couple of years, and their results has also shown that the value of using third party systems has helped them detect their player's physical state on areas where the eye cannot analyze [104].

For a system like pmSys, subjective data is not enough to prove whether the player is injured or fatigued. This is why third party hardware and wearables has been used vividly to capture objective health data for reaching better results, however, the problems with these systems are that they all have different functions and the fact that they focus on collecting different type of data. In order to integrate pmSys in the most efficient way, it is necessary to have an API open to access the data in real time, although there is not a given fact that a product owner of third party hardware's wants to share the data collected. With this type of data follows a lot of problems, the biggest problem of them all is the quantity of the data. The data may be returned in all sorts of ways, some has a logical structure of the data, some just purely dumps all their data into XML (eXtensible Markup Language) or comma-separated values [83] formats.

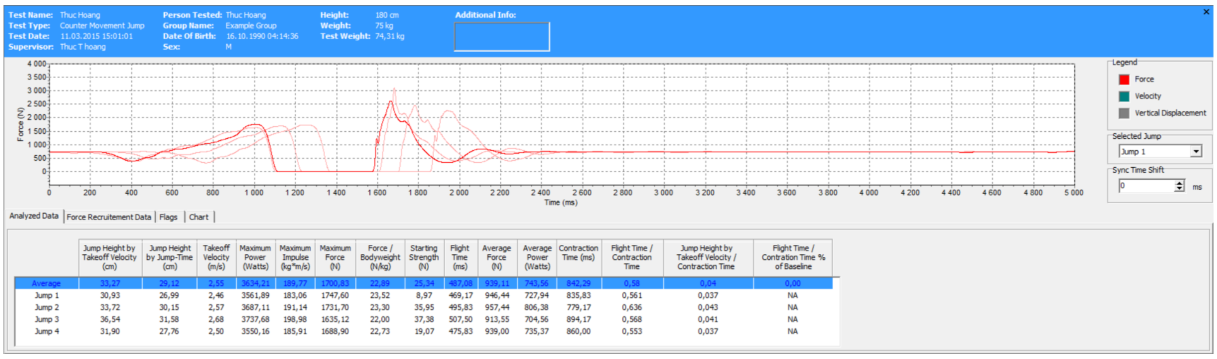
8.2 Example devices - for objective data

In the beginning, the main focus was to create a new way and optimize that particularly way, so that it could become easier for the player to answer their surveys. By creating an mobile application, we got rid of pen and paper which has been used for a very long time within monitoring. The data that came from these player surveys is categorized as **subjective data**, which is data relative to a player's assessment of themselves. This data can only be trusted to a certain degree, since it only registers what the player perceives, not how the body perceives the training sessions. It takes a lot of time to just calibrate a player's perception of the training sessions before it can become useful to the coach. A new type of data source has to be added to make the subjective data trustworthier, and that problem can only be solved in one way.

A way to do this is by finding hardware and third party wearables which collects measurable and observable data from players, but also a fail-safe way to prevent manipulating the data source (which is possible with the subjective data). Just in 2014 and 2015, a whole bunch of companies launched a large scale of wearables that allowed the users to collect and monitor themselves. Not only were they optimized to collect data, but they were also within the price range of normal users. It is important to note that even though the wearables were created for an active user in mind, it does not mean that non-active users cannot use it. Since there is an ocean of sensors and wearables, we have picked out those we know for sure is suitable for tracking training sessions and daily fitness. The devices can be seen in the list below:

- Fitbit
- Jawbone
- ZXY sports tracking
- Moves
- HUR Labs Jumping Board

In the pmSys system, we integrate Fitbit and HUR Labs Jumping Board as a proof of concept. These are the two most used third party hardware by the National team and the Premier teams that have used pmSys. The early idea of retrieving the data from the HUR Labs Jumping Board [105] was to monitor the SQLite [82] database HUR Labs use to store all the data from the jumps. The idea was to create a program locally on the computer that would automatically upload all the data to our Tesseract-system for data processing. The idea sounded pretty simple, but the problem arose when we started decoding the binary data stored in the database. Each data point were split into 4 floats, and since the system records at a rate of 1200 samples per second, the total number of data points was at 6000. The number of data points were easy to parse, but the real problem is the fact that the decoded values we retrieved were not similar the values we read of the graphs in the HUR Labs software.



Test Name: Thuc Hoang
Test Type: Counter Movement Jump
Test Date: 11.03.2015 15:01:01
Supervisor: Thuc T hoang
Person Tested: Thuc Hoang
Group Name: Example Group
Date Of Birth: 16.10.1990 04:14:36
Sex: M
Height: 180 cm
Weight: 75 kg
Test Weight: 74,31 kg
Additional Info:

Selected Jump: Jump 1
Sync Time Shift: 0 ms

	Jump Height by Takeoff Velocity (cm)	Jump Height by Jump-Time (cm)	Takeoff Velocity (m/s)	Maximum Power (Watts)	Maximum Impulse (kg*m/s)	Maximum Force (N)	Force / Bodyweight (N/kg)	Starting Strength (N)	Flight Time (ms)	Average Force (N)	Average Power (Watts)	Contraction Time (ms)	Flight Time / Contraction Time	Jump Height by Takeoff Velocity / Contraction Time	Flight Time / Contration Time % of Baseline
Average	33,27	29,12	2,55	3634,21	189,77	1700,83	22,89	25,34	487,08	939,11	743,56	842,29	0,58	0,04	0,00
Jump 1	30,93	26,99	2,46	3561,89	183,06	1747,60	23,52	8,97	469,17	946,44	727,94	835,83	0,561	0,037	NA
Jump 2	33,72	30,15	2,57	3687,11	191,14	1731,70	23,30	35,95	495,83	957,44	806,38	779,17	0,636	0,043	NA
Jump 3	36,54	31,58	2,68	3737,68	198,98	1635,12	22,00	37,38	507,50	913,55	704,56	894,17	0,568	0,041	NA
Jump 4	31,90	27,76	2,50	3550,16	185,91	1688,90	22,73	19,07	475,83	939,00	735,37	860,00	0,553	0,037	NA

Figure 8.1: HUR Labs Jumping board visualization

In figure 8.1, we can see the visual graph the software produce for each jump, where the selected one is highlighted (selection can be done on the right side). This visualization is very simple with little interaction, however, the results given from the software, as processed data is remarkable. The values can easily be computed by using formulas, but since the data must be extracted and then uploaded to pmSys-Tesseract, hence calculating the same values twice is redundant. All jumpers has their own profile, where information has to be entered beforehand. Information such as name, weight and height must be entered before we are allowed to proceed with the tests. The board will automatically re-weight the test person to make sure that the calculations will be correct, but the board cannot measure the height of the test person. This means that the values entered must be correct, or the calculations based on these two values becomes useless.

Another problem we did not consider was the link between each user in the system and the username in pmSys, and how to do this in a secure manner. We also did not count for offline-support since we originally wanted it to upload for each time the jump test were taken. This would be problematic if the computer do not have a network to upload the data, and there would be no guarantee that the medical staff would remember to upload the data when they got back to their office with a network connection.

8.3 Background

8.3.1 Big data

Big data [106] is a term that has gotten a lot of attention the last couple of years, due to the new digitalization where everything is available on the internet (see section 8.3.2 for more on this). This creates the need for more advanced data analysis when reaching large data sets. This is what big data is all about. It is a term for large data sets with the complexity that normal ways to process data is no longer possible. The input data is not necessary sorted nor processed before integrated in a system, that is why it is essential to create data models to make it easier to create correlations between two data sets. For example, the correlation between how refreshed a player may feel after only sleeping X hours can be calculated over time, and also give the user a prediction of the next expected "refreshed" value when sleeping X hours over Y days.

With big data follows big challenges concerning large data sets, such as searching through it all, storing, transferring, visualization and many others. The reason for these challenges is because of old and new hardware that enables data collection, such as mobile devices (participatory sensing), software logs and wearables is now kept and added into the database that increases in size. The idea of collecting a lot of data can only be useful if it is processed within a useable timeframe. The more data it has to process at once affects the time used on returning all the data to the user. It is important to keep the response time within a set time limit, or else it would be redundant in using a system that would use minutes, hours or even days to process the data. Regarding pmSys, we are still not at the limit where it would be safe to call our datasets for big data, but by integrating all the third party hardware, it would not take too long before searching through all the saved records would take quite a bit of time. We also do not need to create a distributed system for processing all our data, but in the future, if the dataset gets too big for the traditional way of fetching it, it is possible to use existing tools to process all the data. System such as Hadoop [107], GridGrain [108] and HPCC [109] offers a lot of functionality. All of them is built on the idea of a distributed processing system where all the data is scattered on all the nodes, and then use the power of each cluster to process the data faster (see figure 8.2). Some of them uses a in-memory approach, however, that is very costly since memory is expensive compared to normal storage space like on a hard-drive, but what you can gain from it is a much quicker solution since the data is already in-memory and ready to process on the fly.

Apple announced during their WWDDC 2014 presentation that they will start to focus on big data and collecting multiple systems into their own health tracking platform; *HealthKit* [110]. This is big news for

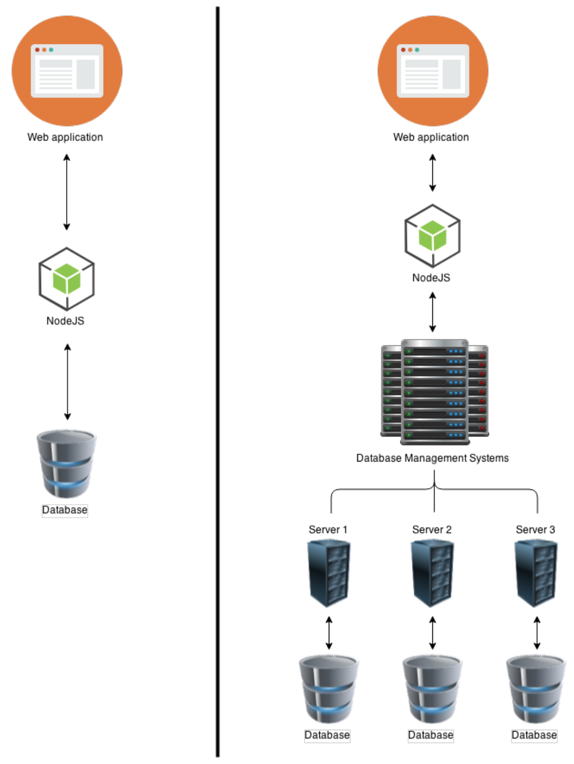
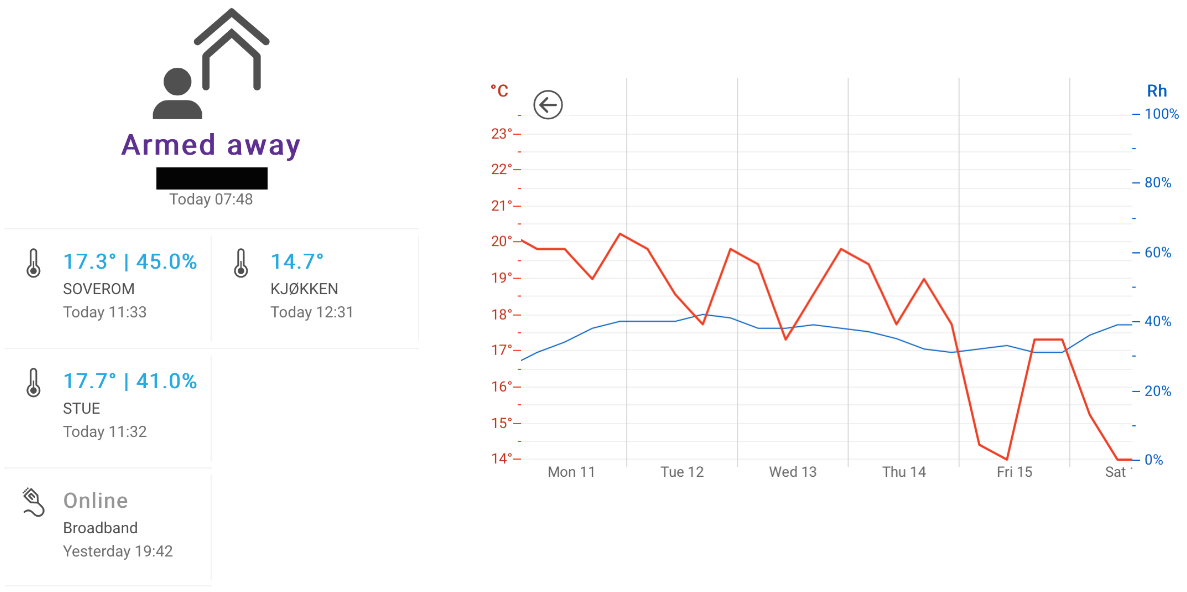


Figure 8.2: Traditional database system versus distributed database systems

mobile health monitoring, since Apple also revealed that they had been working with medical clinics the last six years to develop this platform. This means that the ideology behind Open mHealth and HealthKit is not too different compared to each other (see section 2.3). Because they encourage reuse of code and by collecting everything to one centralized platform, it all can be analyzed.

8.3.2 Internet of Things



Armed away
[redacted]
Today 07:48

17.3° | 45.0%
SOVEROM
Today 11:33

14.7°
KJØKKEN
Today 12:31

17.7° | 41.0%
STUE
Today 11:32

Online
Broadband
Yesterday 19:42

Figure 8.3: Example of what simple sensors can provide of analytics

There have been a lot of talks about **Internet of Things (IoT)** [31], i.e., how everything that can record data and connected to the internet goes under this term. Everything with embedded electronics can be interconnected with each other, and it is possible to create a network of systems to both control and analyze objectively. Third party hardware and wearables have an important role in pmSys since they are actively used to collect and monitor football players.

Figure 8.3 shows us how simple old technology can be reused to provide analytic data in real time if it is configured correctly. The visualization gives us an overview of the temperature in the last couple of days. This information can tell us when the house has a high humidity (Rh) or when the temperature is high/low. Companies such as Bekk has for example successfully utilized IoT to analyze their employees walk patterns in different areas at their headquarters [111]. Using sensors to analyze how much the stairs is used, or retrieving travel routes for the public transportation available in the area is just a few examples. This is just a proof of concept, however, the results from the project are interesting and proves that any piece of data can be valuable if used correctly.

8.3.3 Countermovement Jumps

The Norwegian National team and the Premier teams applies the HUR Labs Jumping Board to do one specific type of jump in order to analyze the muscle effect of the players legs [112]. If a jump were performed with a maximal effort when the player is recovered, the value from this jump would be set as the baseline value for all future jumps during the season. If a jump with maximal effort is performed, but the jump is less than 92% of the baseline, then the muscle is fatigued. High training loads combined with too little recovery time in between each day can cause the condition.

8.4 Fitbit

Fitbit [113] offers a lot of wearables for different types of usages, but the main thing all the wearables do good is to track a lot of things about the human behavior. Step counters, calories burned and sleep are just some of the few features any Fitbit armband can offer. The company has an open web API where developers can authenticate themselves, and then gain access to the API to retrieve data. The user has to grant the developers access to their data, but after that any data that the bands register can be retrieved from Fitbit if it has been uploaded to their servers.

8.4.1 Features

The basic features can be found in the list below, although they sound pretty simple, the data from this can help the user a lot when the data is analyzed. By simply using data such as sleep quality can help the user identify why and when problems occur. Or if the user is feeling sore one day, then the Fitbit band can show a visual graph with data about how much and how many steps a user has walked.

The Fitbit dashboard has support for manually logging activities, food diets, weight and sleep. By logging these areas, it is possible to see how much activity the user has done and the changes the user has done over time. The only problem with this way of logging is that it requires manual labor in order to be as effective as possible. As a default, the Fitbit automatically logs the steps the user takes, and this will be calculated into distance and expected calorie consumption for the activity done by the user (see figure 8.4). There is also support for user input of data for better monitoring of a user's progress on their dashboard. This can be used if the user for example is currently charging the band, within the time it is off, the activity levels are high, i.e., manual activity logging are needed to get the correct data calculations.

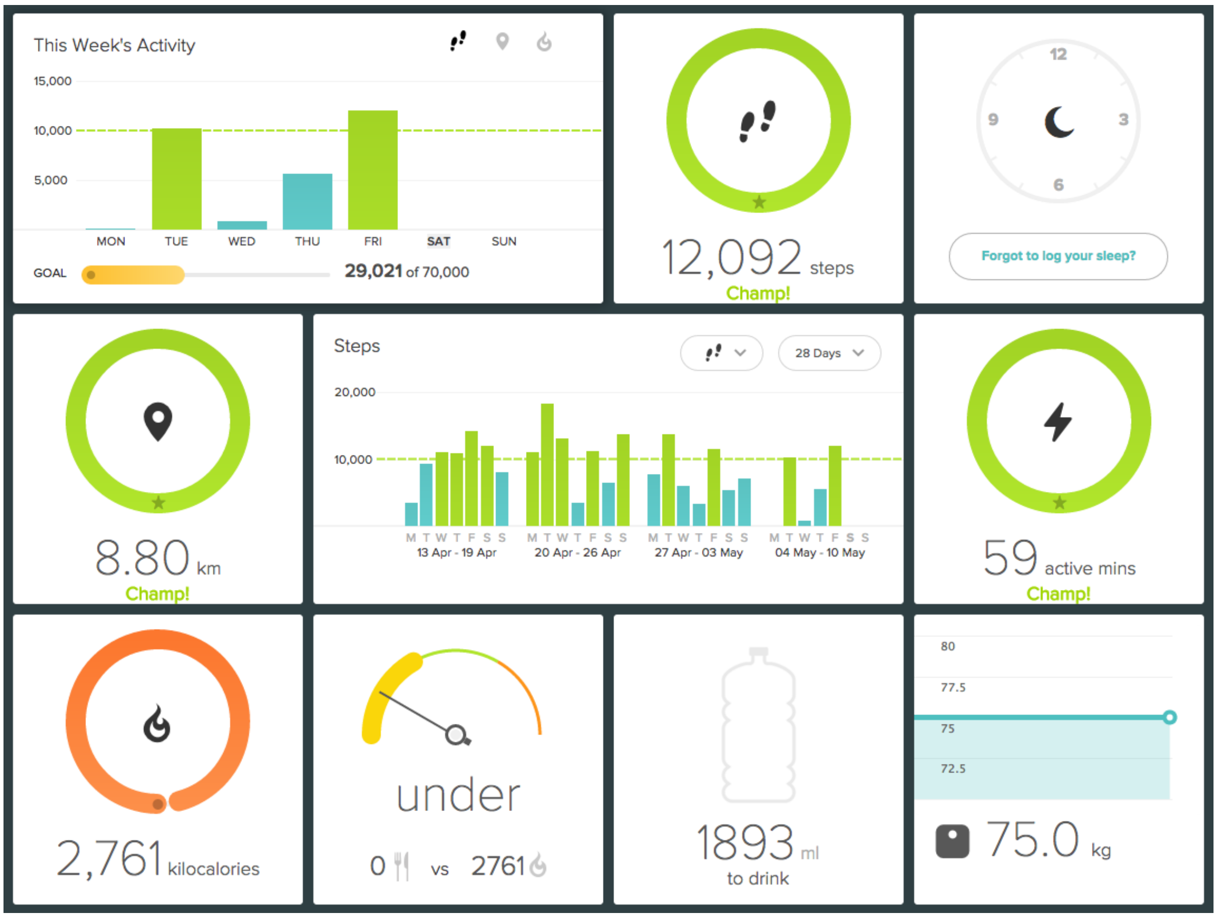


Figure 8.4: Fitbit dashboard visualization

8.4.2 Limitations

The band itself has a battery time consisting of 6-10 days depending on which type of Fitbit device the user has. It is not possible to use the Fitbit while it is charged. It has to be de-assembled and attached to an external charger in order to be able to charge it. It is also important to note that there are two ways to synchronize the data from the band to Fitbit's databases. The first way is by using a given wireless dongle that is attached to a computer, which will connect to the band through Bluetooth and then upload the data to the database if there is a network connection. The second way is to synchronize through a mobile device, which will be uploaded to the database when it has a network connection. One of the biggest limitations by using these type of bands from Fitbit is that if the user want the best analysis possible, then it would require a lot of manual work in order to make the data as consistent and effective as possible.

8.4.3 Uploading data to pmSys

The players can link their Fitbit account to their own pmSys user through the pmSys-app, and give pmSys read-access to their data from Fibit's web API. By going into the profile, the player can link his Fitbit account to his pmSys user account (see figure 8.5). Fitbit uses OAuth [114] which is an open standard for authorization of users in a system. It is used by numerous systems and services where it is the middleware-security layer that takes care of authorization by linking users up against access tokens from their authorization server.

When clicking on the Fitbit image, a login screen is asking the players for their Fitbit user credentials.

When the credentials is typed in correctly, they will be asked to give *pmSys* **read-only access** to their data, and this is used purely to visualize the user data so that the coaches can compare the subjective data (surveys) and the objective data (Fitbit). The whole process takes about one minute, and then the next time the coach wants to see the objective data, he can gain access freely since pmSys has been given access to the data of the player.

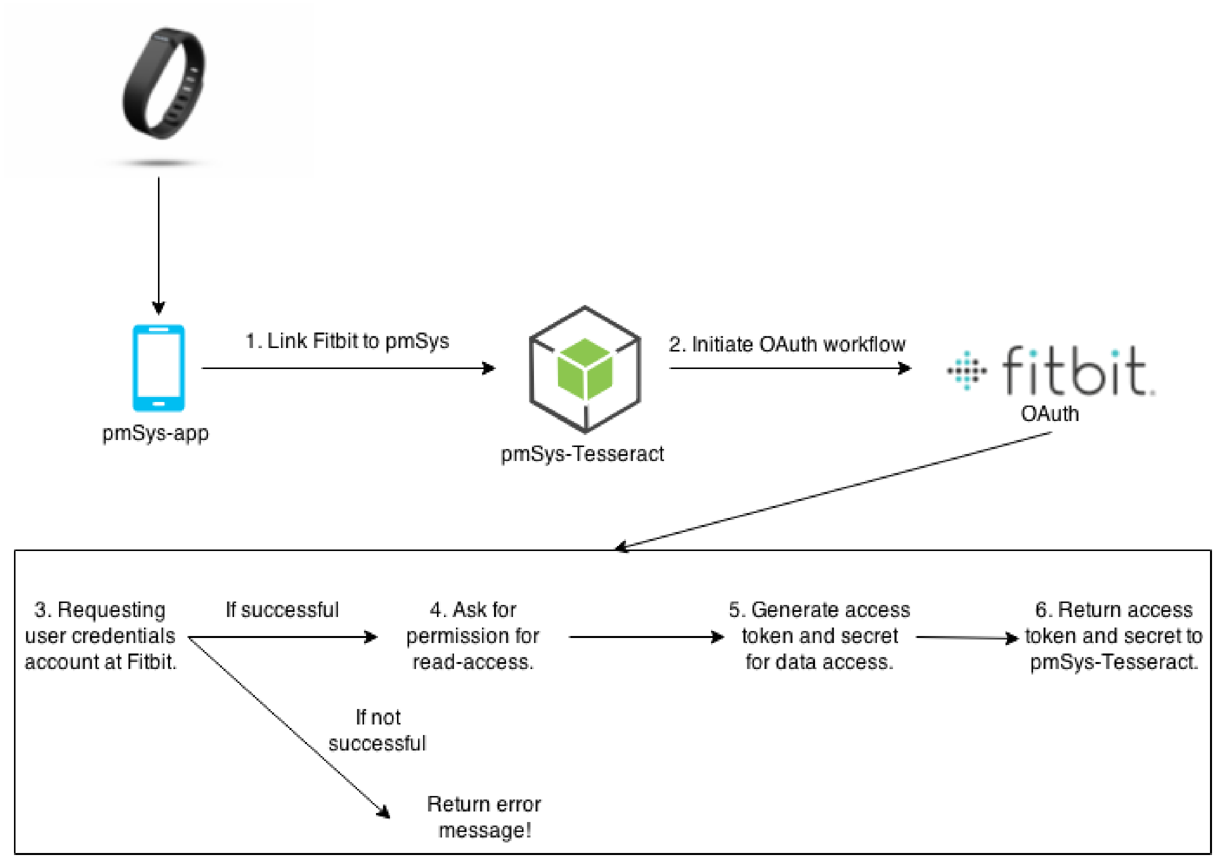

1. Link Fitbit to pmSys
2. Initiate OAuth workflow
pmSys-app
pmSys-Tesseract
fitbit
OAuth
3. Requesting user credentials account at Fitbit.
If successful
4. Ask for permission for read-access.
5. Generate access token and secret for data access.
6. Return access token and secret to pmSys-Tesseract.
If not successful
Return error message!

Figure 8.5: OAuth workflow for read-access to user data

8.4.4 Data processing

The data has already been formatted when pmSys retrieves it from Fitbit's web API. The only thing pmSys does with the data from the web API, is to extract the data that is needed for visualization. When the data is extracted, it is then joined into a JSON object, before returning the object to the client for visual rendering.

8.4.5 Visualization

The data from Fitbit is also from the selected data and is presented in pie charts, the reason for this is to show how efficient the day has been. The data the Fitbit device collects as a default is presented here, and for other information such as weight changes, diet or calorie burned requires manual work. (see figure 8.6). This information could be useful when a player has a declining performance, and can help the coach and the medical staff to find out a solution to help the player return back to the peak physical state.

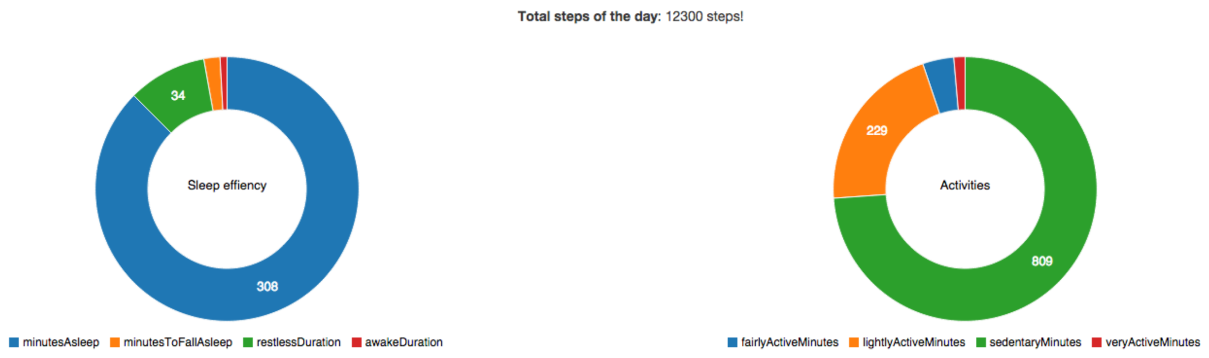


Figure 8.6: Visualization of sleep time and activity levels

8.5 HUR Labs Jumping Board

The HUR Labs Jumping Board (Force platform FP4) [105] is used by the Norwegian Premier teams to measure how much force the player can apply to the board. This is to detect fatigue of the player based on a list of jumps. The board itself is full of sensors that registers 1200 times each second, and each jump has gets allocated 5 seconds in case the player has a delayed jump (not correct timing). The board has a weight sensor that registers the weight of the jumper when stepping on it, and the board also has a force sensor that measures in Newton. The players have to jump multiple times with a specific technique in order to analyze the jump correctly. The Norwegian Premier teams is only doing *Counter Movement Jumps*, and therefore pmSys will only support that particularly jump type.

8.5.1 Features

The jumping board has a lot of features, and all of them is concerning evaluating the types of jumps the board supports. For each jump a set of variables is evaluated and presented in the software for the board, and with this data the medical staff can analyze the results and look for fatigues players. In the beginning of each football season, the player is forced to rest before they perform a baseline jump. A baseline jump [115] is the very first value that is used to compare all other jumps through out the season, and this value makes it possible to detect fatigues or injured players. If a jump is below 92% of the set baseline, then the player should be checked more thoroughly for injuries since a result with less than 92% of the baseline is classified as a muscle injury [112].

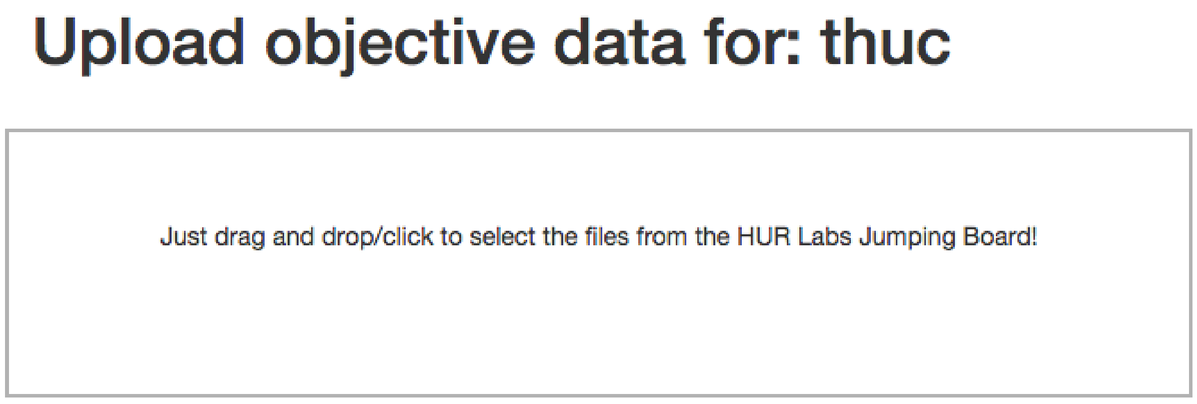
The current version of the platform (PF4) supports these type of jumps:

- Squat Jump
- Counter Movement Jump
- Drop Jump
- Elasticity Test
- Fatigue Test

8.5.2 Limitations

The results from a jump may not be entirely correct, since factors such as technique and timing can heavily affect the result. This is also why the players is forced to jump multiple times to make sure that the technique is correct, which is time consuming, but it is crucial for a correct diagnosis of a players leg's ability to create force in a short amount of time. Another limitation with the board that can affect some teams negatively is the cost of purchasing it. The board is expensive, and unless the teams really want to invest in digital analyzation of their players then the board may not be the best way to invest the money. It is also important to note that the software is only compatible with the operating system, Windows.

8.5.3 Uploading data to pmSys


Upload objective data for: thuc

Just drag and drop/click to select the files from the HUR Labs Jumping Board!

Figure 8.7: Upload interface for jumping board data

Since the idea of automatically uploading the data to pmSys when the jumps were performed was not a suitable feature, we had to find a way to upload the data without the data being wrongly linked to the wrong player. The truth is that there will always exist a chance of data being linked to the wrong player by mistake. We found out that the best solution was create a dynamic upload page for each player (see figure 8.7). The extracted text files could be either drag-and-dropped into the box, or the coach could click on the box to get a prompt where the files could be selected in bulks. The extracted data from the jumps is split in two files, one of them consists of the raw data of the jump(s) (figure 8.8), and the other file of the processed data values that the software has computed (figure 8.9).

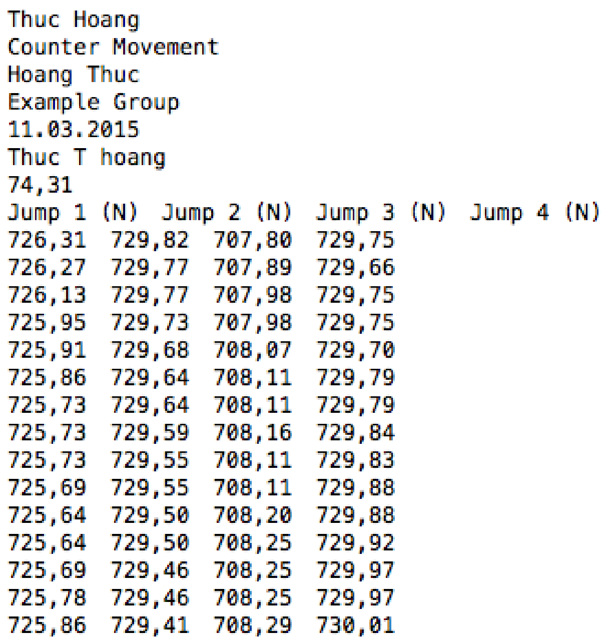
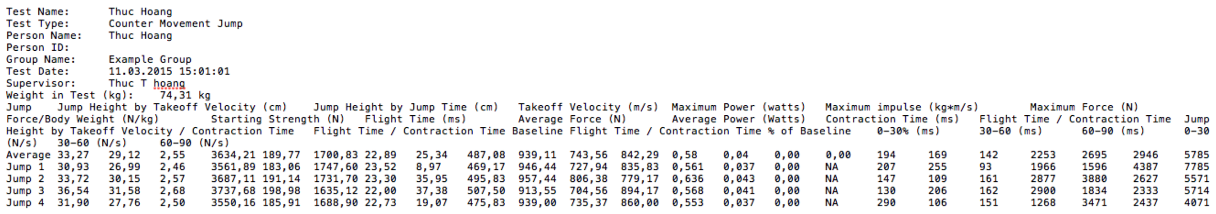

```
Thuc Hoang
Counter Movement
Hoang Thuc
Example Group
11.03.2015
Thuc T hoang
74,31
Jump 1 (N)  Jump 2 (N)  Jump 3 (N)  Jump 4 (N)
726,31  729,82  707,80  729,75
726,27  729,77  707,89  729,66
726,13  729,77  707,98  729,75
725,95  729,73  707,98  729,75
725,91  729,68  708,07  729,70
725,86  729,64  708,11  729,79
725,73  729,64  708,11  729,79
725,73  729,59  708,16  729,84
725,73  729,55  708,11  729,83
725,69  729,55  708,11  729,88
725,64  729,50  708,20  729,88
725,64  729,50  708,25  729,92
725,69  729,46  708,25  729,97
725,78  729,46  708,25  729,97
725,86  729,41  708,29  730,01
```

Figure 8.8: Raw jumping data

If a player performs multiple jumps in the same session, then the extracted data will organize all the jumps in the same file separated by tab-separated values [116] for each jump, but the next data point for that particular jump will follow in the next 6000 lines. For each jump, the system has to parse through 6000 data points to add it to the data structure we use to store each jump's data, and also look for the highest force value of each jump while doing it. Even though it might sound a lot, the system parses through and stores all the data it needs in a matter of seconds. For a monitoring system such as pmSys, performance is not the most important feature, but it has not been neglected while we have developed the system.

The second part of the data that is uploaded to pmSys is the computed values from the HUR Labs software. The data we get from here is the calculated values of the jump data; from the data we can retrieve data such as jump height, force, velocity etc. The analysis data is already pre-computed by the software on extraction, which means that we do not need to calculate the values ourselves, even though it is possible by using physics formulas [117].



```
Test Name:      Thuc Hoang
Test Type:      Counter Movement Jump
Person Name:    Thuc Hoang
Person ID:
Group Name:     Example Group
Test Date:      11.03.2015 15:01:01
Supervisor:     Thuc T hoang
Weight in Test (kg):    74,31 kg
Jump    Jump Height by Takeoff Velocity (cm)    Jump Height by Jump Time (cm)   Takeoff Velocity (m/s)  Maximum Power (watts)   Maximum impulse (kg*m/s)        Maximum Force (N)
Force/Body Weight (N/kg)        Starting Strength (N)   Flight Time (ms)        Average Force (N)       Average Power (Watts)   Contraction Time (ms)   Flight Time / Contraction Time  Jump
Height by Takeoff Velocity / Contraction Time   Flight Time / Contraction Time Baseline Flight Time / Contraction Time % of Baseline    0-30% (ms)      30-60 (ms)      60-90 (ms)      0-30
(N/s)   30-60 (N/s)     60-90 (N/s)
Average 33,27   29,12   2,55    3634,21 189,77  1700,83 22,89   25,34   487,08  939,11  743,56  842,29  0,58    0,04    0,00    0,00    194     169     142     2253    2695    2946    5785
Jump 1  30,93   26,99   2,46    3561,89 183,06  1747,60 23,52   8,97    469,17  946,44  727,94  835,83  0,561   0,037   0,00    NA      207     255     93      1966    1596    4387    7785
Jump 2  33,72   30,15   2,57    3687,11 191,14  1731,70 23,30   35,95   495,83  957,44  806,38  779,17  0,636   0,043   0,00    NA      147     109     161     2877    3880    2627    5571
Jump 3  36,54   31,58   2,68    3737,68 198,98  1635,12 22,00   37,38   507,50  913,55  704,56  894,17  0,568   0,041   0,00    NA      130     206     162     2900    1834    2333    5714
Jump 4  31,90   27,76   2,50    3550,16 185,91  1688,90 22,73   19,07   475,83  939,00  735,37  860,00  0,553   0,037   0,00    NA      290     106     151     1268    3471    2437    4071
```

Figure 8.9: Calculated values from the software

8.5.4 Data processing

The only hardware that needs real data processing is the HUR Labs Jumping Board. This gives an output of 6000 data points for the 5 second window the player has to do a jump while being on the board. Usually this type of data is split by comma (CSV), but for this application all exported data is printed in a new line for each data value. This creates a lot of lines to scroll through, although it makes the data more readable to a certain extent. The real challenge with this data is when to cut the data, since it may contain 6000 data points for each jump, there is actually only about 2-3K[1] data points that is actually relevant for each jump. If we look at figure 8.1, we can see that the data became non-relevant already at 2200 milliseconds which is less than half the the data size. When storing data of this size, where each jump would contain from a range of 70-100K characters then it would take about 80 Kilobytes to store each data entry.

There are several ways to find the timeframe to find only the necessary data; the problem is to find out all the parameters needed to use some of the algorithms we have tested this on. What both methods needs is to find out the highest value the specific jump and also record which millisecond it occurred on. In pmSys we solved this by looping through all the data values which has already been added to an array, meanwhile searching for the highest possible value (see figure 8.10). A pitfall here could be that the loop will not stop until it has looped through all the entries, which could potentially take a very long time if someone were to manipulate the export data from the HUR Labs software.

Each team has about 30 players jumping once a week, and since pmSys is only storing the values of the highest jump of a player, it means that in one month the estimated space needed to store all the data is 30*4*80/1024 = **9,375 Megabytes** for each month. In a year, it would mean that all the data would use **112,5 Megabytes** in the database for each team. If we did not cut the data at a certain point, then the storage would probably be at least double of the estimated storage value.

Next, we evaluate two possible methods for detecting where it would be best most suitable to stop storing data.

[1]Kilo = 1000

```
function findRelevantInterval(counter, array) {
  var deffered = Q.defer();
  var max = 0;
  var pos = -1;

  for(var i = 0; i < array.length; i++) {
    array[i] = parseFloat(array[i]);
    if(array[i] > max) {
      max = array[i];
      pos = i;
    }

    if(i == array.length-1 && pos != -1) {
      var relevantInterval = array.slice(0, pos + 1000);
      var json = {}; // Create a JSON object
      json["counter"] = counter;
      json["data"] = relevantInterval;
      deffered.resolve(json);
    }
    else {
      deffered.reject({"code": 404, "msg": "Error! Position not found!"});
          // If pos is still -1, then it has not been changed
    }
  }
  return deffered.promise;
}
```

Figure 8.10: Algorithm to find the highest value in the array

8.5.4.1 Method 1: By cutting data with a hardcoded value after highest value

The current implementation of data selection is hardcoded into pmSys. The current value is set to 1K data points in case of abnormal jumps, and to make sure that no data is cut off too early. This is the quickest solution based on time used on calculations, but as a trade-off the system will be storing a lot more data that takes disk space over time. As a temporary solution it works just fine, but if it is going to be used in the long run then we believe that this method is not suitable in the long run because of the unnecessary disk usage. If we reuse the calculations in equation 8.5, this method would cut at **2664 data points**.

$$\begin{aligned}
\text{Highest value in jump} &= 1664 \\
\text{Hardcoded value to cut} &= 1664 + 1000 = 2664 \\
\\
\text{Difference between method 1 and method 2} &= 700\text{data points} \\
\text{If each datapoint contain 6 characters * 700 data points} &= 4200 \text{ characters extra to store} \\
\text{Each character uses 3 bytes [118]} &= 4200 * 3 = 12600\text{bytes} = 12,3\text{Kilobyte}
\end{aligned} \tag{8.1}$$

8.5.4.2 Method 2: By using gravitational and kinetic energy calculations

It is possible to use physics to solve this problem, but only if we have all the variables that is needed to be able to calculate where the data should be cut off. Because of this, the data has to be pre-processed to retrieve this data, but due to the time constraints it is not possible to implement at this time. Even though it is not possible to implement this at the moment, it is still a viable way to decide when the data is has no relevance to the graph statistics at all.

First, we calculate the **Gravitational potential energy** (GPE) [119] and convert it into **Kinetic energy** (KE) [120], to find out how long it would take before all the energy the player has on the way down is negated by impact. In order to do this, we need to apply the equations 8.2 and 8.3 to find out how much potential energy the player has before the player is starting to descend down to the ground from that point.

$$GPE = mgh \tag{8.2}$$

$$\begin{aligned} Kineticenergy &= \frac{1}{2}mv^2 \\ v &= \sqrt{KE(\frac{1}{2}weight)} \end{aligned} \tag{8.3}$$

When the player starts to fall down, the GPE is converted into KE. The longer the player is falling, the less GPE there is, but the KE is growing until the player hits the ground where all the energy is transfered into the board. Before we can calculate how long time it takes before the player hits the board, we need to calculate this value (GPE), and then we can use the equation for Kinetic energy to find out the velocity of the player before they hit the board (see equation 8.3). The velocity value we receive after doing all these calculations is needed to be able to calculate the time used before the player hits the ground, by using the equation in equation 8.4.

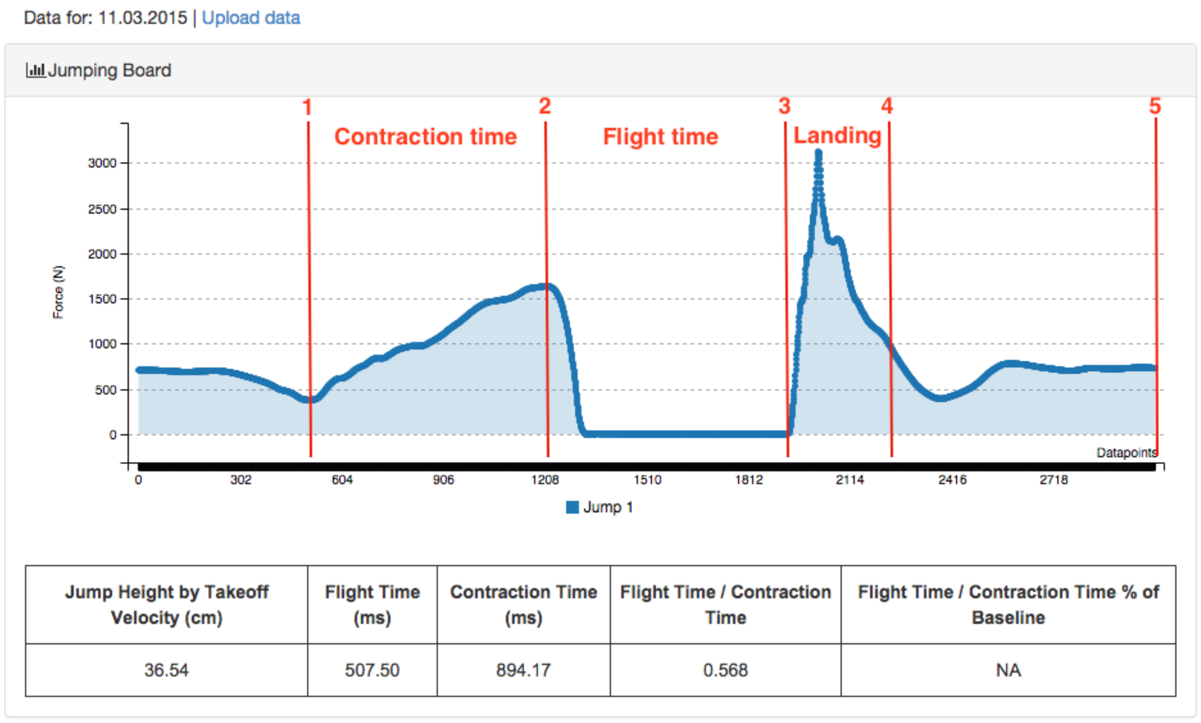
$$Time = \frac{v_{Final} - v_{Initial}}{Acceleration} \tag{8.4}$$

Based on the equations for GPE and KE, we could predict when the player would land, and the data after landing is practically useless since it does not show anything else than that the force on impact is disappearing. If we try to calculate the data of jump 1 in figure 8.1, we can see that the jump height by takeoff is at 30,9 centimeters and the weight is 74,31 kg. By using the equations for GPE and KE it gives us the understanding that we should cut after the highest value + the next 300 data points that takes the player to land from that point. In section 8.5.4, each row of jump data used 80 Kilobytes with this technique, we potentially store 67,8 Kilobytes. A revised calculation would show that a team would use 30*4*67,8/1024 = **7,954 Megabytes**, which equals **15,16%** less data stored each month.

$$\begin{aligned} \text{GPE} &= 74.31 * 9.81 * 0,309 \\ \text{KE} &= \frac{1}{2}74.31 * v^2 \\ \text{v} &= 2.46\text{ms/s} \\ \\ \text{t} &= \frac{2.46 - 0}{9.81} \\ \text{t} &= 0.25\text{(seconds)} * 1000 = 250\text{milliseconds} \\ \\ \text{Highest value in jump} &= 1664 \\ \text{Milliseconds to land} &= 250 * 1.2\text{(to convert from milliseconds to data points)} \\ \text{Ideal cut value} &= 1964 \end{aligned} \tag{8.5}$$

8.5.5 Visualization

The graph presented to the user is the highest jump recorded among the set of uploaded data for a specific date, and this data does not particularly say much, however, it shows a couple of variables the software calculates for us after each jump (see figure 8.11). The area between 1 and 2 in the graph is the contraction time; this is the time the player use to gather force before jumping. Between 2 and 3 is the flight time, during this time, the player is in mid-air which means no force on the board. The pointer at number 4 is the optimized cut method 2 (see section 8.5.4.2), while number 5 is cut method 1 (see section 8.5.4.1). Below the graph is the analyzed data from the HUR Labs software where the variables is calculated and exported.



Jump Height by Takeoff Velocity (cm)	Flight Time (ms)	Contraction Time (ms)	Flight Time / Contraction Time	Flight Time / Contraction Time % of Baseline
36.54	507.50	894.17	0.568	NA

Figure 8.11: Visualization of jumping data

8.6 Evaluation & Discussion

In this section, we evaluate the decisions taken when we implemented the Tesseract system. Objective data can be valuable information that can give a certain level of analysis, since everything we do can be analyzed. The type of data that can be collected is endless due to the fact that there is more than just physical state that can give us a better analysis. It is also possible to use physiological analysis to cross-reference, and to improve how a team should go forward when an event with high physiological load can have a big impact on a player. For example missing on an important penalty goal during the world championship.

Even though pmSys has only started to integrate objective data into the core of the system, the possible outcome from such an integration can be incredible huge in combination with subjective data. To be able to find out what type of impact objective data can provide a football team that applies pmSys, can only be done after running the program throughout the course of at at least one football season. But unfortunately, there is no correct answer to what it could mean or can do before letting it run for a period, and therefore also difficult to evaluate the usefulness of objective data.

8.6.1 Difference between objective data in Ohmage versus pmSys

In section 4.1.1.1, we discussed briefly the applications Ohmage has integrated to collect objective data from a users mobile device. The problem with these mobile applications is that both projects (AudioSens and SystemSens) are no longer maintained and should probably be classified as *deprecated*. These two applications are analyzing the surroundings of the player, and not the player's physical state. Because of this, pmSys is more suitable for monitoring their physical state and the fact that the system itself is custom made for a player's physical analysis.

The potential of the objective system can be extremely big, however, the system is not currently in use by the test teams. The system needs time to calibrate itself before it can be of any use, hence, the results of this system cannot be evaluated nor predicted at this time. It is important to note, that objective data cannot be used as the only input source for physical analysis. Raw data often have limited value, and the only way to make the data valuable is to process it through multiple interpretations where the data is correlated with other input sources. In pmSys case, subjective data is the supporting role of making the objective data relevant and valuable for analysis.

A user scenario where objective data can confirm the subjective data is for example when objective data can be used to analyze if a player has a decreasing performance during training and matches. By cross referencing the subjective RPE load with the sleep pattern of the player (see figure 8.6) and the total amount of steps of the previous days. It is critical that the muscles gets enough time to recover from the previous hard session, and with this data the medical staff can easier identify why the player has a decreasing performance.

8.7 Summary

In this chapter, we introduced pmSys-Tesseract, that has a lot of potential of what it can offer to the rest of pmSys, although it requires some time to both calibrate and retrieve enough data to provide proper analysis. The whole idea is centered behind objective input sources to support the subjective data from the players, hence the system can become a vital part of pmSys, but as for now, the system is just an extra feature with great potential for incredible analysis. This proof of concept is just the beginning, however, there exists a lot of areas where optimization has to be done to be as effective as it can become. The biggest challenge for pmSys-Tesseract is the amount of third party hardware it will support in the future, the more input sources the system has, the more resources has to be allocated to keep the system running. For each input source added to the system, the complexity of the system will increase drastically.

Chapter 9

Conclusion

9.1 Summary

In section 1.2, we defined the problem of creating a digital monitoring system that optimizes the collection, the storage of data, the data analyzation and the visualization of a player's health data. The system we created mainly supports subjective data through questionnaires, however, the system also supports objective data through third party hardware.

We have collaborated with NiH throughout this thesis to analyze the usefulness of a reporting and monitoring system in professional Norwegian football teams. As a result, we created a rich featured system for both the coach and the players under the name of **pmSys**. The main objective of pmSys, is to be an assisting tool for both coaches and the players by replacing the old registration and monitoring system i.e., pen and paper.

The pmSys-system consists of multiple subsystems created specifically to solve an important task within the system. We created a mobile application for data collection, named pmSys-app. The mobile application is available on both iOS and Android, where the users can answer surveys in just a matter of seconds compared to pen and paper. They can also monitor their own subjective performance through visualization of their own data; as a result, they can reflect upon their own effort and support them in improving themselves over time.

A web portal with analysis capabilities has been created specifically for the coaches. This is where they can monitor their players and get instant analysis of the subjective data through visualization of the survey answers collected. The idea is to detect early a player's physical state before they get injured, which is vital to their performance during training sessions and matches.

A one-way communication system between the coach and the players has also been implemented, to assist the coaches to remind the players to register their surveys. The coaches can send push notifications to a player's mobile device through the web portal, they can also set up automated push notifications to notify the players when to register surveys. Several user studies has been conducted with objective and subjective users on their experience with pmSys, and the results indicate that pmSys is an excellent tool where data collection and data analysis is a vital part of the system. The system has also received positive remarks from coaches regarding the usefulness of the web portal as an analysis tool.

Finally, we looked at the possibility of integrating third party hardware into pmSys for better analysis. A new system dedicated for collecting objective data has been created and named pmSys-Tesseract. Subjective data can provide the coaches with the perceived RPE load, however, objective data can monitor the RPE load on the body. By combining these two data types, the coach can enhance the analysis tool to see variables the naked eye cannot see, whereas the hardware can capture these invisible variables through sensors. The proof of concept shows the potential of objective data, however, the system has not been tested nor used by the teams. Therefore, no definite evaluation of the system can be presented.

9.2 Main Contributions

In this thesis, we have shown all the systems in our monitoring system, pmSys. The system is a collection of many subsystems created with monitoring of a football player's physical state in mind. pmSys is currently deployed for the Norwegian National and several Tippeliga teams. As a part of the NiH project agreement, we receive feedback from active players and coaches regarding the functionality of the system.

As a result, we have created a dedicated data collection application for the players on iOS and Android under the name pmSys-app (pmSys in the app stores). The mobile application received media attention early in the release phase on NRK's (Norsk rikskringkasting AS) web news [9]. The reporting process has been optimized for registering surveys quickly and effective within seconds, with less button clicks and a summary page for confirmation of the entered data. Furthermore, the mobile application has support for self-reminders, where the players can set up the reminders themselves. The coach can also remind the players through push notifications, which is an easier way to contact the players then sending text messages.

We have developed a monitoring and data analysis tool for the coaches as a web portal, pmSys-Trainer. The web portal provides the coaches tools in monitoring, assisting and analyzing of the players physical state. The web portal has tools for data visualization, push notifications to a player's mobile device and survey answer statistics. The idea of the web portal is to analyze data instantly, instead of feeding data into third party software in order to recieve the same results. This is timesaving and can help the coaches set up correct training sessions without injuring the players due to overtraining.

A good example of how effective the pmSys-app has been for collecting health data with push notification interaction through the pmSys-Trainer can be seen in section 5.6.4. Furthermore in section 5.1 one of the feedback we received was that the players wanted to answer the surveys every day but they simply forgot to do it because of no reminders was integrated in Ohmage (without setting it up themselves on iOS).

9.3 Future work

During the thesis we received a lot of feedback and requests for improvements and features on pmSys. In the next sections, a list of features and improvements we think could be done in the future will be listed.

9.3.1 Ohmage back-end

9.3.1.1 Update campaigns

It is not possible to update a campaign with new questions or values with the current Ohmage back-end. This is a major drawback, since once it is uploaded and active, the data is connected to the campaign. Hence, if a letter is wrong, the campaign has to be deleted with it all the registered survey responses.

9.3.1.2 Validation of hashed password

This feature is critical for all systems of pmSys, due to the fact that the REST API's can be accessed if they know what to send with the HTTPS requests. By adding this feature, the only way to retrieve data is by validation of the hashed password with the back-end.

9.3.1.3 Optimalized results

To reduce bandwidth and to increase the performance on both the pmSys-app and the pmSys-Trainer, the data has to be processed and optimized. If the data is returned pre-processed, then the web browser can focus on visualizing the data instead of matching the data before being able to visualize it. A prime example is survey responses, if the responses were returned with the username as key, and the survey types below it as key where the value is one survey response for each key, then the performance could potentially increase (see figure 9.1).

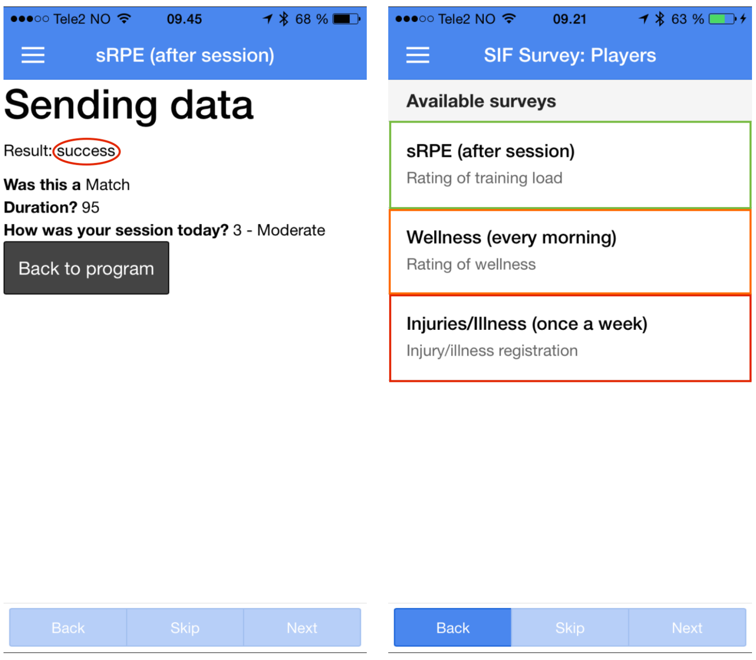
```
{
  thuc: {
    rpe: {
      0: [
        // Survey response for RPE
      ]
    },
    wellness: {
      0: [
        // Survey response for Wellness
      ]
    },
    injuries: {
      0: [
        // Survey response for Injuries
      ]
    }
  }
}
```

Figure 9.1: Example of new data format

9.3.2 pmSys-App

9.3.2.1 Color highlights of survey list

Players wish to know if they have answered a specific survey, by adding border color to indicate which surveys has been registered in case of duplicates (see figure 9.2(b)). The reason for this feature request, is because there is no indication except for the confirmation page after confirming on the summary page (see figure 9.2(a)). In the example, green means *"OK"*, orange means *"Not done today"* and red is *"Must answer today!"*, thus the color codes can be changed to something more intuitive later.



●●●○○ Tele2 NO 09.45 68 %

sRPE (after session)

Sending data

Result: success

Was this a Match

Duration? 95

How was your session today? 3 - Moderate

Back to program

●●●○○ Tele2 NO 09.21 63 %

SIF Survey: Players

Available surveys

sRPE (after session)
Rating of training load

Wellness (every morning)
Rating of wellness

Injuries/Illness (once a week)
Injury/illness registration

(a) Confirmation of sent survey (b) Color codes

Figure 9.2: Color highlights of survey status

9.3.2.2 Encrypted end-to-end chat between coach and player

The push notification allows the coach a one-way communication channel between the coach and the player. This feature can enable discussion between the two of them, to confirm the survey responses before further analysis of the physical state. This can be achieved by implementing (elliptic curve) Diffie-Hellman [121] with Perfect Forward Secrecy.

9.3.2.3 Language localization (i18n)

This feature allows the players to define their own translation over to their own native language (if they are not Norwegian or fluent in English). Furthermore, this feature can help pmSys become more international by supporting multiple languages such as French, Spanish, German etc.

9.3.2.4 Rewamp offline mode by using SQLite

As mentioned in section 5.6.2, SQLite might be a better suit for the pmSys-app. Instead of creating a large amount of files on the mobile device, the SQLite database creates a new BLOB for each survey response.

9.3.2.5 Security of the mobile application

It is possible to decipher the source code of the pmSys-app, a possible way to avoid this, is to obfuscate the source code before deploying the application. By using secrecy as a part of the design, known as *Security through obscurity* [122]. The data the pmSys-app creates is stored in plain text directly on the file system. Even though the data is not sensitive, it is important to secure every part of the application. There is a plugin created to secure files at runtime [123], hence increasing the complexity of the application for better security.

9.3.2.6 Add Windows Phone support

The pmSys-app supports iOS and Android but it does not support Windows Phone. The reason for this is due to the plugins used in the application does not have implemented support for Windows Phone. There are a few users of pmSys that uses Windows Phone, however, these users have to use our web solution to register their surveys until the pmSys-app is developed further.

9.3.3 pmSys-Trainer

9.3.3.1 Artificial Intelligence (AI) or Machine Learning

As of now, the coaches have to manually check for irregularities among their players. By developing AI for pmSys-Trainer, the coaches can be notified on their mobile device when the system detects that something is wrong with a player. AI can also be exchanged for machine learning, where the analysis is based on previous recorded data for more personal analysis of a player.

9.3.3.2 User mapping

In pmSys, all usernames are obfuscated by random generated letters and numbers in combination. Only the coach knows which player is behind a username, thus creating extra work for the coach. If two usernames are similar, the coaches might mix the users and give faulty personalized training. By creating a secure mapping of the users, the coaches can save a lot of time on skipping the name lookup phase.

9.3.3.3 Open up the web portal for the players

The web portal is restricted to the coaches of each team, in this way we can focus on the coach. In the future, it could be a good idea to let the players' receive a detailed analysis of themselves, without asking the coach for their own analysis. By doing this, the players can monitor their own performance.

9.3.4 pmSys-Tesseract

9.3.4.1 New visualization layout

Instead of showing data for a specific day at a time, the visualization can be extended to show over a period of time. For example sleep pattern for the last two weeks gives the coach a better understanding of a player's sleep pattern if the performance drops in between training days.

9.3.4.2 Storage of latest fetched data from Fitbit

As of now, the pmSys-Tesseract system does not store the data retrieved from Fitbit. The reason for this is because data from Fitbit is always available, however, the idea of storing the data for faster lookup might speed up the process.

9.3.4.3 More third party hardware

The system currently supports HUR Labs jumping board and Fitbit, however, there are many other hardware with other type of monitoring capabilities. By adding new technology, the system will get more complex, but that is needed for better analysis and it might help for better performance.

9.3.4.4 Data warehouse

Data warehouse (DW or DWH) [124] is a system used for reporting and data analysis. The DWH is a centralized data point where it integrates multiple data sources before it analyzes the data. By changing into this, the data can be processed into data models, hence sorting the data where it belongs which enhances the analysis capabilities and performance of the system (see figure 9.3).

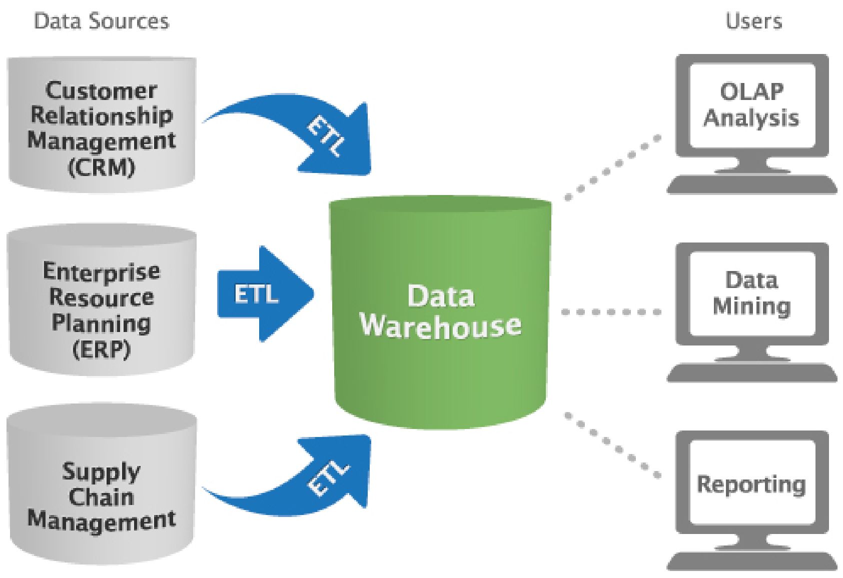


Figure 9.3: Example of how a data warehouse works [125]

Appendix A

Accessing the source code

The source code for the pmSys project is divided into three repositories. Access to the repositories can be given upon request.

A.1 pmSys-app

`https://bitbucket.org/nktteam/pms-app`

A.2 pmSys-trainer

`https://bitbucket.org/nktteam/pms-trainer`

A.3 pmSys-push

`https://bitbucket.org/nktteam/pms-pushserver`

Appendix B

User Surveys

B.1 pmSys vs. Ohmage

Brukerundersøkelse

Fra en skala fra 1 til 5 (helhetsvurdering), hvordan vil du rangere...

*Må fylles ut

1. **Hvilken bakgrunn har du?**
   Studieretning / Yrke

   ....

2. **Driver du aktivt med sport i fritiden?**
   Merk av for alt som passer
   - ☐ Ja
   - ☐ Nei

3. **Brukervennligheten til pmSys? ***
   Markér bare én oval.
   - ◯ Dårlig
   - ◯ Ok
   - ◯ Bra
   - ◯ Veldig bra
   - ◯ Utmerket

4. **Brukervennligheten til Ohmage? ***
   Markér bare én oval.
   - ◯ Dårlig
   - ◯ Ok
   - ◯ Bra
   - ◯ Veldig bra
   - ◯ Utmerket

5. **Designet (grensesnittet) til pmSys?**
   Markér bare én oval.
   - ◯ Dårlig
   - ◯ Ok
   - ◯ Bra
   - ◯ Veldig bra
   - ◯ Utmerket

6. **Designet (grensesnittet) til Ohmage?**
Markér bare én oval.

- Dårlig
- Ok
- Bra
- Veldig bra
- Utmerket

7. **Navigasjonen i pmSys?**
Hvordan er det å manøvrere i applikasjonen?
Markér bare én oval.

- Dårlig
- Ok
- Bra
- Veldig bra
- Utmerket

8. **Navigasjonen i Ohmage?**
Hvordan er det å manøvrere i applikasjonen?
Markér bare én oval.

- Dårlig
- Ok
- Bra
- Veldig bra
- Utmerket

9. **Hvordan presenteres innholdet i pmSys?**
Markér bare én oval.

- Dårlig
- Ok
- Bra
- Veldig bra
- Utmerket

10. **Hvordan presenteres innholdet i Ohmage?**
Markér bare én oval.

- Dårlig
- Ok
- Bra
- Veldig bra
- Utmerket

11. **Hvilken applikasjon foretrekker du?**
Merk av for alt som passer

☐ Ohmage
☐ pmSys

12. **Har du kommentarer til pmSys eller Ohmage?** *

....

....

....

....

....

B.2 Rating of pmSys

Brukerundersøkelse

PmSys brukerundersøkelse

*Må fylles ut

1. **Hva synes du om prosessen for RPE, wellness og injury rapportering?** *
   Markér bare én oval.

   ◯ Dårlig
   ◯ Ok
   ◯ Bra
   ◯ Veldig bra
   ◯ Utmerket

2. **Hvilken rapporteringsverktøy foretrekker du?**
   Markér bare én oval.

   ◯ Penn og papir
   ◯ Excel
   ◯ Web survey
   ◯ Mobil applikasjon
   ◯ Annet

3. **Hvis annet, hvilke?**

   ....

4. **Sparer pmSys deg for tid ved rapportering?**
   Er pmSys raskere enn andre verktøy du har brukt?
   Markér bare én oval.

   ◯ Ja
   ◯ Nei

Fra en skala 1 (dårlig) til 5 (utmerket), hvordan vil du rangere (helhetsvurdering)...

5. **Brukervennligheten til pmSys?** *
Markér bare én oval.

- Dårlig
- Ok
- Bra
- Veldig bra
- Utmerket

6. **Designet (grensesnittet) til pmSys?**
Markér bare én oval.

- Dårlig
- Ok
- Bra
- Veldig bra
- Utmerket

7. **Navigasjonen i pmSys?**
Hvordan er det å manøvrere i applikasjonen?
Markér bare én oval.

- Dårlig
- Ok
- Bra
- Veldig bra
- Utmerket

8. **Hvordan presenteres innholdet i pmSys?**
Markér bare én oval.

- Dårlig
- Ok
- Bra
- Veldig bra
- Utmerket

9. **Nyttigheten av pmSys sine funksjoner?**
Påminnelser for rapportering? Visualisering?
Markér bare én oval.

- Dårlig
- Ok
- Bra
- Veldig bra
- Utmerket

10. **Utbytte ved bruk av pmSys?**
Har du fått noe igjen av å bruke pmSys?
Markér bare én oval.

- Dårlig
- Ok
- Bra
- Veldig bra
- Utmerket

11. **Har du andre kommentarer til pmSys?** *
Forbedringer / Ønskede funksjonalitet / Ris / Ros

Appendix C

Surveys in pmSys

C.1 RPE Survey

Rating of Session RPE

When?

Within 15 minutes after each training session and match. (typically in the dressing room). If that is not possible, do it as soon as possible. If you want to register the session you had yesterday, please check for "yesterday" in the check box on the last page of the registration.

Registration contents

Please answer the following questions:

1. Was this a (match, team session or a individual session)?
2. Which type of session (football session, endurance session, strength/speed session or other)?
3. Duration? (number of minutes)
4. How was your session today (0-10)?

What kind of information do we get?

Session RPE provide information on intensity, duration and frequency of your training sessions and matches. By tracking these data over time, the coach can supervise the total training load and the variation in training load.

Clarifications

- Dictionary
  - Match: official match or friendly match.
  - Team session: Training session that include the team or part of the team
  - Individual session:
  - Football session: a session on the field which includes the ball
  - Endurance session: For example a running, cycling or swimming session ment to improve endurance
  - Strength training: Strength training or core training (typically in the gym)
  - Other: Other activity that doesn't fit into the other (e.g. playing tennis, yoga)
- Rate how intense you experienced the session
- The rating should be an average of the whole session (fig.1). Take into account periods of high intensity running and periods of standing still.
- 10 is the highest exertion that you can imagine. Imagine pushing yourself running a 3000 m test, without any break.
- 0 is equivalent to rest, and should not be used in combination with training.
- It is exhausting to sprint, having a high heart rate, fast breathing, but also to tackle, jump and duel. How exhausted you feel in your muscles and mentally is also a part of the RPE.
- A match is typically rated 6-7-8, but could also be higher or lower.
- A strength training session is typically 2-3-4-5 (because of much pauses)

Minutes:	**5**	**10**	**15**	**20**	**25**	**30**	**35**	**40**	**45**	**50**	**55**	**60**	**65**	**70**	**75**	**80**	**85**	**90**	**Average**
RPE:	8	5	6	3	6	8	5	6	9	8	7	6	4	6	4	6	7	10	6,5

Fig. 1. *Example from a 90 minutes session. The assessment should reflect an average of the whole session. Imagine that you rate every 5th minute of the session and then calculate the average value.*

Rating of Session RPE

Visualization

- Training load: is the RPE score multiplied by the duration of the session. High training load occurs either by high RPE score, high duration or both.
- Weekly load: is the average training load the last 7 days.
- Monotony describes the variation in training load over the last 7 days. High monotony means low variation in training load.
- Strain is the average weekly load multiplied by the monotony. High strain means that the weekly load is high combined with low variation in training load. High strain means less time for recovery and is associated with overtraining or injuries.

Session RPE - rate of perceived exertion	
Rating	Explanation
0	Rest
1	Very, very easy
2	Easy
3	Moderate
4	Somewhat hard
5	Hard
6	
7	Very hard
8	
9	
10	Maximal

C.2 Wellness survey

Rating of wellness

When?

Please rate your wellness every morning, 7 days a week. The rating must take place after getting out of bed, but before training. For example before or after breakfast or in the dressing room before the session.

Registration contents

1. "Readiness to play"
2. "Fatigue"
3. "Sleep Quality"
4. "Hours of Sleep"
5. "General Muscle Soreness"
6. "Stress Levels"
7. "Mood"

What kind of information do we get?

Wellness indicates how well the players overcome or responds to the training load and how well he recover? A lower score than normal over time may indicate a higher risk of overuse injuries.

Clarifications

- Rate as best as you can according to the questions
- On the scale, 3 is normal, 1 er "worst" and 5 "best".
- "Readiness to play" has a scale from 1-10, where 1 is "not ready at all" and 10 is "maximally ready".
- Dictionary
  - *Fatigue:* means tiredness resulting from mental or physical exertion or illness.
  - *Sleep quality:* means "how was your sleep last night?"
  - Hours of sleep: "how many hours did you sleep last night?"
  - *General muscle soreness:* means general soreness in the musculature (especially in the legs)
  - *Stress levels* means a state of mental or emotional strain or tension resulting from adverse or demanding circumstances
  - *Mood* means emotionally state of mind
  - *Readyness to play:* means "how ready (physically and mentally) are you to play if there is a match today/tonight?"

Visualization

On the visualization page, you can view your latest rating of fatigue, sleep, soreness, stress and mood. You can also view your ratings for the last 30 days. A thick red line represents the average of the five wellness parameters.

	5	4	3	2	1
Fatigue	Very fresh	Fresh	Normal	More tired than normal	Always tired
Sleep quality	Very restful	Good	Difficulty falling asleep	Restless sleep	Insomnia
Hours of sleep	-	-	-	-	-
General muscle soreness	Feeling great	Feeling good	Normal	Increase in soreness/tightness	Very sore
Stress levels	Very relaxed	Relaxed	Normal	Feeling stressed	Highly stressed
Mood	Very positiv mood	A generally good mood	Less interested in others and/or activities than usual	Snappiness at team-mates, family and co-workers	Highly annoyed/ irritable/down

C.3 Injury survey

Injury registration

When?
Once a week on a fixed day.

Registration contents
Part 1:Please answer the following questions as best as you can

1. Have you had any difficulties participating in normal training and competition due to injury, illness or other health problems during the past week?
2. To what extent have you reduced your training volume due to injury, illness or other health problems during the past week?
3. To what extent has injury, illness or other health problems affected your performance during the past week?
4. To what extent have you experienced symptoms/health complaints during the past week?

Part 2: If you have experienced injuries/illnesses, you will continue with these questions

1. Is the health problem an injury or illness?
2. Select the area that best describes the injury / illness?
3. Please state the number of days over the past 7-day period that you have had to completely miss training or competition du to this problem?
4. Is this the first time you have reported this injury?
5. Have you reported the problem to the medical device?
6. Do you have more injuries to report?

What kind of information do we get?
The injury registration systemize information about acute injury, overuse injuries and health problems. The registration may detect health problems and symptoms before it develops into an overuse injury. It also record small injuries/illness that are often overseen in traditional injury registration

Clarifications
It is important that you register all your health problems every week, even if you have registered the same problem before, or if you are receiving treatment for it. If you have several injuries/ illnesses within one week, be sure to record all of them by going through the registration several times. Record the most serious injury/illness first.

Your team physician/physiotherapist/fitness coach will receive a message when you record and injury. It is important to emphasize that this system does not replace your regular contact with the medical team. Please continue to make direct contact with the team physician or physiotherapist when you need it.

Visualization
The visualization indicates a severity score of the injury/illness to be used in research. Each of the questions (1-4) scores 0-25, and the larger the sum is, the larger the severity score is. It is important to emphasize that only your team physician or physiotherapist can diagnose and decide how seriously your injury or illness is.

Bibliography

[1] Belastningsovervåking i fotball. http://www.nih.no/forskning/prosjektarkivet1/forskningsprosjekter-ved-nih/belastningsovervaking-i-fotball/. Accessed: January 20th 2015.

[2] Ohmage MWF. https://oit.ucla.edu/mobile-web-strategy/ohmage-mwf. Accessed: February 15th 2015.

[3] Ohmage. http://www.ohmage.org/. Accessed: January 4th 2015.

[4] Cong-Nguyen Nguyen. Implementation of a digital player monitoring system: pmsys. Master's thesis, University of Oslo, 2015.

[5] Kennet Khang Vuong. Pmsys: a system for sports athlete load, wellness and injury monitoring. Master's thesis, University of Oslo, 2015.

[6] Peter J. Denning, Douglas E. Comer, David Gries, Michael C. Mulder, Allen Tucker, A. Joe Turner, and Paul R. Young. Computing as a discipline. *ACM*, 32, 1989.

[7] Scrum (software development). http://en.wikipedia.org/wiki/Scrum_%28software_development%29. Accessed: February 17th 2015.

[8] PMS Project Management. https://trello.com/b/M1aLlssW/pms-project-management. Accessed: January 4th 2015.

[9] Ny app skal sikre EM-suksess. http://www.nrk.no/troms/ny-app-skal-sikre-em-suksess-1.12048486. Accessed: February 16th 2015.

[10] Carl Foster, Jessica A. Florhaug, Jodi Franklin, Lori Gottschall, Lauri A. Hrovatin, Suzanne Parker, Pamela Doleshal, and Christopher Dodge. A new approach to monitoring exercise training. *Journal of Strength and Conditioning Research*, 15(1):109–115, 2001.

[11] Kid-Edgar Sørensen. Ruoksat: A system for capturing, persisting and presenting the digital footprint of soccer knowledge and expertise. Master's thesis, University of Tromø, 2013.

[12] Likert Scale. http://en.wikipedia.org/wiki/Likert_scale. Accessed: March 15th 2015.

[13] Benjamin Clarsen, Ola Rønsen, Grethe Myklebust, Tonje Wåle Flørenes, and Roald Bahr. The

oslo sports trauma research center questionnaire on health problems: a new approach to prospective monitoring of illness and injury in elite athletes. *BJSM Online First*, 2013.

[14] Benjamin Clarsen, Grethe Myklebust, and Roald Bahr. Development and validation of a new method for the registration of overuse injuries in sports injury epidemiology: the oslo sports trauma research centre (ostrc) overuse injury questionnaire. *BJSM Online First*, page 3, 2012.

[15] R. Meeusen, M. Duclos, C. Foster, M. Gleeson A. Fry, D. Nieman, J. Raglin, G. Rietjens, and J. Steinacker. Prevention, diagnosis, and treatment of the overtraining syndrome: joint consensus statement of the european college of sport science and the american college of sports medicine. *Med Sci Sports Exerc*, 45:186–205, 2013.

[16] Aaron J. Coutts, Karim Chamari, and Ermanno Rampinini Franco M. Impellizzeri. Monitoring training in soccer: Measuring and periodising training. 2008.

[17] J. Burke, D. Estrin, M. Hansen, A. Parker, N. Ramanathan, S. Reddy, and M. B. Srivastava. Participatory sensing. *ACM*, 2006.

[18] Andel som har smarttelefon. `http://medienorge.uib.no/statistikk/aspekt/tilgang-og-bruk/379`. Accessed: March 15th 2015.

[19] Open mHealth About. `http://www.openmhealth.org/about/`. Accessed: February 14th 2015.

[20] Deborah Estrin and Ida Sim. Open mhealth architecture: An engine for health care innovation. *Science*, 330:759–760, 2010.

[21] One In Every 5 People In The World Own A Smartphone, One In Every 17 Own A Tablet. `http://www.businessinsider.com/smartphone-and-tablet-penetration-2013-10`. Accessed: March 16th 2015.

[22] Silo. `http://www.oxforddictionaries.com/us/definition/american_english/silo`. Accessed March 17th 2015.

[23] H. Tangmunarunkit, C. K. Hsieh, J. Jenkins, C. Ketcham, J. Selsky, F. Alquaddoomi, D. George, J. Kang, Z. Khalapyan, B. Longstaff, S. Nolen, T. Pham, J. Ooms, N. Ramanathan, and D. Estrin. Ohmage: A general and extensible end-to-end participatory sensing platform. *UCL Computer Science Technical Report*, 2014.

[24] DHIS2. `https://www.dhis2.org/`. Accessed: February 14th 2015.

[25] JSONP. `http://en.wikipedia.org/wiki/JSONP`. Accessed: March 25th 2015.

[26] Use of DHIS 2 in HIS: data collection, processing, interpretation, and analysis. `https://www.dhis2.org/doc/snapshot/en/user/html/ch01s03.html`. Accessed: February 16th 2015.

[27] Milan Lab. `http://www.acmilan.com/en/club/milan_lab`. Accessed: March 28th 2015.

[28] Inside AC's MilanLab. `http:`

//www.meerssemanlab.com/Meersseman_Lab/Inside_AC_Milan_Lab.html. Accessed April 5th 2015.

[29] Jeanne G. Harris, Elizabeth Craig, and David A. Light. Accenture research report. 2010.

[30] DHIS2 Appstore. https://www.dhis2.org/appstore. Accessed: February 14th 2015.

[31] Internet of Things. http://en.wikipedia.org/wiki/Internet_of_Things. Accessed: April 2nd 2015.

[32] Node.js is taking over the Enterprise – whether you like it or not. https://www.centurylinkcloud.com/blog/post/node-js-is-taking-over-the-enterprise-whether-you-like-it-or-not/. Accessed: March 28th 2015.

[33] Stop Fighting Node.Js In The Enterprise. http://www.wintellect.com/devcenter/dbanister/stop-fighting-node-js-in-the-enterprise. Accessed: March 28th 2015.

[34] NodeJS. https://nodejs.org/. Accessed: January 20th 2015.

[35] MongoDB. https://www.mongodb.org/. Accessed: January 20th 2015.

[36] Create, read, update and delete. http://en.wikipedia.org/wiki/Create,_read,_update_and_delete. Accessed: February 16th 2015.

[37] JSON. http://en.wikipedia.org/wiki/JSON. Accessed: February 15th 2015.

[38] About Users, Classes and Campaigns. https://github.com/ohmage/server/wiki/About-Users,-Classes-and-Campaigns. Accessed: February 15th 2015.

[39] MWF Mobile Web Framework. http://mwf.ucla.edu/. Accessed: March 16th 2015.

[40] Home. https://github.com/ucla/mwf/wiki. Accessed: March 16th 2015.

[41] General: Principles and Strategy. https://github.com/ucla/mwf/wiki/General%3A-Principles-and-Strategy. Accessed: March 16th 2015.

[42] OpenCPU. https://www.opencpu.org/. Accessed: March 15th 2015.

[43] AudioSens. https://github.com/cens/audioSens. Accessed: March 22th 2015.

[44] SystemSens. https://github.com/falaki/SystemSens. Accessed: March 22th 2015.

[45] Lov om behandling av personopplysninger (personopplysningsloven). https://lovdata.no/dokument/NL/lov/2000-04-14-31. Accessed: March 28th 2015.

[46] Edward Snowden.

http://www.biography.com/people/edward-snowden-21262897. Accessed: April 24th 2015.

[47] National Security Agency. https://www.nsa.gov/. Accessed: April 24th 2015.

[48] Wikipedia: National Security Agency. http://en.wikipedia.org/wiki/National_Security_Agency. Accessed: April 24th 2015.

[49] Heartbleed. http://heartbleed.com/. Accessed: April 24th 2015.

[50] Everything you need to know about the Heartbleed SSL bug. http://www.troyhunt.com/2014/04/everything-you-need-to-know-about.html. Accessed: April 24th 2015.

[51] Keymetrics - NodeJS monitoring. https://keymetrics.io/. Accessed: March 22th 2015.

[52] Act of 14 April 2000 No. 31 relating to the processing of personal data (Personal Data Act). https://www.regjeringen.no/en/topics/health-and-care/public-health/Act-of-18-May-2001-No-24-on-Personal-Health-Data-Filing-Systems-and-the-Processing-of-Personal-Health-Data-Personal-Health-Data-Filing-System-Act-/id224129/. Accessed: February 18th 2015.

[53] Personal Data Regulation. https://www.datatilsynet.no/English/Regulations/Personal-Data-Act1/. Accessed: February 18th 2015.

[54] User Manipulation. https://github.com/ohmage/server/wiki/User-Manipulation. Accessed: March 22th 2015.

[55] Ajax (programming). http://en.wikipedia.org/wiki/Ajax_%28programming%29. Accessed: March 28th 2015.

[56] Michael S. Mikowski and Josh C. Powell. *Single Page Web Applications*. B and W, 2013.

[57] An Intro Into Single Page Applications (SPA). http://blog.4psa.com/an-intro-into-single-page-applications-spa/. Accessed: March 18th 2015.

[58] Trygve Reenskaug. Thing-model-view-editor: an example from a planningsystem. *Xerox PARC*, 1979.

[59] What is a hybrid mobile app. http://developer.telerik.com/featured/what-is-a-hybrid-mobile-app/. Accessed: March 18th 2015.

[60] AngularJS. https://angularjs.org/. Accessed: April 24th 2015.

[61] Google+ Angular. https://plus.google.com/+AngularJS/posts/aZNVhj355G2. Accessed: April 24th 2015.

[62] Ionic Framework. http://ionicframework.com/. Accessed: February 14th 2015.

[63] Collection Repeat: Estimate, Iterate, Improve. http://blog.ionic.io/collection-repeat-iteration-two/. Accessed: March 25th 2015.

[64] x2js. https://code.google.com/p/x2js/. Accessed: March 28th 2015.

[65] Survey Upload. https://github.com/ohmage/server/wiki/Survey-Manipulation#surveyUpload. Accessed: March 24th 2015.

[66] NVD3. http://nvd3.org/. Accessed: February 16th 2015.

[67] cordova-plugin-local-notifications. https://github.com/katzer/cordova-plugin-local-notifications/. Accessed: March 24th 2015.

[68] cordova-plugin-file. https://github.com/apache/cordova-plugin-file. Accessed: March 24th 2015.

[69] The App Life Cycle. https://developer.apple.com/library/ios/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/TheAppLifeCycle/TheAppLifeCycle.html. Accessed: March 24th 2015.

[70] Background Execution. https://developer.apple.com/library/prerelease/ios/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/BackgroundExecution/BackgroundExecution.html. Accessed: March 24th 2015.

[71] App Store Review guidelines. https://developer.apple.com/app-store/review/guidelines/. Accessed: February 15th 2015.

[72] Certificates. https://developer.apple.com/support/technical/certificates/. Accessed: February 15th 2015.

[73] iOS Developer Enterprise Program. https://developer.apple.com/programs/ios/enterprise/. Accessed: April 28th 2015.

[74] Google Play. https://play.google.com/store. Accessed: February 15th 2015.

[75] Uncle Ben citation. http://en.wikipedia.org/wiki/Uncle_Ben. Accessed: February 16th 2015.

[76] Mobile: Native Apps, Web apps, and Hybrid Apps. http://www.nngroup.com/articles/mobile-native-apps/. Accessed: April 26th 2015.

[77] Mark Zuckerberg: Our Biggest Mistake Was Betting Too Much On HTML5. http://techcrunch.com/2012/09/11/mark-zuckerberg-our-biggest-mistake-with-mobile-was-betting-too-much-on-html5/. Accessed: April 26th 2015.

[78] UIWebView. https://developer.apple.com/library/ios/documentation/

UIKit/Reference/UIWebView_Class/. Accessed: April 26th 2015.

[79] Just In Time Compilation. http://en.wikipedia.org/wiki/Just-in-time_compilation. Accessed: March 25th 2015.

[80] WKWebView. https://developer.apple.com/library/ios/documentation/WebKit/Reference/WKWebView_Ref/. Accessed: April 26th 2015.

[81] WKWebView. https://github.com/Telerik-Verified-Plugins/WKWebView. Accessed: April 26th 2015.

[82] SQLite Homepage. https://www.sqlite.org/. Accessed: March 28th 2015.

[83] Comma-separated values. http://en.wikipedia.org/wiki/Comma-separated_values. Accessed: May 4th 2015.

[84] Express. http://expressjs.com/. Accessed: February 15th 2015.

[85] Nunjucks. http://mozilla.github.io/nunjucks/. Accessed: February 15th 2015.

[86] Redis. http://redis.io/. Accessed: February 16th 2015.

[87] How we made GitHub fast. https://github.com/blog/530-how-we-made-github-fast. Accessed: February 16th 2015.

[88] Bootstrap. http://getbootstrap.com/. Accessed: March 28th 2015.

[89] SB Admin. http://startbootstrap.com/template-overviews/sb-admin/. Accessed: April 27th 2015.

[90] C3.js. http://c3js.org/. Accessed: March 25th 2015.

[91] Github: Performance issue on graphs with more than 500 data points. https://github.com/masayuki0812/c3/issues/172#issuecomment-41130654. Accessed: April 26th 2015.

[92] Apple Push Notification Service. https://developer.apple.com/library/ios/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/Chapters/ApplePushService.html. Accessed: March 24th 2015.

[93] WWDC 2014 Session Videos. https://developer.apple.com/videos/wwdc/2014/. Accessed: April 27th 2015.

[94] node-apn. https://github.com/argon/node-apn. Accessed: January 16th 2015.

[95] Projects, Applications, and Companies Using Node apn. https://github.com/argon/node-apn/wiki/Projects,-Applications,-and-Companies-Using-Node-apn. Accessed: January 16th 2015.

[96] Google Cloud Messaging.
http://developer.android.com/google/gcm/gcm.html. Accessed: January 16th 2015.

[97] node-gcm. https://github.com/ToothlessGear/node-gcm. Accessed: January 16th 2015.

[98] C2DM. https://developers.google.com/android/c2dm/. Accessed: January 16th 2015.

[99] MySQL. http://www.mysql.com/. Accessed: January 14th 2015.

[100] Cron. http://en.wikipedia.org/wiki/Cron. Accessed: April 18th 2015.

[101] node-cron. https://github.com/ncb000gt/node-cron. Accessed: January 16th 2015.

[102] Tesseract. http://marvelcinematicuniverse.wikia.com/wiki/Tesseract. Accessed: March 28th 2015.

[103] Montoye HJ, Washburn R, Servais S, Ertl A, Webster JG, and Nagle FJ. Estimation of energy expenditure by a portable accelerometer. *Med. Sci. Sports Exerc.*, 1983.

[104] Schrödingers katt: Fotball.
http://tv.nrk.no/serie/schrodingers-katt/DMPV73000915/09-04-2015. Accessed: April 16th 2015.

[105] HUR Labs Force Platform (FP4).
http://www.hurlabs.com/tuotteet/hyppytestaus/force-platform-fp4. Accessed: April 27th 2015.

[106] Big Data. http://en.wikipedia.org/wiki/Big_data. Accessed: April 27th 2015.

[107] Apache Hadoop. http://hadoop.apache.org/. Accessed: May 7th 2015.

[108] Grid Grain. http://www.gridgain.com/. Accessed: May 7th 2015.

[109] High Performance Computing Cluster. http://hpccsystems.com/. Accessed: May 7th 2015.

[110] HealthKit. https://developer.apple.com/videos/wwdc/2014/#203. Accessed: April 28th 2015.

[111] Bekk øver på IoT-utvikling.
http://www.digi.no/for_utviklere/2015/05/07/bekk-over-pa-iot-utvikling. Accessed: May 7th 2015.

[112] Stuart J. Cormack, Robert U. Newton, Michael R. McGuigan, and Tim L.A. Doyle. Reliability of measures obtained during single and repeated countermovement jumps. *International Journal of Sports Physiology and Performance*, pages 131–144, 2008.

[113] Fitbit. https://www.fitbit.com/. Accessed: May 2nd 2015.

[114] OAuth. http://oauth.net/. Accessed: April 28th 2015.

[115] Baseline (medicine). http://en.wikipedia.org/wiki/Baseline_(medicine). Accessed April 27th 2015.

[116] Tab-separated values. http://en.wikipedia.org/wiki/Tab-separated_values. Accessed: May 4th 2015.

[117] Travis Ficklin, Robin Lund, and Megan Schipper. A comparison of jump height, takeoff velocities, and blocking coverage in the swing and traditional volleyball blocking techniques. *J Sports Sci Med*, 2014.

[118] 10.1.10.2 The utf8 Character Set (3-Byte UTF-8 Unicode Encoding). http://dev.mysql.com/doc/refman/5.0/en/charset-unicode-utf8.html. Accessed: April 29th 2015.

[119] Potential Energy. http://en.wikipedia.org/wiki/Potential_energy. Accessed: March 28th 2015.

[120] Kinetic Energy. http://en.wikipedia.org/wiki/Kinetic_energy. Accessed: March 28th 2015.

[121] Diffie Hellman key exchange. http://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange. Accessed: May 7th 2015.

[122] Security through obscurity. http://en.wikipedia.org/wiki/Security_through_obscurity. Accessed: May 9th 2015.

[123] Cordova safe. https://github.com/disusered/cordova-safe. Accessed: May 9th 2015.

[124] Data warehouse. http://en.wikipedia.org/wiki/Data_warehouse. Accessed: May 9th 2015.

[125] Data warehouse image. http://www.stratebi.com/datawarehouse. Accessed: May 13th 2015.