UiO : Department of Informatics
University of Oslo

PmSys: a monitoring system for sports athlete load, wellness & injury monitoring

Player monitoring system

Kennet Vuong

Master's Thesis Spring 2015

PmSys: a monitoring system for sports athlete load, wellness & injury monitoring

Kennet Vuong

Abstract

Most football players, mostly young talents, end their bright future and football career, due to injuries and overloads. A player's performance during a game might be affected by various factors, such as muscle soreness, fatigue, mood or sleep quality. By monitoring a player, we may be able to predict and detect a possible injury, or prevent a reduced performance in a game. The most common monitoring tools used in sports context today, are still pen and paper.

In this thesis, we present an implemented digitized monitoring system, where we are monitoring a player's load, wellness, and injury through questionnaires. The system has made it easier to collect, process, store, analyze and present data using modern technologies. We have used a mobile application to collect data, the application also provides useful visualization feedback, thus helps a player to monitor them self. The system also has a web portal, where a coach can analyze, evaluate and monitor players captured data. This player monitoring system (pmSys) can help sports clubs and athletes to ease the process of manually capturing and processing data, and rather have focus on monitoring and analyzing. The Norwegian national team and the Norwegian top division (Tippeligaen) used this system to monitor their players. We believe this system can help sport athletes to reduce the number of overloads, injuries and reduced performance in a game for the future.

Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank my supervisor Pål Halvorsen for your great advice, ideas and input throughout this thesis. Your overwhelming feedbacks, encouragement and commitment have been truly inspiring, and I am grateful for the opportunities you have given me, to work on an exciting project.

Further I would like to thank Thuc Hoang and Cong Nguyen Nguyen for being great collaborators and this thesis would not been done without your creative ideas, great discussions and hard working. I personally want to thank both of you for the support, motivation and encouragement throughout this thesis.

I would also like to thank Håvard Wiig and Thor Einar Andersen from Norwegian school of sport sciences, for giving me the opportunity to combine my thesis with your research, in addition, provided great feedback and discussions. I also want to thank everyone that participated in the user study and everyone that have been testing and using this system throughout this thesis.

Next, I would like to thank Håvard Johansen, Dag Johansen and Svein Arne Pettersen from University of Tromsø, and Carsten Griwodz and Håkon Kvale Stensland from Simula Research Laboratory, for your great discussions and inputs throughout this thesis.

Finally, I would like to thank my friends and family for their support and encouraging words. A special thanks goes to my girlfriend, Lina Mai Nguyen, who have been a significant support to me throughout this thesis, I could not finish this thesis without you.

Oslo, May 18, 2015
Kennet Vuong

Chapter 1

Introduction

1.1 Background

By monitoring a person over period, we can learn about the person's unusual behaviors and habits. A method for monitoring is to observe and capture relevant data for the person's activity. Then, process the captured data before we evaluate and analyze it. A common approach to such research today, is to use pen and paper to gather information. For example, observe and take notes with pen and paper, or perform reporting through questionnaires. Regardless, the data must be collected, processed and analyzed afterwards. This process is not only time demanding, but also impractical in a long run.

Norwegian school of sports science (NIH) is performing a research study about monitoring load and fatigue in Norwegian top football [13]. Here they want to find the usefulness of subjective and objective tools to monitor football players in professional football. Furthermore, study the relation between a player's load, fatigue, physical performance and injury during a game, but also after and before a game. With help from these monitoring tools, they want to see if it is possible to predict an injury or a reduced performance for a player.

The Norwegian top division (Tippeligaen) consists of 16 professional football teams [14]. Each team has at least 20 players [15], which mean we have minimum 320 (20*16) football players in Tippeligaen. From a statistical report in 2012 by the Norwegian Football Federation (NFF), it is registered over 28 000 football teams in Norway [16]. From a non-technical perspective in football context, pen and paper is still the primary tool and most used approach for data collecting. Monitoring over *560 000* (28000*20) football players with pen and paper, is nearly impossible to organize and analyze reported data. Alternatively, consider web survey services, such as Google Forms[1] to collect data through questionnaires. Regardless of these two methods, the data must be manually processed and analyzed, before presented and evaluated.

With use of modern technologies, monitoring becomes more efficient and time saving. We can for example, have a mobile phone to collect data, and then, automatically send the data to a distributed server, and the server automatically processes the captured data and stores it securely. A web application can fetch the data from the distributed server, graphically presenting and visualizing the captured data with charts. This envisioned system does not only simplifies the process of monitoring sports athletes, but saves time and let users focus on analyzing and evaluating collected data. It is also possible extend the system to interpret and analyze captured data by itself. For example, implement machine learning, and let the system tell when a player is vulnerable to injuries, or have reduced performance in the next games.

In this thesis, we investigate and research an implementation of such monitoring system by self-report of subjective data. We will work in close cooperation with PhD student, Håvard Wiig and Dr. Thor Einar Andersen from NIH, to see if our envisioned system can support their research project.

[1] Google Forms - `https://www.google.com/forms/about`

1.2 Problem definition

Monitoring individuals in sport context with pen and paper involves not only data collecting, but also data processing and presenting. Use of pen and paper is not only time demanding and impractical, but make the user spend more time on processing data, than analyzing and evaluating, which the latter is the purpose of monitoring. With modern technology and use of internet, we can automate most of the monitoring tasks. In addition, ease the data collection process, but also make it fast and effective.

In this thesis, we therefore investigate the question whether we can design and develop a system that can effectively ease the process of data collecting, processing, analyzing and presenting for monitoring. Main goal will be to design a digitized approach to self-reporting and monitoring. We envision and aim for a *Player Monitoring System (pmSys)*, and we will evaluate the usefulness of existing systems for pmSys. As last in this thesis, we want to find effectiveness of deploying the system in cloud, and compare cloud hosting towards self-hosting.

Since, NIH already has their method for collecting objective data. Thus, we want to focus on collect subjective data through self-report, and not through external objective measurement devices. A potential approach to this problem area is to use a mobile phone to collect and capture data and then, provide a web portal to present and visualize captured data, with for example graphical charts. In addition, to store and process reported data have a persistent distributed server.

1.3 Limitations

As we mentioned in our problem statement and in this thesis, we will focus on collecting subjective data through questionnaires and self-reporting. We want the system to support third-part objective measurement devices in the future, but in this thesis, we have limited our scope to only collect subjective data. Our goal are to develop a modern and digitized tools to collect, process, store, present and analyze, therefore, we prioritize usability and functioning tools higher than user interface (design). Other technology possibilities such as, integrate third-party devices, machine learning and artificial intelligent is not a part of this thesis, but should be considered as future work.

Since we are trying to support NIH's research with our envisioned system, NIH will be our test users and provide feedback on our research. Hence, the system has only been tested in football context, and not in other sports contexts. Security aspects and challenges of the system will not be taken to consideration in this thesis, but we have partly taken this to account by storing encrypted user ID's. Thus, there is no way to identify the person. We wanted to test the system in various cloud services, with the primary goal to find any latency advantage from one cloud service compared to another. We will not perform any in-depth research to the network subject. Moreover, the system will only be tested with the free tiers in the cloud services, thus we have not evaluated hardware specifications/performance.

1.4 Research methods

In this thesis, we follow the methodology *design* described in the report *Computing as a discipline by ACM Task Force* [17]. This involves the design, implementation and evaluation of existing systems and pmSys. Since the system is being used in real life scenario, we have identified and defined requirements and missing features in close cooperation with the users, who also provided us great feedback and discussions.

For our development process, we tried to follow Scrum software development methodology with a twist to it. Each sprint consisted of planning, developing, testing and ended with a summary. We tried to have 1-2 week sprints where we implemented new features and solve various tasks. We described each task with various use cases that explained a user's approach to different problems, which led to new requirements and features for the system. All development tasks and use cases were organized, described and discussed at Trello. Trello is a free and visual way to organize anything with anyone[2]. We marked

[2]Trello. `https://trello.com/`

each task on Trello with different priorities and followed the plan with 1-2 week sprints to implement new features and solve tasks.

We have not followed any testing frameworks or methodologies, and we have neither created any unit tests for the implemented applications. However, we tested the applications against various use cases and requirements. We got help from test users at NIH to test the applications. We have also performed a user study for the implemented system, to confirm whether the system was fulfilling the intentions of a monitoring system.

1.5 Main Contributions

In 2014, we had four professional clubs from the Norwegian top division, who used our system to monitor their players. These clubs were Strømsgodset IF, Rosenborg BK, Viking FK and Tromsø IL. The system we provided them at that time was to collect, process, analyze and present data with use of existing system. From this, we learned and identified new requirements that led us into the development of a new improved system. We name this system Player Monitoring System (pmSys).

The main contribution of this thesis, we have implemented a working system that is used by football players. The system provides a monitoring system through questionnaires, and with data collective, processing, analyzing and presenting tools. This new improved version of pmSys is in use by to two top division teams, Strømsgodset IF and Sarpsborg 08. In additional, the system is in use by the Norwegian national team, both senior and under-21 team. With the Norwegian national team using our system, also drew media's attention, and was featured in NRK's newspaper 18. November. 2014[3]. The last team to use the system is Lørenskog IF under-16.

The second contribution to this thesis, we have developed a mobile application that works on two different mobile platforms. The application itself provides a fast approach to self-reporting, and is customized to the athlete's use and needs. The mobile application provides graphical visualization feedback, which give a player the possibility to monitor their own progression and captured data. The application works on Android and iOS devices, and has until May 2015, been downloaded over 190 times. We have also performed a user study to confirm whether this mobile application was better application that existing one, and whether self-reporting through mobile application was a better approach than pen and paper.

The third contribution in this thesis, we have created a web portal to give visualized feedback, and work as an analyze tool in the web browser. This portal was created for the coaches to analyze and monitor their players, and ease and remove the process of manual data processing and analyzing. In combination with the web portal and mobile application, we have implemented our own communication service between the mobile phone and the web portal, allowing a coach to send push messages directly to a player's mobile phone.

The last contribution of this thesis, we have studied deployment of the system in various cloud services. We found the network latency on mobile broadband to be slightly difference in the cloud compared to self-hosting. We also improved the system by lowering the network latency and adding new endpoints to reduce data transfers between the server and client.

To summarize, we have implemented a working system that is used in real life to monitor football players. We have implemented an effective way to collect data, simplified and removed the process of parsing and processing captured data manually. We have also created presentation tools to visualize and present captured data where users can analyze and evaluate captured data.

1.6 Outline

We begin this thesis with presenting requirements and related background in chapter 2, such as sport related backgrounds, and other related systems and concepts. In chapter 3, we investigated and studied Ohmage as a platform for pmSys. Ohmage is a data collection platform through self-report. We then

[3]Ny app skal sikre EM-suksess, NRK. `http://www.nrk.no/troms/ny-app-skal-sikre-em-suksess-1.12048486`, accessed = 2015-05-03

evaluated the existing mobile application and web portal as monitoring tools. We concluded the chapter with an explanation on why Ohmage did not fulfilled our purpose. In the next three chapters, we have presented three different applications used in combination to form a player monitoring system. In chapter 4, we present pmSys mobile application created for athletes in sport context. Here, we present technologies we used to implement the application. Furthermore, we evaluated the new implemented application against the existing application. We then present two user studies we performed to support the evaluation of the new application. In chapter 5, we present a implemented web portal for the coaches where they can analyze and monitor players based on reported data, presented and visualized with graphical charts. This web portal is also able to remotely send push messages directly to the players' mobile phone. In chapter 6, we present a distributed sever which represent the persistent layer of pmSys. Here, we investigated the usefulness of deploying the system in various cloud services, and studied improvements and optimizations towards this layer. The three latter chapters are presented in parallel as shown in Figure 1.1. We summarize and conclude in the last chapter.

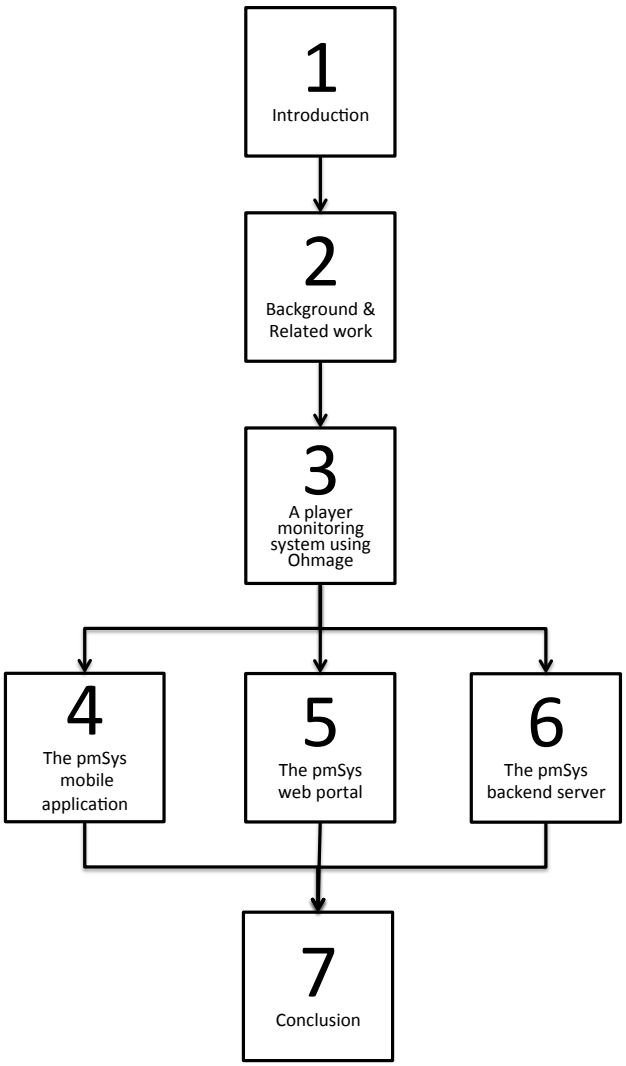


Figure 1.1: Outline

Chapter 2

Background and related work

In this chapter, we present system requirements for pmSys. Then, we are going to present sports related background, such as what data we collect, and why we are collecting it, and how we can present it. Furthermore, we talk about a concept, Participatory Sensing, which is a concept where the participants do data sensing. Further, we present a system for collecting data from third-party monitoring devices, open mHealth. Furthermore, we going to get a brief touch on DHIS2, which is a health-information-system developed by University of Oslo. Next, we talk about how other football clubs monitor their players and another existing monitoring system. At last, we present the most relevant system for pmSys. We summarize and conclude with a brief evaluation of presented systems.

2.1 System requirements

In this section, we present general requirements for our envision pmSys. The system should provide a way to capture and submit data through self-reporting. Then, the system must be able to process and analyze submitted data. At last, the data captured data must be persistent. The system should present the collected data through visualization with user-friendly and interactive graphical charts, this allows the users to monitor and evaluate captured data. The envision system is illustrated in Figure 2.1. We distinguish between function-requirements and non-functional requirements. The function requirements describe the functions the system should be able to do. Non-functional requirements describe how the system should work, behave and perform.

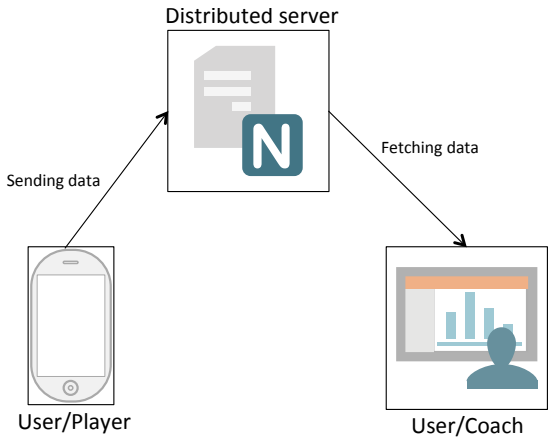


Figure 2.1: The envisoned system.

2.1.1 Functional requirements

Self-reporting

We wanted to use the mobile phone to collect self-reported data with questionnaires. Therefore, the mobile phone must provide a way to present and submit finished responses, for example through a mobile application. Since, we wanted to capture data with the mobile phone, and present the captured data afterwards, the data then must be stored on a distributed server. With a client-server model, we are dependent on internet connection. Therefore, we must provide self-reports for the user independently from whether the network connection is present or not on the mobile phone.

Data processing

After a user has completed a survey and submitted the responses through their mobile phone. The system should then, be able to receive any incoming data from the users at any time. , the system can process, validate and analyze the received data. The system must have a persistent layer to store, organize and secure the captured data. However, capturing and storing data alone, cannot form a monitoring system, therefore, the system must also present and provide data to the client applications.

Presentation and visualization

When the system has collected and saved the data, then the system must provide a presentation of collected data. With a web portal that is accessible from everywhere, we can for example present the reported data in visualized charts. Thus, the data is easier to analyze and evaluate. The mobile application should also provide feedback based on reported data, this allows the users to monitor themselves and motivate them use the system.

2.1.2 Non-functional requirements

Usability

Most of the users, who are football players and sports athletes, might have non-technical background, thus, the system must provide high usability through user-friendly design and simple user interactions. The goal of the system is to provide tools and applications that are easy-to-use. All features and user interfaces (design) must be simple to understand, and not require a manual or any guidance to use.

Performance

In this thesis, we have not focused on hardware performances and specifications, but we rather priorities improved performance related to the users. Such as, the reporting process must be fast and effective, thus the user will report and use the system more frequently. Moreover, removing unnecessary user interactions also result in a faster system. The persistent layer should be able to store data and provide data instantly. The system in general should be fast and reliable.

Scalability

The system must be able store large amount of data, such as responses, surveys, user information and so on. Storing large amount of data over time, and handle multiple concurrent user interactions dependent on various factors. Thus, the system must be able to scale with the growth of the system.

Availability

The system must be available at any time and from everywhere. For the self-reporting process, the user should not be dependent on internet connection, and should be able to conduct a survey at anytime, which mean the mobile phone must be available for conducting self-reports at any time. The persistent storage should provide user access and present data regardless of users' location and server location.

Security and Privacy

The data collected in the system, must be stored securely, and not be accessed by unauthorized users. Due to strict rules and policies in Norway, we have to apply for permission to store and process sensitive information. Therefore, we only store anonymous and generated user id's that cannot be used to identify a person in real life. Only the person (coach/coordinator) who have applied for permission to *the person data act* (Personopplysningsloven) [18], know the connection between the encrypted username and the person/player. Nobody else knows this connection and in case the server is exposed for hacking, the data will not be abused. The connection between a user in the system and the person in real life is managed offline by the responsible person.

User roles

The system should have a clear distinguish between the user roles. The system should describe and provide four different user roles. The *players* who provide the system with data, through self-reported questionnaires. The *medical staff* and *coaches* analyze and interpret the captured data, and gives feedback and results to the players. The *project coordinators* and *researches*, which have the same role as the coach, but have access to multiple teams. At last, we have the *system administrators*, which is responsible for system administrative tasks, such as maintaining, user creation and support the users.

2.2 Sports background

In the previous chapter, we introduced how we wanted to implement a system for monitoring sports athletes. Before we can present such system, we need to discuss the background on what information we collect, and why, but also how we use this information to monitor sports athletes. A methodology to monitor sports athletes is introduced and discussed in these four articles:

- The Oslo Sports Trauma Research Center questionnaire on health problems: a new approach to prospective monitoring of illness and injury in elite athletes [19].
- Development and validation of a new method of the registration of overuse injuries in sports injury epidemiology: the Oslo sports Trauma Research Center(OSTRC) overuse Injury Questionnaire [4].
- A new approach to monitoring exercise training [1].
- Neuromuscular, Endocrine, and perceptual fatigue responses during different length between-match micro cycles in professional Rugby league players [20].

Here, they monitored an athlete's Rating of perceived Exertion (RPE), wellness and overuse injury through questionnaires and self-report. These three subjects are also the basic to our data collection and monitoring approach, thus we only want to collect subjective data from the players.

2.2.1 Rating of perceived exertion

Rating of perceived exertion (RPE) is a common method to rate the intensity of an exercise [21]. In football context, this would be captured after a training session or a match. The RPE value is given by a category rating from 0 to 10. This RPE scale is based on Gunnar Borg's RPE scale [1]. An example of RPE scale is presented in Figure 2.2, where typical follow up question is "How was your session?", and the present athlete respond with a number.

A reason for collecting subjective RPE data over third-part measurement devices, is because the objective measure devices such as heart rate monitors, cannot tell how an athlete actual felt about the intensity of the session [1]. By capturing the athlete's RPE value, a daily exercise score(load) can be calculated by multiplying the duration of training/match session and the RPE value. By capturing RPE scores for a week, we can present and calculate a weekly load for an athlete. From this, we can create training-session indices, such as monotony and strain, and potentially provide an index of the likelihood

Modification of the Category Ratio Rating of Perceived Exertion Scale

Rating	Descriptor
0	Rest
1	Very, Very Easy
2	Easy
3	Moderate
4	Somewhat Hard
5	Hard
6	.
7	Very Hard
8	.
9	.
10	Maximal

Figure 2.2: RPE scale [1]

of untoward training outcomes [1]. Monotony is a measure of day-to-day training related to the start of overtraining when the weekly training load is high [2]. The strain is calculated in combination with Training load and Monotony, and is a method for monitoring training when players are undertaking high training load, and the recovery becomes the fundamental to training when high load are being taken [2]. All mathematical equations to calculate the various numbers, is presented in Figure 2.3

$$Load = RPEvalue * duration \tag{2.1}$$

$$Monotony = WeeklyLoad * standarddeviationofweeklyload \tag{2.2}$$

$$Strain = WeeklyLoad * Monotony \tag{2.3}$$

Figure 2.3: RPE related calculations [2]

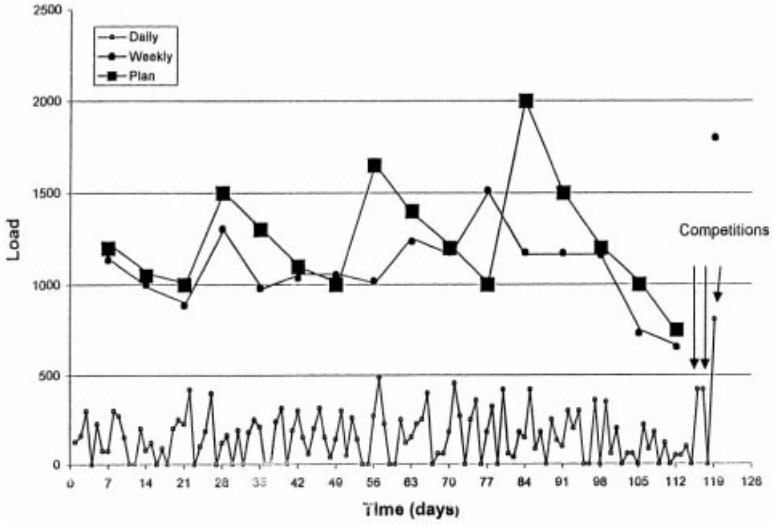
With the weekly and daily load, strain and monotony for a player, we can create a weekly exercise diary and reveal an overall weekly pattern of exercises [1]. We can also present the weekly load in graphical charts and allow the coach to have a visual impression of the session and the player's perception of the session. In additional, the coach is able to add expected/planned load for the team session and analyze how a player or the team perceived in terms of to the planned load. An example of weekly training diary and visualization of this diary is presented in Figure 2.4.

2.2.2 Wellness

We capture wellness data through five topics: fatigue, sleep quality, general muscle soreness, stress level and mood. These five wellness values are evaluated and rated by a scale from 1 to 5. An example of such wellness sheet is presented in Figure 2.5. Once a day should the data be captured, and are preferred to be done in the morning, to get the best realistic results. and are preferred to be done in the morning, to get the best realistic results. In combination with wellness score and weekly load, can the coach determine whether the player is ready for the upcoming match or training session, and further see if the load is affected by wellness [3]. Wellness data can help players recover between match and training, for example, reduce training load [3].

Day	Training activity	Session rate of perceived exertion (RPE)	Duration (min)	Load
Sunday	Cycle 100 km	5*	180	940
Monday	Jog 5 km + extensive stretch	2*	25	50
Tuesday	Skate 6 × 10 min at AT pace/5 min rec	7*	120	840
	Explosive weights + abs	7*	40	280
Wednesday	Cycle 30 km	3*	60	180
Thursday	Skate 10 × 3 min at 5 km pace/5-min rec	8*	75	390
Friday	Jog 5 km + extensive stretch	2*	25	50
Saturday	Skate 20 × 1 min at tempo/2-min rec	8*	75	390
	Explosive weights + abs	7*	40	280
	Weekly load			3400
	Monotony (× *SD*)			1.26
	Strain (load × monotony)			4284

(a) A schematic example of training diary [1]



(b) A visualization example of weekly load over a period [1]

Figure 2.4: RPE monitoring [1]

	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	

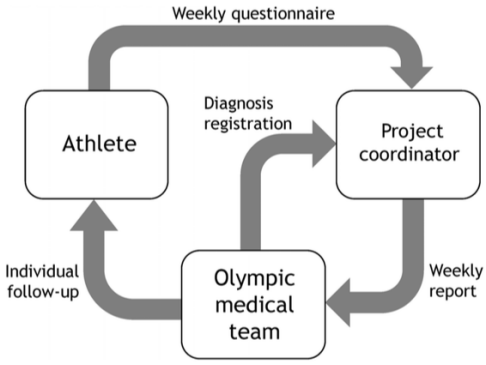
Figure 2.5: An example of wellness sheet [3].

2.2.3 Monitoring illness and injury

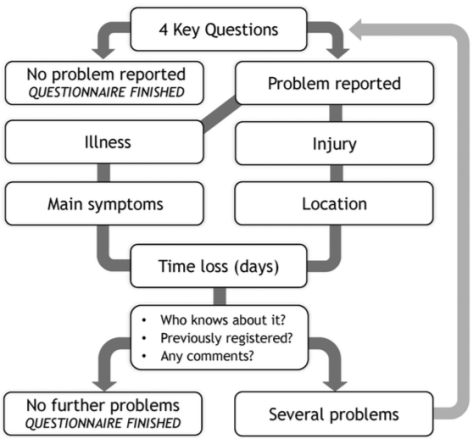
In the paper by Oslo Sports Trauma Research Centre, *a new approach to prospective monitoring of illness and injury in elite athletes [19]*, they introduced a method for monitoring players based on weekly questionnaires. The surveys were sent to athletes over email weekly. After an athlete has finished the survey, a project coordinator collects them and assembles a report based on the responses. Then, coordinator sends the report to medical staff, which follows up with the athlete [19]. An example of this monitoring process can be found in Figure 2.6(a). However, if an athlete forgot to report, the medical staff had to send a reminder over email.

The weekly questions are to capture health problems such as injury and illness for a player. The survey starts with four key questions, which can be found in Figure 2.7. If the athlete choose the minimal value for each questions (no problems), the questionnaire is finished for the week [19]. However, if the athlete reported anything else than the minimum value, the questionnaire continues by asking them whether their problem is an illness or an injury. If the athlete reported an injury problem, the following question is where the injury is located, and if it was an illness problem, the athlete was asked for symptoms (see Figure 2.6(b)).

Based on the four key questions, each answers form each question is presented to a numerical value from zero to 25, and these four numbers summarized gives a severity score from zero to 100 for each overuse problem. The severity score is used as an objective measure of the consequences of an overuse problem, and can be used to monitor the progress of overuse problems [4]. These data is used to define athletes' injury status, and see if they have potential to moderate or severe reductions in sports performance or participation [19].

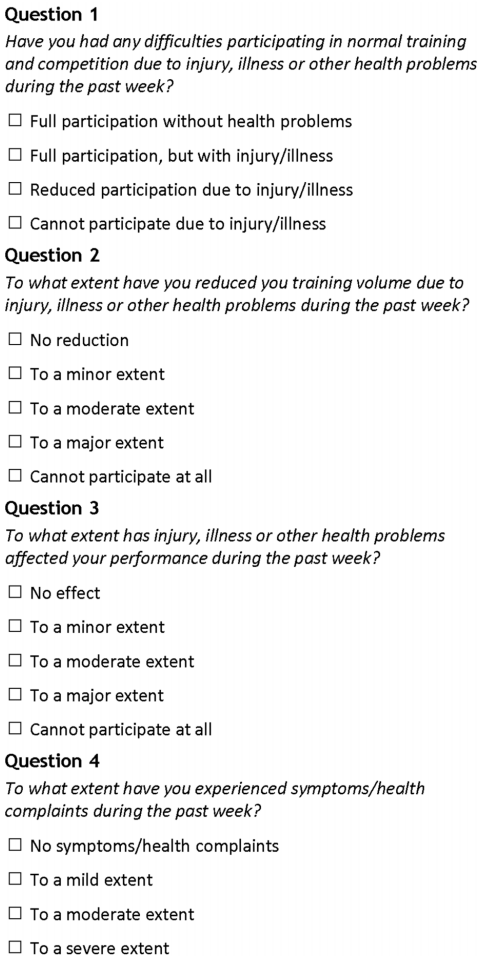


(a) Methodlogy for monitoring [19]



(b) Diagram of injury report [4]

Figure 2.6:



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 2.7: Injury questionnaire [4]

2.3 Participatory sensing

Participatory sensing is a concept about use the mobile devices to form interactive, participatory sensor networks that enable public and professional users to gather analyze and share local knowledge [22]. This concept moves the data sensing from observing devices to the users. An example of participatory sensing architecture is shown in Figure 2.8. The presented figure shows the architecture in participatory sensing with use of mobile devices to capture data, then sent to a server and presented afterwards.

From the previous section, we mentioned how subjective data might give more precise data than objective data from third-party devices, because we can report how we actual feel, compared to what the monitoring devices think we feel. By following the participatory sensing concept, we move the sensing part from sensor devices to mobile devices, and the users are in control of what they wants to report. According to a report from MediaNorge in 2014, 81 percentage of Norwegian population have a smart phone device [23].

Participatory sensing applications uses data from the mobile phone as a sensor to gather information in collaboration with its operator [22]. A campaign model defines participatory application architecture. This campaign model consists of four different roles. *Initiators*, who create campaigns and specify data collection challenges. *Gatherers*, the mobile users who participate in data gathering that may be network-triggered, user-initiated or continuously reporting. *Evaluators*, who verify and classify collected data on behalf of the campaign. *Analysts*, the person who interprets and presents conclusions. These four roles are taken from article Participatory Sensing [22]

We wanted to build a system that is partly close to this concept, where the reporting responsibility is moved to the users, instead of to have coaches take notes and reports. Participatory sensing concept emphasizes using various sensors located in the mobile phone to capture data. However, we have followed this concept partly by using the mobile phone to capture data with the user self-report through

questionnaires.

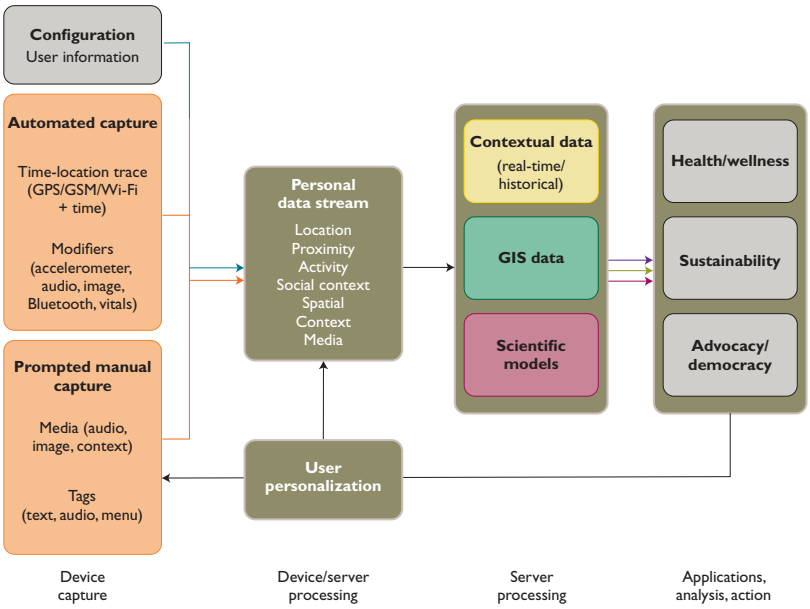


Figure 2.8: Common architecture in participatory sensing applications [5]

2.4 open mHealth

Chronic diseases like diabetes, asthma and obesity account for 46% of global diseases burden, using traditional model for episodic care in clinic and hospital-based settings is sub optimal for improving chronic disease outcomes [24]. mHealth is based on the principles of eHealth, which is health services and information with use of *information and communication technology* (ICT). mHealth move the focus towards the mobile phone instead [25]. By using mobile devices in conjunction with Internet and social media, we can enhance disease prevention and management by extending health interventions beyond traditional care, this approach is referred in the article *Open mHealth Architecture* [24].

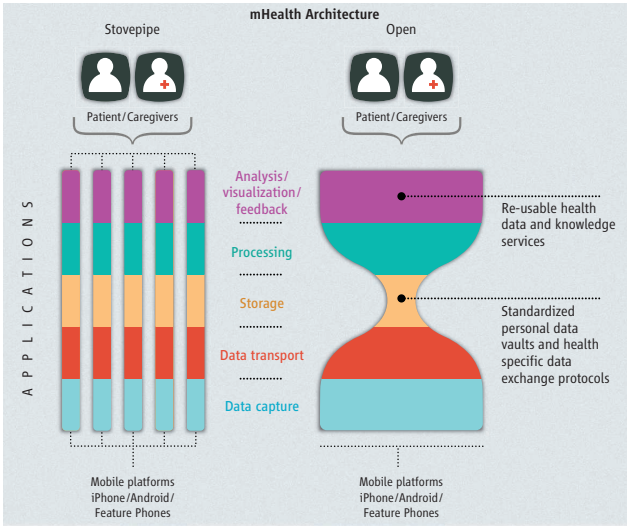
The idea behind open mHealth is to make it feasible for patients, to collect and share relevant data in real time to any time. And not only when visiting a clinic, it allow more rapid convergence to an optimal treatment. For example, a patient who needs to monitor their own health by writing notes for each day, and report to the clinic monthly or weekly. With an application to report this data in real time to a hospital or clinician, who are following the patient's health status, the patient can get instantly reports, and the clinics can look for patterns of the response, if any abnormal is present, the patient can be contacted immediately for further guidance [24].

Today's health learning systems is often closed application and systems with their own proprietary data format, management and analysis, such as "stove-pipe" architecture described in Figure 2.9(a). Here we see that Stovepipe architecture is closed for reusing and sharing of data. With open mHealth architecture, we get an open interface for data capturing and reusable data for the users. Open mHealth architecture is built around shared data standards and global communication networks. With an open architecture, components have well-defined, published interfaces, which allow interconnection and use in ways other than originally implemented and intended [24].

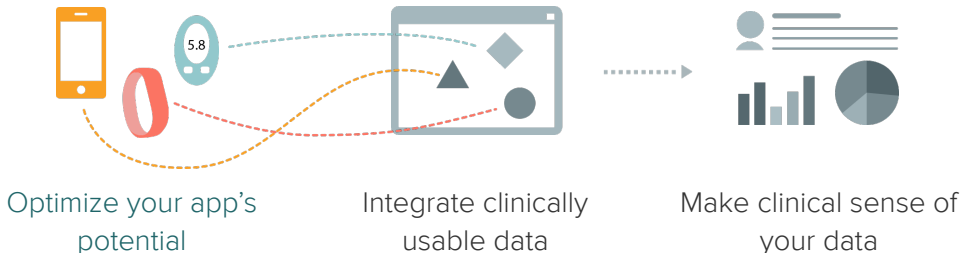
Open mHealth architecture is currently only capturing data from objective devices, such as hearth monitor devices and more. It is mainly focusing on saving and capturing health data, for example body weight, body height, hearth rate and so on[1]. The captured data is used to provide visual feedback to be analyzed, see Figure 2.9(b). Our envisioned pmSys is about data collecting based on subjective data and with questionnaires. Open mHealth does not provide methods to capture data through questionnaires, and neither does the system provide any applications to present data. Open mHealth is great concept and

[1]Read more about open mHealth here. `http://www.openmhealth.org/`

system, but not ready to be used as our system platform yet. However, as a part for future work, open mHealth's third-party data collection approach and methodology is relevant for our system.



(a) Stove-pipe vs open mHealth. [24]



(b) Open mHealth system [26]

Figure 2.9: open mHealth system

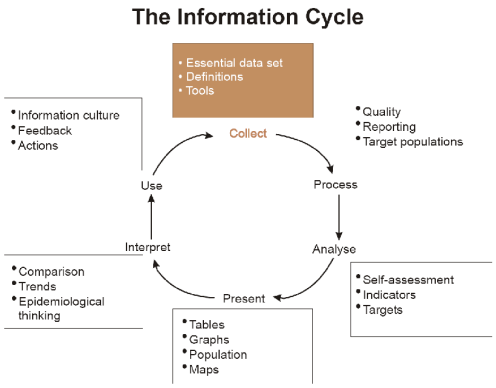
2.5 District Health Information System

District health information system 2 is an open source system developed by Health Information System program (HISP). HISP is an open and globally network established distributed developers spread over countries like India, Vietnam, Ireland, Tanzania and Norway, and the development is coordinated by University of Oslo (UIO) and Norad [27]. The system is used in more than 30 countries in Africa, Asia and Latin America, and these countries are trying to adapt DHIS2 as their national-wide Health-information-system (HIS) [27]. DHIS2 is a tool for collecting, validation, analysis, and presentation of aggregate statistical data, tailored (but not limited) to integrated health information management activities [27] (See figure 2.10(a)). In general, DHIS2 is a generic tool rather than pre-configured database, with an open meta-data model and a flexible user interface, that allows the user to design the contents of a specific information system, without the need for programming background [27]. Some key features of DHIS2 [27].

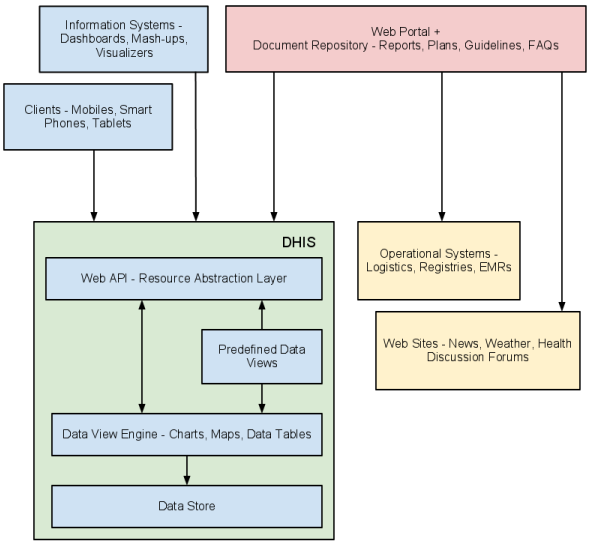
- No need to have programming experience in order to start using DHIS2. Customization and localization is done through DHIS2's user interface. In addition, it can be easily set up in different countries, regions or district.
- It can provide data entry tools which can either be in standard form (predefined) and tables or it can be custom made to replicate forms.
- DHIS2 can provide validation and improvements of data quality.
- Flexible and dynamic (on-the-fly) data analysis in the Data Visualizer.

- User management module for passwords, security, and fine-grained access control (user roles).
- Messages can be sent to system users for feedback and notifications. Messages can also be delivered to email and SMS.
- Integration with other software systems and use the DHIS 2 Web-API and the Integration Engine.

DHIS2 is designed and created for clinics and hospitals, where they need a system to keep personal health data organized and stored, such as personal journals. DHIS2 provides a great system to be used on desktop computers within a web browser, the system is also heavily focused on health and personal data. Thus, DHIS2 does not have a mobile application where patients can report data, but DHIS2 provides a web API that can easily extends the system with new applications. DHIS2 cover another area and their purpose is to replace health systems national wide, our goal is to implement a monitoring system through questionnaires, using DHIS2 as a monitoring system throws away the whole purpose and potential of DHIS2. But we partly want to take DHIS2 web API architecture (see Figure 2.10(b)), and use it as a requirement for our envisioned system. Thus, we can easily extend the system with new customized applications to the users' needs.



(a) The information cycle of DHIS2 includes data collecting, processing, analyzing, presenting, interpreting and provides feedback [27]



(b) DHIS2 architecture [27]

Figure 2.10: DHIS2 platform

2.6 Other sport systems

Milan labs is a research center for the Italian football club A.C Milan, and has been doing research on health objectives since 2002 [28]. Milan labs primary goal is to study the health of the players, and identify guidelines to get the best out of the Milan players. Milan Lab's most used methodology is by following up on each player wellness, and use information technology to optimize and present the results. Seattle Sounders from Major League Soccer (MLS) in U.S.A is also monitoring their players through wellness and RPE. Most of every professional clubs have their own way to monitor and follow up their own players. This information on how they do it or what they use, is not often shared among the public. Tromsø IL is one of teams in the Norwegian top division that uses monitoring devices such as XYZ[2], to monitor their players in training and match [29]. As we mentioned in Open mHealth section, these monitoring devices and systems is closed for extending with new applications. The users are also bounded to use their software and nothing else.

[2] `http://www.zxy.no/`

Athlete Monitoring[3] is a system for monitoring athletes through questionnaires on load, wellness, illness, fatigue and more. They are offering mostly the same functionalities we want to provide, where coaches can analyze the players reported data, and the players can self-report data through a mobile phone. The problem with Athlete Monitoring system is that it cannot be easily extended by other developers, and the system does not provide any open API like mHealth and DHIS2. The users of Athlete monitoring system is also bounded to use their software. Another cons with Athlete Monitoring, it comes with a cost, and the more users we have, the higher is the price. What our envisioned system want to accomplish differently from Athlete monitoring, is to have open public API, where the system can easily be extended by new applications. Then the users are not bound to use our software, thus they can create their own analyzing tools and software.

2.7 Ohmage

Ohmage[4] is a modular and extensible open-source platform. The platform provides mobile and web participatory sensing platforms that records, stores, analyzes and visualizes data from prompted self-report and continuous streams [30]. Ohmage is built upon the Participatory Sensing concept where the patient in health context can report different symptoms regularly, and clinics can monitor their patients and provide instant feedback. Ohmage have a rich set of API endpoints for their server back end that provides data access across all applications [30]. Ohmage uses mobile application to collect data and a web portal to analyzing captured data. How to Ohmage platform is built can be viewed on Figure 2.11

As a monitoring system, Ohmage fulfill our purposes and requirements. Ohmage have self-reporting mobile applications on iOS and Android. The ohmage-platform also has a back end server with data storage and user management. This back end is also open to be extended with new applications. At last, Ohmage comes with a web portal where the captured data is presented and visualized.

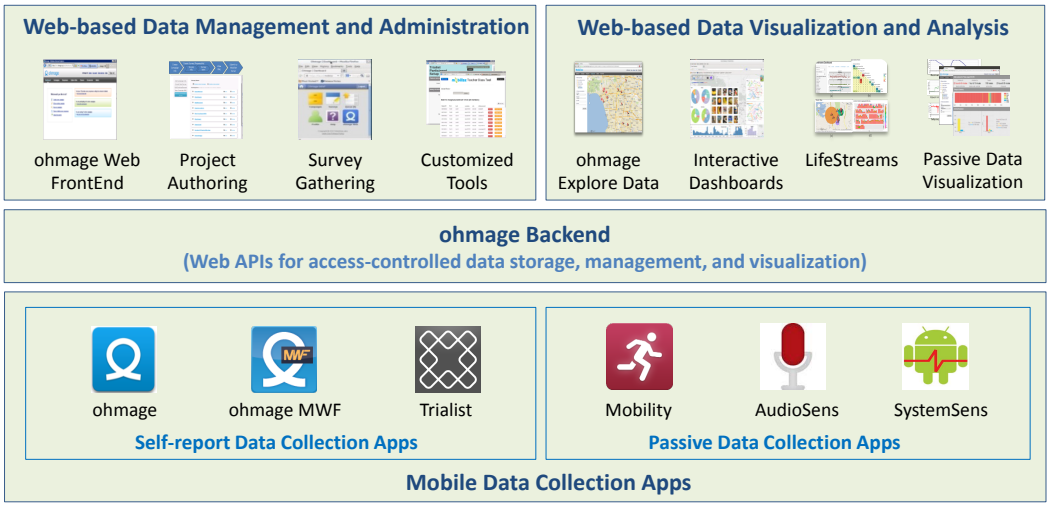


Figure 2.11: Ohmage architecture with applications built around Ohmage backend

2.8 Summary

Our goal is to develop a system to monitor players through questionnaires, and as we have already mentioned, the combination of pen, paper and excel sheet is not an option. Online web services do provide a user-friendly presentation of the survey, but these surveys must be conducted through a web browser and does not provide offline reporting. After all responses have been submitted through online web surveys, the coordinator still have to manually organize and analyze the data afterwards, but also use time to generate readable charts before they can start to evaluating the result.

[3] http://www.athletemonitoring.com/
[4] http://ohmage.org

We think that having a distributed server with a persistent database partly solves our problem, because then, the data is available from everywhere, in addition, organized and structured using database technologies. The server can then validate reported data, before it, stores captured data. In additional to the server, we need to have a reporting tool and analyzing tool. With a mobile application that is installed directly on the phone, we perform self-reports at anytime from anywhere, and replace the pen and paper approach. With a system, that automatically process and present the captured data, will let the user have focus on analyze and evaluate collected data, rather than manually organize and process captured data. The combination of mobile application, analyze application, a distributed server, solves, and relief the previous monitoring approaches, and introduces a new method and process for monitoring.

The only system we have presented and discussed until now that fulfilled most of ours requirements, were Ohmage. Ohmage has a mobile application to conduct self-report through questionnaires, and a web portal to analyze and present data, and a persistent server to save data securely. All Ohmage applications are built around the back end server, which means anyone can easily extend the system with new application to fit their use, without removing the core functionalities of existing applications and Ohmage platform. In next chapter, we present the first version of pmSys, using Ohmage.

Chapter 3

A player monitoring system using Ohmage

We start this chapter with a discussion and evaluation from the previous chapter. Next, we presents pmSys using Ohmage, and how Ohmage system fits our purpose and requirements. Then, we are going to present a mobile application we used for data collecting through self-reporting and our evaluation around this application. Furthermore, we discuss and evaluate a web portal developed by Ohmage, which is made for analyzing and presenting captured data. We summarize with cons and pros with Ohmage, which led us to new requirements and features for pmSys.

3.1 System discussion

The goal of this thesis was to implement a monitoring system to capture an athlete's load, wellness and injury through questionnaires. We wanted to modernize this approach by using modern technologies. We presented the functional requirements for our envisioned system in the previous chapter; in short, the system must be able to perform three main tasks *Capture, Process and Present* data.

We wanted to solve these tasks by partly follow the participatory sensing concept, about using mobile phone as a sensing platform, and have the users conduct surveys through self-reporting with their mobile phones. Thus, we need to present the captured data in order to analyze and monitor afterwards. Therefore, we must have a place to save the reported data from the users. With a distributed server to collect and process the reported data from the users, we can have applications and tools built around it to form our envision system. With an open interface API like DHIS2 and open mHealth, make the system both, scalable and extendable by new applications.

Although with all systems mentioned in previous chapter, we started to design our envisioned system from scratch. We wanted the system to have a database to store all captured data, and with a single database layer alone would not work, because we must validate and analyze reported data before we can store it into the database, if not anyone would submit random reports and upload data as they want. With an application layer above the database, the application layer can then handle all incoming and outgoing data before communicating with the database layer. Then, we have to consider user management and data privacy, and as we designed and evaluated this idea, the larger and advanced the system became new problems and requirements evolved.

Since, NIH wanted to start their research as soon as possible, and wanted to try our envisioned system for their research. We knew that the system implemented from scratch would not meet NIH's wish, thus we considered the idea of using existing systems. We decided to try Ohmage as our monitoring system. Ohmage provided applications we needed to conduct self-reporting, store reported data and a web portal for presenting data. Ohmage have a server application that we could deploy to a distributed server. The server application allowed us to define and create our own surveys, and then, present the surveys in Ohmage's mobile application. With a distributed server approach, the server can capture and process reported data at any time, which provides the system availability. In addition, Ohmage has a web portal for users to view and analyze the reported data; the web portal also provides basic visualizations. At last, Ohmage provides a user management system, we could use to separate two teams in the system, which means, we can save data from multiple teams on the same server, and Ohmage will provide data isolation, and prevent unauthorized data access. In Ohmage, we were also able define our own username.

Thus, we could encrypt usernames with machine-generated text and all user information and reported responses will be stored anonymously. Only the coach with permission knew the connection between a user in pmSys and in reality. This information was for security reasons not stored on the server or in Ohmage. The first version of pmSys with Ohmage is presented in Figure 3.1.

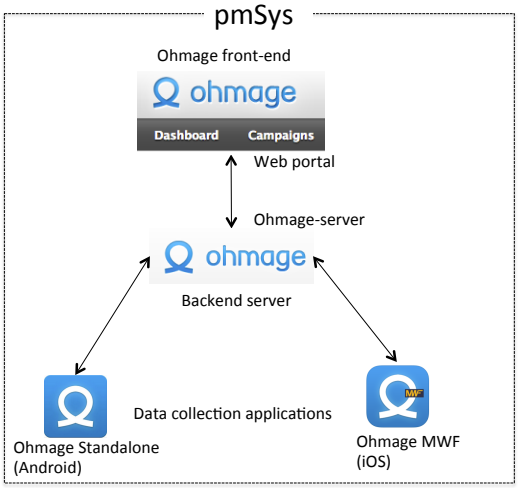


Figure 3.1: First version of pmSys using Ohmage

3.2 Ohmage-system

We got a brief introduction to Ohmage in Section 2.7. Ohmage-platform consists of four primary components: (1) Ohmage backend that serves as data storage and provide unified interface for data access. (2) Mobile data collection applications, that runs on participants' phones for collection and feedback. (3) Web based data management and administration tools for study management and administration. (4) Web based data analysis and visualization tools for exploring, analyzing and visualizing the collected data [30].

Ohmage fulfilled our requirements as a system, and provided a system that could report, process and present data. Since the system was open and extendable with new applications, we could develop new application to fulfill our usage if necessary. The first version of pmSys was based on Ohmage system. Moreover, we used all Ohmage applications, such as Ohmage mobile application for self-reporting, and the front-end web-portal to analyze and present data. We distributed Ohmage mobile application to our users, for them to start report. Next, the coaches were introduced to Ohmage web-portal, where they could view captured data. In the next sections, we present some features Ohmage-system was providing, and the features we used to define pmSys with Ohmage. These features were also one of the main reasons we decided to build pmSys with Ohmage.

3.2.1 Campaign

From Participatory Sensing concept, we mentioned a campaign model, which is a group of users participating in a program; this is called campaign in Ohmage. For our monitoring system is campaign representing a program for a team. A campaign is used to define various surveys for a team. In a campaign, we can define different surveys, and within a survey definition, we can define multiple prompts. A prompt is an information type within a survey; it contains information on how questions should be present, and various options connected the questions. The prompt type is whether a single choice question, number question or time stamp, we discuss these three prompts later. A campaign in Ohmage is defined in XML-format, and contains a name and a uniform resource name (URN) of the campaign.

In Ohmage, the URN to a campaign have an important role, we are using the URN to identify a team in conjunction with classes (We come back to classes later). A standard Ohmage URN consists of

[urn:campaign:...]. And in our system, we are defining the urn by concatenating *urn:campaign* with a team's initials and who the campaign is created for, this is either players or coaches. Then, we end the campaign URN with a version number of the campaign, and the language of the campaign. This version number is updated every time a campaign is altered, and the language is either *eng* or *nor*. We used the language definition to create the campaigns in various languages. It is important that the team name in campaign URN is the same in the class URN. Because we are using the class and campaign urn to find and validate, which team a user belongs. An example of an URN: *urn:campaign:**demo:players:6:eng***

A system administrator can only upload the XML defined campaign. We are not allowing the coaches to do upload campaigns, because we want the coaches to focus on monitoring and analyzing the player's collected data rather than administrative tasks. A campaign has one running state, running or stopped. When a campaign is stopped, it is not possible for the users to submit data to the campaign. However, still possible for players and coaches to read reported data from the campaign. If the campaign is set to running, only then users can report data. In additional to the running states, each campaign has a privacy states, which is shared or private. We are not using this feature to define anything in pmSys. Nevertheless, we have set the privacy state to be private to ensure that shared data is not possible. This also allows us to keep data isolated, and avoid players from accessing each other's data. In Figure 3.2, we have presented a small example on how a campaign is defined in Ohmage.

```
<?xml version="1.0" encoding="UTF-8"?>
<campaign>
  <campaignUrn>urn:campaign:demo:players:5:eng</campaignUrn>
  <campaignName>Demo Survey: Players</campaignName>
  <surveys>
    <survey>
      <id>srpe</id>
      <title>sRPE (after session)</title>
      <description>Survey after session</description>
      <contentList>
        <prompt>
          <id>srpeWorkload</id>
          <displayLabel>Workload</displayLabel>
          <promptText>How was your session today?</promptText>
          <promptType>single_choice</promptType>
          <!-- Properties tag-->
          <value>0</value>
          <label>0 - Rest</label>
          <value>1</value>
          <label>1 - Very, very easy</label>
          <!-- More rpe options -->
          <value>10</value>
          <label>10 - Maximal</label>
          <skippable>false</skippable>
        </prompt>
        <!-- Other prompts here -->
      </contentList>
    </survey>
    <survey>
      <id>wellness</id>
      <!-- Wellness survey -->
    </survey>
    <survey>
      <id>overuse</id>
      <!--  Injury/illness survey -->
    </survey>
  </surveys>
</campaign>
```

Figure 3.2: A tiny example of a Campaign file

3.2.2 Surveys

Within a campaign it contains one survey collection and each surveys within the collection is identified with a unique name. We have three survey items within the collection, and the three survey items consist of RPE, wellness and overuse injury. Within all the survey items, the questions and options are defined

in prompts. A prompt contains information about how a question should be presented, including the question and various options.

In Table 3.1, we see which prompt type Ohmage supports and a description on what each prompt can do. We are only doing questionnaires with a simple question and single option/value as answer; therefore, the only prompt types we need and use are single choice, number and timestamp. Single choice is a prompt type with a question, and multiple options presented, but only one can be selected. Number prompt consists of a number with min, max and default value, here the players can for example, register how long their session was. Timestamp is used in the campaign for coaches, where the coach can set a time and date for their planned sessions, and is used to report expected/planned RPE value a session.

Prompt type	**Descrption**
single_choice	A single-choice answer from the user
multi_choice	A multiple-choice answer from the user.
number	A whole or decimal number.
timestamp	A moment in time by a user down to the second granularity. the timestamp includes both date and time.
text	A free-form piece of text provided by the user.
photo	A photo taken by the user, generally at the time that the user was prompted.
video(only Android)	A video taken by the user at the time that the user was prompted.
audio(only Android)	An audio recording taken by the user at the time that the user was prompted.

Table 3.1: Available prompt types in Ohmage [12]

A prompt can also have conditions on whether to show the prompt or not. This way we can limit the questions presented and do not need to present them all. The condition logic in Ohmage is based on which option the user picks on previous questions. For example, we can ask a player "Was this a?", and the player can pick between (1) Match, (2) Team session, (3) Individual session. Then, the next questions is presented based on what the player picked previously. If the player picked *Team session*, the next question would be "Which type of session?", with options like (1) Endurance training, (2) strength training and (3) other. However, if the player picked "Match" from the previous question, then the question about which session types, would not be presented. Possible prompt conditions in Ohmage are equal (==), less than or equal (<=), greater than or equal (>=), not equal (!=), greater than (>) and less than (<). All these conditions can be used in conjunction with *AND* and *OR*. Each prompt have a unique id within a survey, for example id1 for question one, and id2 for question two. The condition logic is based on these ids. We can for example define a condition logic with (id1 == 1) OR (id2 == 0) for a question, then the question will be presented based whether the id1 was 1 or id2 was 0, otherwise, the question will not be presented.

3.2.3 User system

Ohmage's user system is one of the central entities in ohmage, and controls accessibility for the users. It exists three entities within Ohmage, users, class and campaigns. A user can be connected to one or more classes, and campaigns can have one or more classes. Within a class, we have two roles, restricted and privileged. Within a campaign, it exist four different roles, participant, author, analyst and supervisor. These roles is explained more on Ohmage's wiki page [31]. A user's role is defined by the class role and the campaign role.

In pmSys, we are not using the Ohmage roles to the utmost. Instead, we have defined our own roles: player, coach, and coordinator/researcher and system admin. *System admins* are the ones who have responsibility for the system, they are the ones who are creating users, and assign a player or a coach to a team, and giving them the right privileges. System admins also have the responsibility to upload campaigns, and assign the correct team to the campaign. The *player* role is just a normal user in

Ohmage, and corresponds to the participant role in Ohmage. This user only can read their own data, and have restricted role within a class. A *coach* is a user with privileged permission within a class. Privileged users are able to read all players' data. However, the coach cannot read data from other teams in the system. The *coordinator/researcher* has more like an analyst role. They have the same privileges as the coach; in addition, they allowed being an analyst on multiple teams. The coordinator role can only access data they have assigned to. If the system not assigns a coordinator to any campaign, then they would not be able to read or analyze any data. The purpose with this coordinator role is that they can analyze and interpret data for multiple teams. An example for this role is the researches and coordinators from NIH.

We are using Ohmage classes to define a team, within a team, we have two different classes, one class that represent players, and the other class represents coaches Each class in Ohmage has an uniform resource name (URN). This urn is used as a unique identifier for a team, and the syntax is *urn:class:nameOfClass*. In pmSys, we have included the team name's initials in the urn, and the group role, which are coaches or players. At the end of the class URN, we added a number; this is to separate multiple teams within a club, or a club with same initials. An example of team demo's URN would be, *urn:class:demo:players:1*, and for the coaches, it would be *urn:class:demo:coaches:1*. If the demo club has another team (user group) and, want the data isolated from team 1, we can add a class with the URN, *urn:class:demo:players:2*, and still have the coaches accessing data for both teams.

An illustration of pmSys user system with Ohmage is presented in Figure 3.3. This figures illustrates two example teams, alfa and beta. The two teams have two classes, which represents the coaches and the players. The coaches are privileged in their class, and the players have restricted privilege. The player class is assigned to the player campaign, where the surveys are defined and the responses are stored, because of the restricted privilege, the players cannot read each others responses. The coach class is assigned to both player and coach campaign, the coach campaign is used to report planned RPE and is only shared between the coaches. By setting the coach to privileged in players campaign allow the coach to read every reported responses from each player. The same distribution and allocation counts for the other team. It is important to notice, that the teams cannot view each others data. And then, we have the analyst and researcher team on top, and they have the privilege role within a class, and is assigned to campaigns they are research for. The illustration describes the whole user system in pmSys.

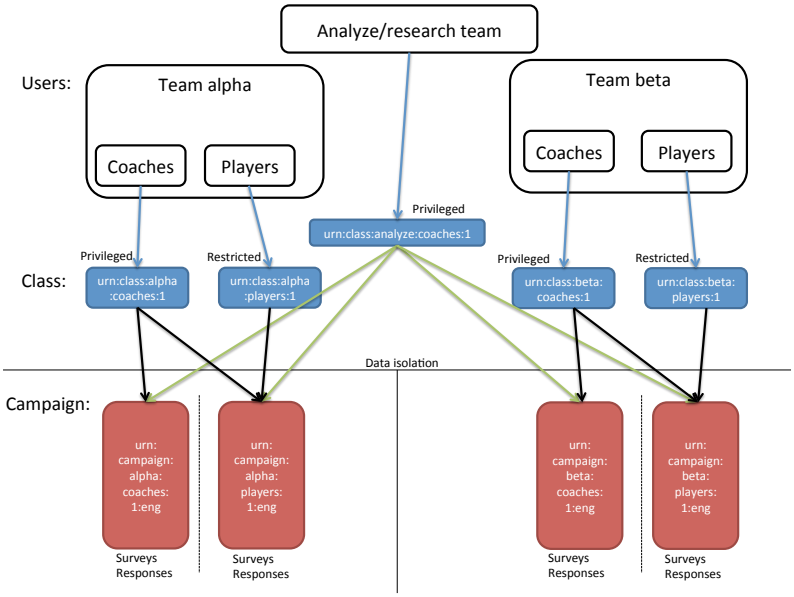
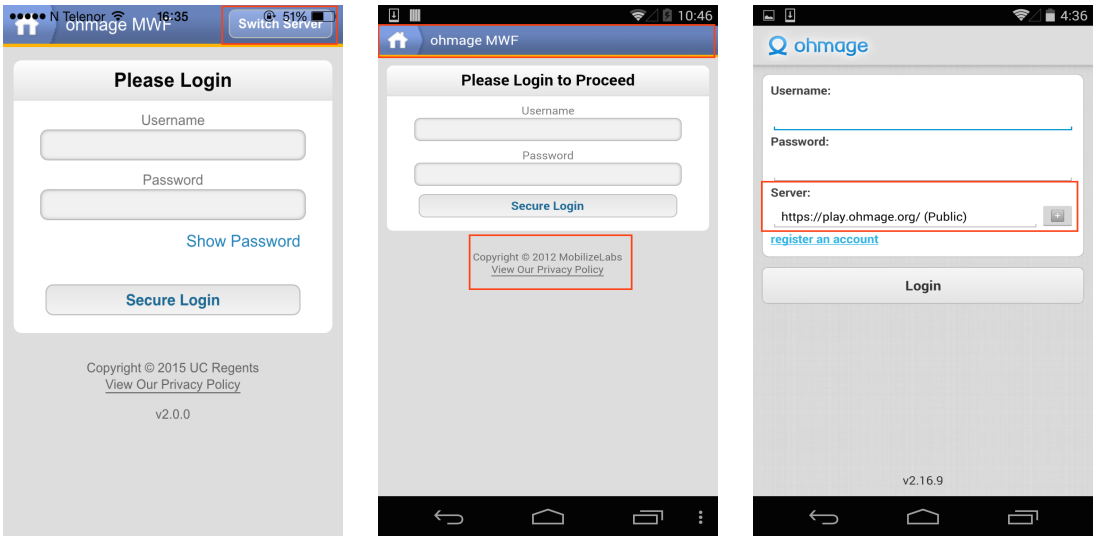


Figure 3.3: pmSys user system

3.3 Ohmage mobile applications

Along with Ohmage platform, Ohmage have two mobile applications for self-reporting, one for Android devices, and the other one for iOS devices, such as Apple iPhone. Ohmage MWF, is the iOS version of Ohmage, and is developed by University of California (UCLA) [32]. MWF stands for Mobile Web Framework, and is a development framework by UCLA [33]. The same Ohmage MWF application exists

on Android, but has not been updated since 2013 [34]. Therefore, Ohmage MWF version on android had various bugs that did not existed in iOS version. The problem with Ohmage MWF on Android, was that we could not change endpoint server we wanted to communicate with, but this was possible on the iOS version, see Figure 3.4(b) Android users were addressed to use Ohmage standalone version, because the Android MWF version did not work. The Ohmage standalone is not built with MWF framework, and is developed by Ohmage team [32].



(a) Login screen, Ohmage MWF (iOS)

(b) Login screen, Ohmage MWF (Android)

(c) Login screen, Ohmage standalone

Figure 3.4: Not able to select server on Ohmage MWF Android version

The first version of pmSys was distributed to our users with both Ohmage mobile applications. The goal was to start monitoring and capture data at as early as possible. Then, evaluate the application to find new requirements and missing features, and then, implement a new application that is customized and created for the sports context. With two different applications, we quickly faced different challenges with our users. Both them, and we have to learn how to use both applications, if would be easier if we only could deal with one application. In the next sections, we present the features in both Ohmage mobile applications on iOS and Android.

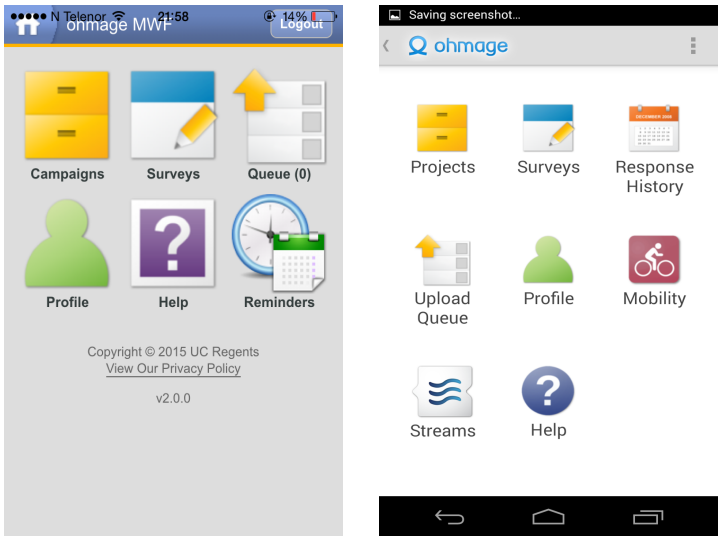
3.3.1 General features

After we have logged in to the applications, we are presented a dashboard with access to all features in both applications (See Figure 3.5). On Android we have *Projects, Surveys, Response history, Upload Queue, Profile, Mobility, Streams and Help*, and iOS we have access to *Campaign, Surveys, Queue, Profile, Reminders and Help*. We can see here that both application did not provide the same features.

We did not get *Mobility* and *Streams* feature in Ohmage standalone to work. But the mobility feature was created to capture data on whether you walked, ran or drove with use of accelerometer, Wi-Fi and GPS on the phone [30]. Mobility data capture could been a useful feature for our system in the future, such as collecting data on how far a person have walked or ran for a day, and then, predict a person's load for a day. Data streams were created to send passively and continuously captured data from mobile sensors and the application [30], and we did not get this feature to work either. The *profile* and *help* on both application, was to customize application settings and present various application guidance.

3.3.2 Campaigns and projects

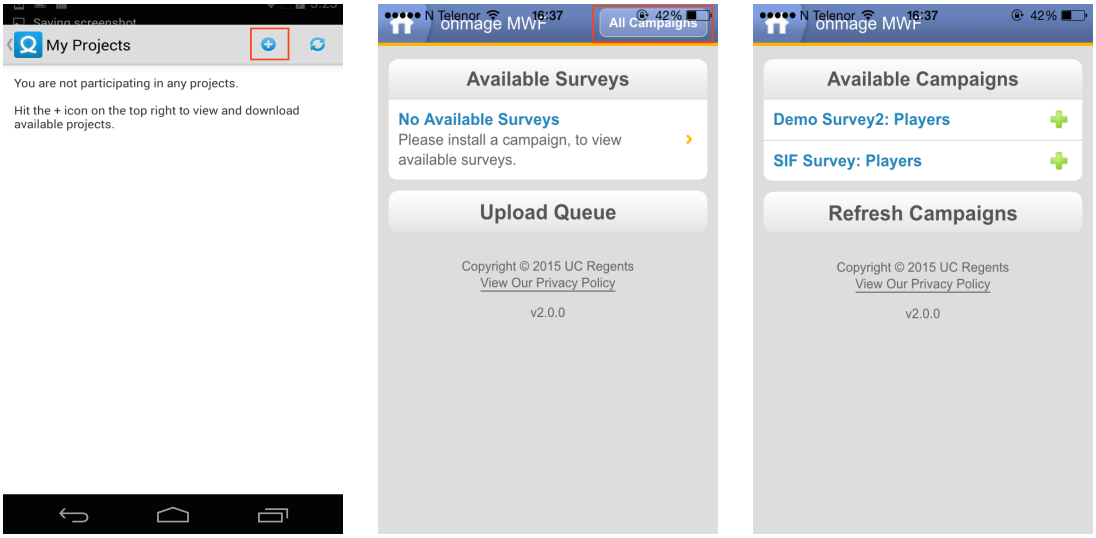
In Ohmage standalone was the campaigns feature called Projects, and on iOS was called Campaigns. In order to conduct a survey, we need to download the campaigns we want to conduct. In both applications were the approaches with download of campaigns the same. We click on the feature, and then, we need



(a) Dasbhoard screen, Ohmage MWF (iOS) (b) Dashboard screen, Ohmage MWF (Android)

Figure 3.5: Screenshot of Ohmage dashboard screens

to add available campaigns before we can conduct the surveys. Both approaches are briefly presented in Figure 3.6.

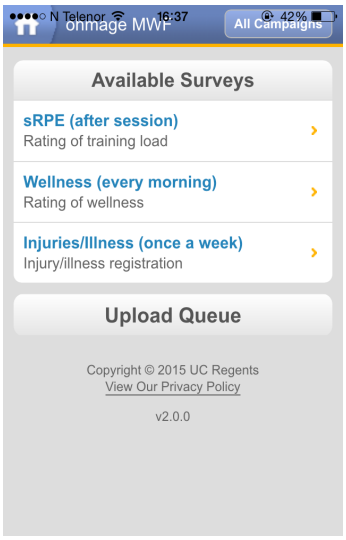


(a) Empty project list (Android) (b) Empty project list (iOS) (c) List of available campaigns (iOS)

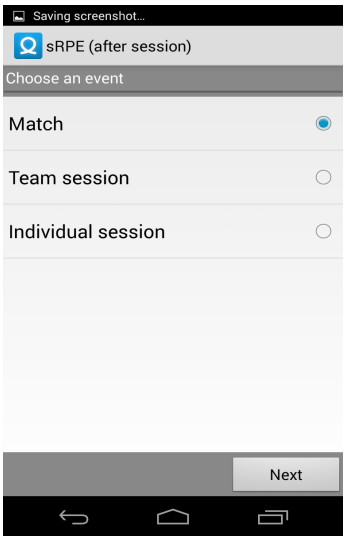
Figure 3.6: Adding a campaign on both applications

3.3.3 Surveys

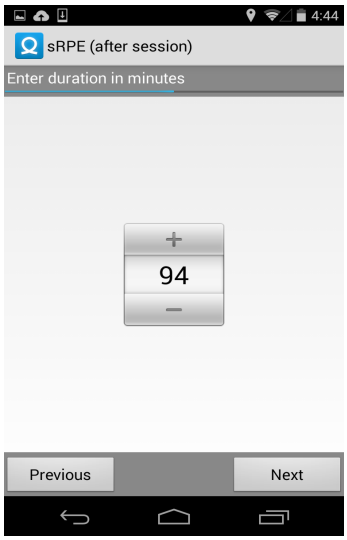
When we click on the survey function, we get to choose between surveys we want to conduct. A user can choose between RPE, Wellness or injury report. Both Ohmage applications support most of the prompt types, but within the three surveys we have presented, we have only used single-choice, number and timestamp. The method for reporting a survey were the same on both applications, we pick a survey, and are presented a question, and various options. After we have picked the option we wanted, we click the next button to move to next question. When the survey is finished, we get presented a summary text. On Ohmage MWF we were asked whether we want to share our GPS location or not, and the users were told to click no here. Because we do not want to save any data that could be used to identify a person in real life. Screenshots of the prompt types and available surveys are presented in Figure 3.7



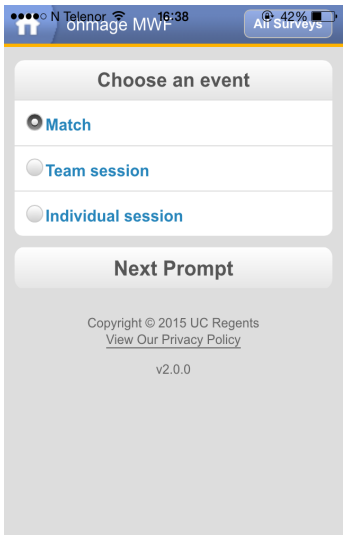
(a) Available surveys (iOS)



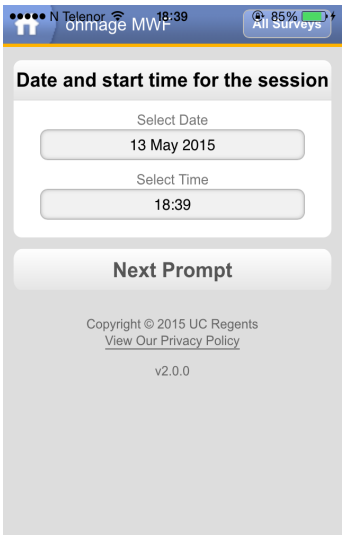
(b) Single choice (Android)



(c) Number prompt (Android)



(d) Single choice (iOS)



(e) Time stamp (iOS)

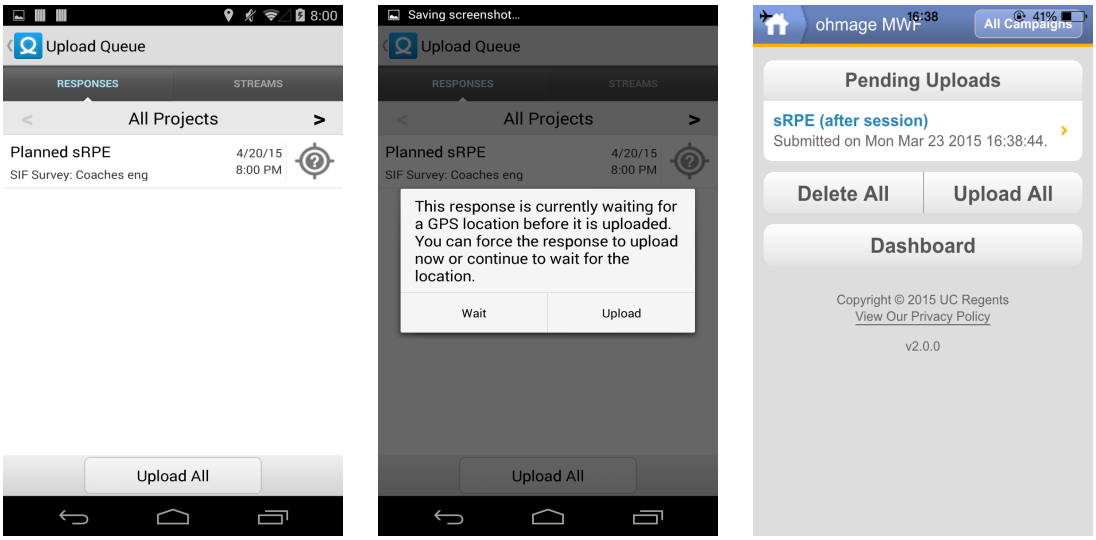
Figure 3.7: Surveys and prompts in pmSys

3.3.4 Upload queue

One of the great features Ohmage applications were providing was the ability to report offline regardless of internet connection. This allowed the players to conduct any surveys at any time. If a user conducts a survey offline, the reported survey will be placed in an upload queue in the application. The response will be stored in this queue until the user has internet connection again, and the user has to manually upload the responses. If the user has internet connection, Ohmage MWF uploads the response right after the user has finished. However, on Ohmage Standalone was the response placed in the queue regardless of internet connection, this happen because Ohmage standalone was waiting for GPS location, and since we do not want to provide our GPS location, the response will not be uploaded automatically, thus, the user must manually upload the response (see Figure 3.8)

3.3.5 Reminder

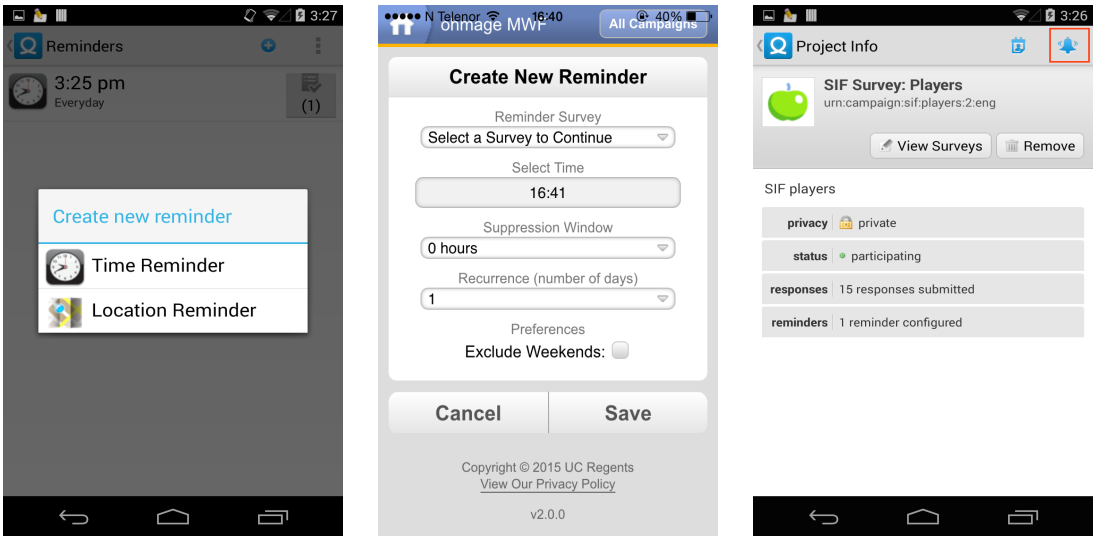
Reminders are available in both Ohmage applications. Reminder feature was used to schedule a reminder, to help players to conduct surveys. When the reminder was triggered, it came as a notification on the mobile phone. The reminder function in Ohmage standalone was hidden under campaign information, and was not easy to find. In additional, Ohmage standalone also supports location-based reminders, which let us mark a location on a map, or search for an address, and then, the reminder will trigger once



(a) One response in queue (Android) (b) Manual upload to server (Android) (c) View response before uploading

Figure 3.8: Upload queue in Ohmage

we get into our marked area (see Figure 3.9).



(a) Empty project list (b) Add reminder (iOS) (c) Reminder was hidden on Android

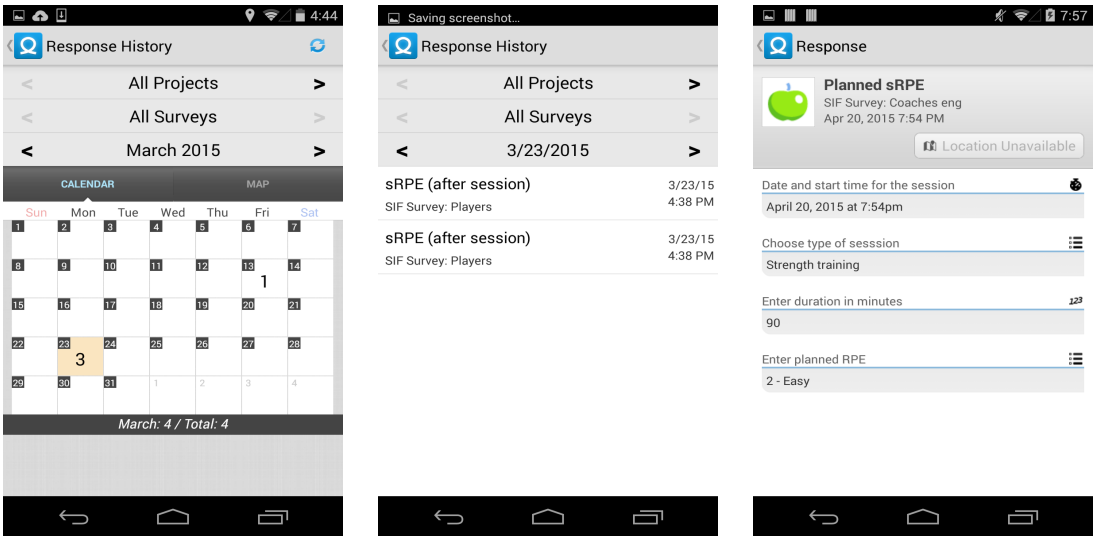
Figure 3.9: Adding reminders in Ohmage

3.3.6 Response history

Response history feature was not presented in Ohmage MWF, but only in Ohmage standalone. This feature allowed us to view our response history in detailed. We can filter based on various surveys and dates, and view each response in detailed (see Figure 3.10).

3.3.7 Evaluation of mobile application

Both applications worked as they were intended to do, which was to provide a self-report application. It was easy to execute/conduct a survey, but the process of need to add a campaign, before we were able to conduct the survey, could be removed. Hence, the application will be faster and get to the point with self-report. The procedure of executing a survey could also be improved; we could remove unnecessary interactions, and let the user finish the survey faster. An improvement towards this procedure, could have been to remove the next button, in this way, the single choice questions would be faster to finish, but also



(a) Calender (b) List of responses (c) Detailed response

Figure 3.10: Response history in Ohmage standalone

have less clicks. The number prompt could also be improved, by for example provide the possibility to do intervals on increment or decrement of a number, instead of increasing the number with 1, we could increase the number with intervals of 5 instead. Both these improvements would speed up the reporting process, but also provide fewer clicks for a user to finish a survey.

Reminder feature was an excellent feature that helped users to do their surveys. In our case, this feature was very important. In order to monitor players, we need the players report data frequently and in time. With the first version of Ohmage, one of the feedbacks we got from the coaches was: *Players wanted to report data, but even with the reminder feature, they tend to forget*. Hence, we got a new idea; what if let the coach help to remind each players to do their survey? Because the coach know who have reported, and who have not.

Response history only existed in Ohmage standalone and we found the feature relevant and useful. However, it would be better if we could provide players more useful presentation and feedback of reported data. Instead of presenting a response history, we could present the reported data in graphical charts. This way, the players can follow and monitor their own progression, and have a training diary built-in the application. We believe that providing useful feedback with visualizing can increase the motivation to use the application as more than a reporting tool.

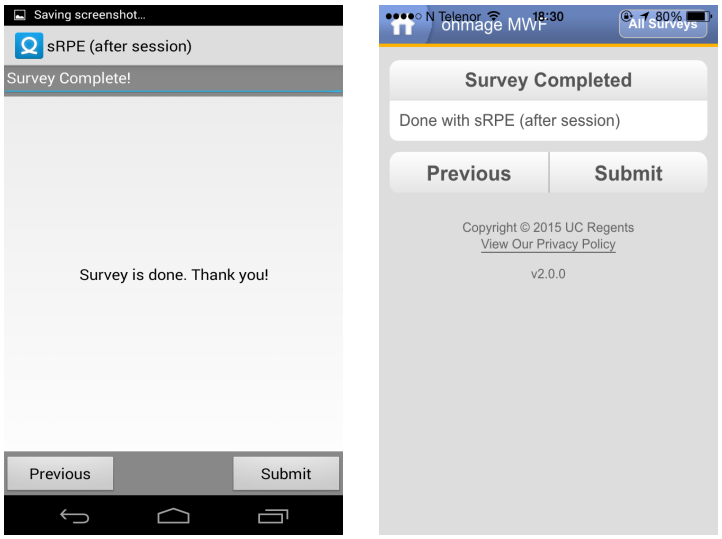
Both applications have a queue where the finished responses were placed in when the application could not upload the responses to the server. In addition, when the application has internet connection the user must perform a manual upload of the responses to the server. Ohmage standalone would not upload the responses regardless of internet connection if the GPS were not provided. We found this process inconvenient, and a suggestion to this process, is to remove the queue and use technologies to detect whether a connection is present or not, and then automatically upload the responses.

Since one of our requirements was not save any data that could be used to identify a person in reality, we must have every user to turn off the GPS acquisition in Ohmage applications. To have applications that automatically record GPS location if the user is not aware of it could seriously harm our system and at worst, break the law. We want an application that does not record any location data at any circumstances.

With the injury reporting process we presented in Section 2.2. We wanted a player to report more than once if they have more than one injury or illness to report. This could not automatically be done in Ohmage applications, the users have to manually start the survey over again if they wanted to report the second time. Ohmage does not support action based condition logic. This means we could for example define a specific condition to restart a survey based on the user's answer. We solved this in our first version of pmSys with Ohmage, by telling the user to report twice if the player has more than one problem to report.

By the end of each survey, ohmage presents a summary page with a message we defined on campaign

upload (see Figure 3.11). Instead of showing a summary message, a better approach, would have been to show a summary page of answers the user have picked. Then, the user can perform an extra check, and go back and change if they needed to.



(a) Standalone version (b) MWF version

Figure 3.11: Summary page in Ohmage applications

3.3.8 Conclusion

Both applications did what we wanted them to do, which was self-reporting through a mobile application. However, with our evaluation we just mentioned, the application could be improved and customized even more, to fit an athlete better but also perform faster. We could provide the user more useful feedback and allow the user to monitor and follow their own progression, based on the reported data. This makes the application not only a self-reporting tool, but also a monitoring application. Hence, we wanted to implement a new application that is customized and created to fit an athlete better, we also want to have on application which look the same and provide the same features on all platforms. We hope to replace both Ohmage applications with a pmSys mobile application, this application will be presented in the next chapter. Hence, with the evaluations we just mentioned, the application could be improved and customized to fit an athlete's need even better, but also perform faster.

3.4 Ohmage web portal

The web portal in Ohmage is a web application that communicates with Ohmage backend server, and fetches data from the server. Ohmage portal is written in Java, using Google's web framework, GWT (Google Web Toolkit), and runs as a web application in a Java Server servlet container. In the next subsections, we present the features Ohmage web portal provide.

3.4.1 Campaign

Before we can start reporting data, the campaigns with various surveys need to be uploaded to the server. Through the web portal, we can upload the XML defined campaign, and it will then be parsed and validated. Only system administrators are allowed perform this task.

3.4.2 Viewing responses

Users can use view their submitted responses in this portal. Only users with privileged class role (coaches) can read and delete the players' responses through the portal. A screenshot from the portal

is presented in Figure 3.12. In this figure, we can see a filter to the left, where we can specify start/end date, and filter the responses based on users and surveys.

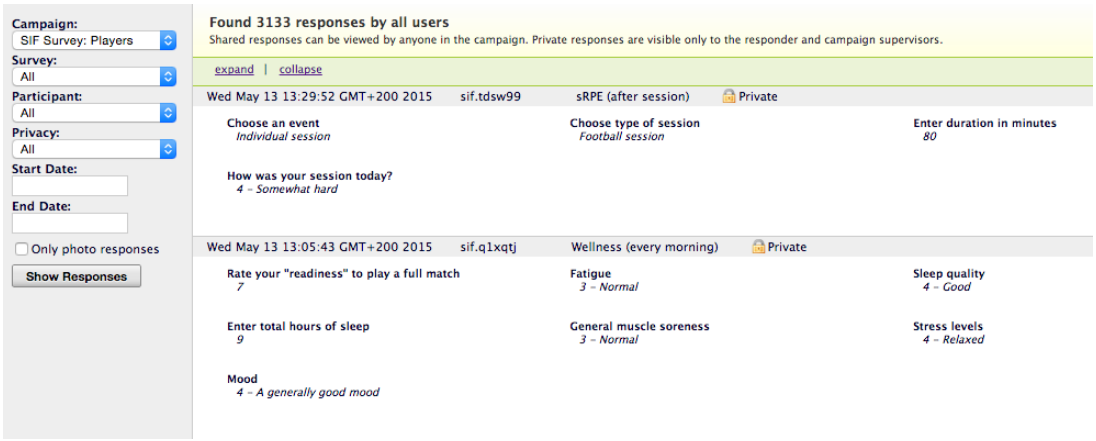


Figure 3.12: An example of viewing responses through the web portal

3.4.3 Export responses

The portal also provides a possibility to export all saved responses. The data is presented in a CSV[1]. The coaches and coordinators used this function frequently to generate graphical charts through external programs, such as Excel, to present the data, but also to analyze and evaluate the data afterwards. This CSV file was not created to be open in Excel, and was very messy, the coaches/coordinators needed to organize and process this file manually afterwards. This workflow of processing and generating charts manually was not convenient, and was not any different from using pen and paper and then, manipulate captured data in Excel afterwards.

3.4.4 Visualization

The visualization feature in Ohmage, was based on statics visualization. Ohmage provide five types of visualization, Total responses, Single variable and Multi variable. Total responses is visualizing based on number of responses the users have uploaded to the server, but also present a leader board for responses (See Figure 3.13). The Single variable visualization, presents graphical plots based on a single prompt over a time. The last visualization type let us present two different variables into one graph, and provide a scatter plot and a density plot (See Figure 3.14).

We did not found the charts any useful for monitoring and analyzing a player. The coaches wanted graphical charts, where the numbers were calculated before presenting it. For example, we could visualize a player's load based on the reported workload multiplied with the duration, in this way, we provide a better graphical analyzing feedback of the player's load, and would be easier to interpret and compare various numbers for one player against another. The feedback from the coaches who used the first version of pmSys, said that they wanted the portal to visualize data, that can be used to monitor load, wellness and injury for a player, rather than statistics on total responses and who have reported what. We want to give the coaches better presentation of collected data.

3.4.5 User management

The user system in Ohmage was one of the main reasons we wanted to use Ohmage-platform as a player monitoring system. The web portal provided an easy and very powerful user management, to create users and assign various classes and campaign permissions. We could also change password for the users through the portal. In Figure 3.15, we presented a screenshot of two classes in the portal.

[1]Comma separated values

Username	Total Responses	Private Responses	Shared Responses
sif.0w616j	130	130	0
sif.1396v0	127	127	0
sif.2bz8jw	121	121	0
sif.2cvdop	128	128	0
sif.32hody	74	74	0
sif.4nr1wd	84	84	0
sif.62ipsi	128	128	0
sif.7ylan0	147	147	0
sif.a0zq3z	3	3	0
sif.ai3la9	142	142	0
sif.bq7l0e	95	95	0
sif.eluew0	130	130	0
sif.h2dfbg	171	171	0
sif.jxrokk	147	147	0
sif.l1so7g	143	143	0
sif.lwa7z0	136	136	0
sif.n9czhb	139	139	0
sif.q1xqtj	129	129	0
sif.r4chyg	119	119	0
sif.rosclp	109	109	0
sif.rv231e	85	85	0
sif.t7qif3	18	18	0
sif.tdsw99	152	152	0
sif.un6qw5	65	65	0
sif.uztl6m	143	143	0
sif.zhkf16	121	121	0
sif.zxy0ou	147	147	0
Total (All Users)	**3133**	**3133**	**0**

Figure 3.13: Leaderboard in web portal

3.4.6 Hard-coded endpoints

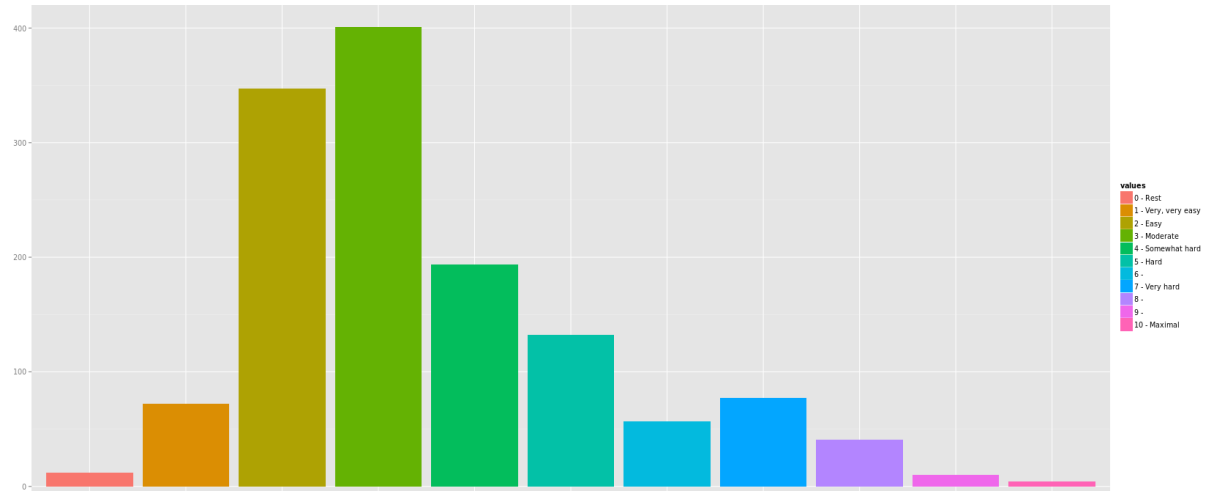
We analyzed the front-end's source code, and we found all the API endpoints were hard-coded and pointed to the URL *app*. Which means Ohmage backend server have to be hosted at the same server as the web portal, and must be hosted under the URL path, app. Otherwise would the front end not work (See Figure 3.16). It would be easier and more portable portal, if we could pick the endpoint server our self, in case we want to run the Ohmage-server somewhere else. For example, we could host Ohmage back end server in the cloud, and have the web portal hosted on a local server.

3.4.7 Conclusion

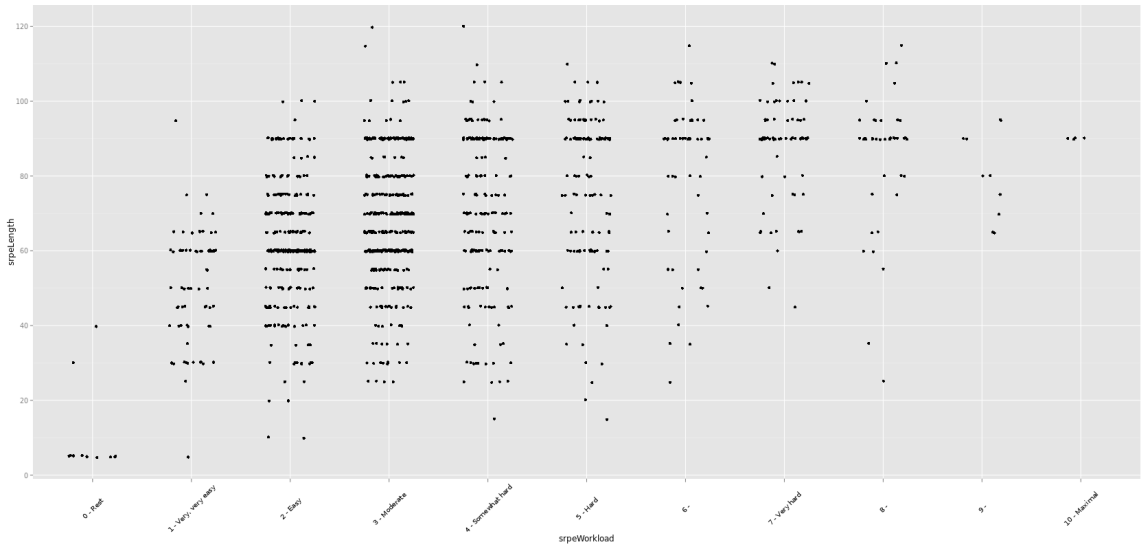
We conclude that the web portal worked as a user management tool, because it provided an easy and clean user interface, for user manipulation and uploading a campaign. However, except from these two features, the coaches did not found the web portal very useful as a monitoring application for their players. With the first version of pmSys, the coaches exported the data from Ohmage, then, organized the data by themselves, and then, generated graphical charts to present and analyze the data afterwards. With a new web portal that is created for analyzing and presenting data, where the coach can monitor and evaluate the reported data. We also wanted to investigate whether it was possible for the coach to communicate with a player directly from this new web portal.

3.5 Summary

Ohmage was a great system for data collecting with self-report, but was not optimized and created for monitoring sports athletes. We decided to keep Ohmage-server because of the stability and user system Ohmage provide, and build new application around Ohmage backend server. We found both Ohmage mobile applications great as reporting tools, but again, they could be better by providing feedback through visualization of reported data. The application could also have a faster report process, by removing unnecessary user interactions. By giving a better feedback on reported data by visualizing, we believe this approach can increase the motivation for the user to report, but also help the user to monitor their own progression. We want to replace both existing mobile applications for the players, and create one optimized and customized reporting application the users.

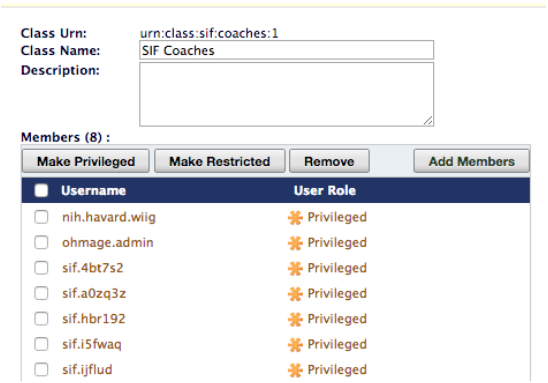


(a) Response distribution



(b) Scatter plot

Figure 3.14: Visualization in web portal



(a) Coaches

(b) Players

Figure 3.15: Class administration in web portal

We also want to create a new portal for the coaches, to work as a better analyzing and monitoring tool. The Ohmage web portal only presented visualization on statics for the collected data, but we could with a new portal, present the data that is processed and analyzed before it is presented on graphical charts. Furthermore, the coach can focus on analyzing and interpreting data, rather than processing data.

```
public class AwConstants {
    private final static String releaseServerLocation = ".."; // same as web server
    public enum AwUri {
        /* /app/ is hardcoded*/
        AUTHORIZATION("/app/user/auth_token"),
        LOGOUT("/app/user/logout"),
        APP_CONFIG_READ("/app/config/read"),
        USER_CHANGE_PASSWORD("/app/user/change_password"),
        USER_CREATE("/app/user/create"),
        USER_DELETE("/app/user/delete"),
        USER_INFO_READ ("/app/user_info/read"),
        USER_RESET_PASSWORD("/app/user/reset_password"),
        CAMPAIGN_READ("/app/campaign/read"),
        SURVEY_RESPONSE_READ("/app/survey_response/read"),
        CLASS_READ("/app/class/read"),
        VISUALIZATION_URL("/app/viz"),
        WHO_AM_I("/app/user/whoami");
        /* And more..... */
     /* Check https://github.com/ohmage/gwt-front-end/blob/master
     /src/edu/ucla/cens/mobilize/client/AwConstants.java */
 }
}
```

Figure 3.16: Hardcoded Front-end endpoints

In addition, we want to see whether it is possible to have a communication channel between coaches and players. In the next three chapters, present a new implemented pmSys mobile application first. Secondly, we present a new pmSys web portal created for the coaches. At last, we present optimizations towards the backend server, and we have tested our system hosted in various cloud services.

Chapter 4

The pmSys mobile application

In this chapter, we present the new mobile application we implemented as a replacement for Ohmage mobile application. We are going to present technologies we used to implement this new application and the improvements we talked about in the previous chapter. We also present two user studies we performed to confirm whether pmSys mobile application was better or not. Then we present the result of the new implemented application.

4.1 Motivation

In the first version of pmSys, we distributed the system with Ohmage mobile applications as a self-reporting tool. With the improvements of Ohmage applications from the previous chapter, a new mobile application was highly demanded. We wanted to improve the reporting process for our users. We also wanted to motivate the players to use the application by visualize captured data with graphical charts, instead of showing raw response as in Ohmage.

The players tend to forget to conduct their survey, especially after training and in the mornings. Providing a possibility to add reminders, helped to a point, but not everyone was able to, or knew how to add reminders. What if we could have the coach to send instant message to their players, to remind each player of conducting their survey. What if we could bring this idea further, by letting the coach create a scheduled reminder that can automatically send a message to all the players who have not conducted their survey and remind them.

By using two different mobile applications, we have to learn two different applications, but what if we could deal with only one application, that looks and function the same on every device. Due to our limited time, we decided to develop an application that is platform independent, and followed the principle *build once, deploy everywhere*. In the next sections, we present all technologies, frameworks and tools we used to develop the new application. We also present the result of our new application compared with Ohmage applications.

4.2 Design

In this section, we present technologies and design decisions made for our application. We are going to talk about native application versus hybrid application. Moreover, we present the frameworks and tools that we have analyzed, and used to implement the new application.

4.2.1 Mobile application development

In the mobile application development context, it exists three categories for development, native, hybrid or web application. Considering which one of these three to use is also a dilemma [6]. In mobile application development, it exist five common challenges:

Multiple standards In mobile application development it iss hard to only deal with one standard, because of the existence of multiple standards and platforms. For example on Android devices,

even on the same platform(Android), various smart phones operate with different hardware specifications, such as RAM, CPU, Screen Size and Storage capacity. The same problems exist on iOS devices as well.

Lack of analysis tool Mobile application development has limited support for automated testing in native mobile application development. For example testing sensors such as location service, or multi gesture touch. There is need for better analysis tools for measuring and monitoring various metrics of applications and testing.

Development platforms divide For example on Android, which is an open source mobile operating system, it exist various Android versions. Some manufactures modify the OS source code, and that results in multiple standards, which then result in limited portability from one android version to another. However, iOS and Windows platform have their source code closed for the public, thus platform divide is not an issue. In the other hand, iOS and windows application development, requires IDE[1] skills and knowledge, which is not required in Android.

Frequent version releases Every operating system is releasing new versions of their operating system frequency, see Table 4.1, to fix bugs and secure their operating system. This creates different challenges for the developers. Developers must learn different programming languages for each platform, and new API release for every new versions. In additional does the developer have to keep up with new software development kit (SDK) updates.

Limited storage capacity Most smart phones have limited storage and can be full over time. With use of network connection, we can store data at distributed servers, then limited storage capacity is not such challenge anymore. But at some point, the application might need to store data on the phone's storage, when the phone does not have internet connection.

All presented challenges is taken from the article by Phyo Min Tun [6].

Released year	Android	iOS
2007		iOS 1.0 iOS 1.1
2008	- Apple pie	iOS 2.0 iOS 2.1 iOS 2.2
2009	- Banana Bread - Cupcake - Donut - Eclair	iOS 3.0 iOS 3.1
2010	- Froyo - Gingerbread	iOS 3.2 iOS 4.0
2011	- Honeycomb - Ice cream sandwich	iOS 4.3 iOS 5.0
2012	- Jelly bean	iOS 6.0
2013	- Kitkat	iOS 7.0

(a)

	Android	iOS
Language	Java	Objective-C/Swift
Tools	Android SDK	Xcode
Package format	.apk	.ipa

(b)

Figure 4.1: Mobile operating systems release [6]

Native mobile application

Native mobile applications are installed directly on the mobile phone, and can be launched without any container or intermediary tool [6]. Native applications can all access the phone's operating system API's and use built-in services, such as camera, accelerometer or location services (GPS). However, developing native applications require deep knowledge of the specific platform. When building native applications, we have access to native API and UI, and that gives us the ability to use the benefits of device specific

[1] IDE - Integrated development environment

software and hardware. One of the biggest disadvantage of building a native application, is that the source code written for one platform, cannot be used on another platform, for example source code written for Android cannot be used on an iOS platform. Native application development pattern is also time and cost inefficiencies, due to duplicate effort for development and maintenance. Another disadvantage of building native application, is every platform require a platform-specific SDK, and we are forced use their unique tools. Native applications needs to be distributed through the platforms application store, the advantage of this is that we do not need to distribute our application directly to our users by ourselves. However, in order to apply and distribute our application to a such store, we need to go through a large amount of policies and rules, before our application gets approved, and can be uploaded to their store.

Web application

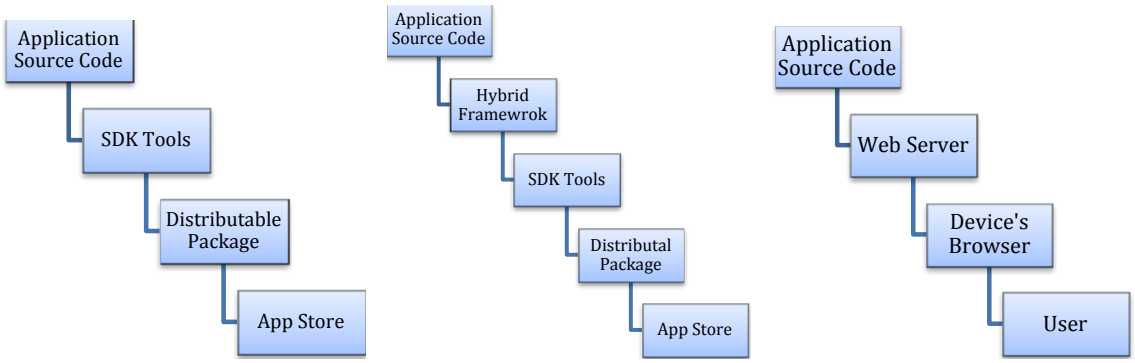
It exist two forms of web applications, one where the view is rendered on server side, and then sent to user's web browser. The other approach is by moving the server rendering to client side, and the client only focused on pulling relevant data from a server afterwards. Both approaches are using HTML, CSS, Javascript to build applications. Either way of doing it, the application is accessed through the mobile device's web browser. An advantage of using this approach is that the web application is less complex, inexpensive, faster to build and easier to maintain [6]. Hosting source code on a distributed server enables developers to continuously update the application, without going through submission and approval to an application store. Another advantage of using this approach is the support for platform independent, the only requirement is a web browser. Disadvantage with using this approach, is when the device need to work offline, without connection. Without connection to the server, the user cannot interact with the application or get presented any content, because the content it's stored on the server. Another disadvantage of creating a web application, it the native API and other features the phone is providing, is not accessible through a web application.

Hybrid application

Hybrid mobile application combines both native development and web technologies [6]. Hybrid applications look very similar to a web application, because it is using web technologies to present the content, but the application is wrapped inside a native container. By doing this, the application can use all native features the phone is offering. To use this approach saves development time and maintenance cost, because the source code can be reused on multiple platforms. For example, by using a hybrid application framework, such as Cordova, which is an open source framework. Cordova provides JavaScript interface to access native features on the phone. Disadvantage of developing hybrid application is that, it cannot be developed independently of the OS, and needs to be wrapped in or used with a native container. The native container gives access to native features, but the latest user interface might not be fully supported. The native performance cannot be achieved with a hybrid application, since it is wrapped in a container, and uses the phone's browser to render content.

Conclusion

Due to our time limit, we limit our support to only create application for Android and iOS devices. We picked the Hybrid application approach, because we want to follow the principle *build once, deploy anywhere*. Thus, developing a hybrid application with web technologies gives the other users who do not have an iOS or an Android device, to use the application from a web browser, without mobile device features. In addition if we went for native application development, we have to learn the platform specific language, for example: java for Android phones and Objective-C for iOS devices. But that is not all, we have to develop two application in parallel. We did not go for web application approach, because of the need of accessing the phones native API. In order to receive instant messages, sent from a coach, and storing and report surveys offline, the native API was needed. Thus, was hybrid application the most optimal approach for creating a new application. The three development approaches are illustrated in Figure 4.2



(a) Native application (b) Hybrid application (c) Web application

Figure 4.2: Mobile application development approaches [6]

4.2.2 Apache Cordova

Apache Cordova is a hybrid mobile application framework that allows the developer to access native device function such as camera or accelrometer in Javascript. Cordova applications are built with web technologies such as, HTML, CSS and Javascript. Cordova provides application development without need to deal with any native code(Java, Objective.c etc), but only with use of web technologies. Cordova is using the platform's SDK to wrap the developed application into a native container. This makes the application able to be distributed through the platform's application store, and use native features. Cordova have large support for platforms such as iOS, Android, Blackberry, Windows phone and more [35]. All native features is accessed through use of plugins, Cordova have already some basic plugins. It exists more than 900 Cordova plugins [36], and the plugins are built in native code and it's platform dependent. If we found a plugin we wanted to use, we have to check whether the platform was supporting the platform, we are developing for.

4.2.3 Single Page Application

Since Cordova is using web technologies to build an application, we use the single page application approach to create our application. According to *Single Page Web Application* by *Mikowski and Powell* [7], *"An Single Page Application(SPA) is an application delivered to the browser that does not reload the page during use"*, by reload, means that the page is not refreshed on user interaction. This approach gives the user a native feeling of using the application. Also avoids view rendering on server side, and we move the view rendering over to client side. By doing this, we are reducing the data transferred between the client and the server, and let the server focus on sending relevant data, rather than how buttons and other component should look like. Figure 4.3 illustrates how SPA application works in theory, and how the business logic and HTML rendering is transferred to client side, and the server is focusing on sending and receive data.

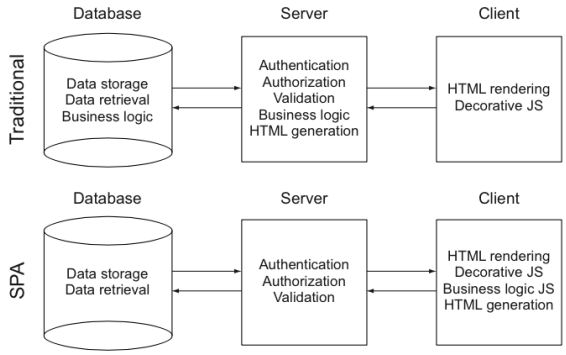


Figure 4.3: Tradional web application vs Single Page application from Manning Page 8 [7]

4.2.4 Software design pattern

Building mobile application is challenging in general. Structuring and organizing is yet another challenge in mobile application development, especially when we work in team. Model-View-Controller is an application structuring principle for implementing interactive application components [37]. MVC consists of a three-way layer to structure components: [37]

Model The model represents application domain, state and behavior.

View The view deals with what the users see, basically a graphical representation of the model.

Controller The controller handles the communication between the model and the view.

We have tried to follow this software application pattern in our application development.

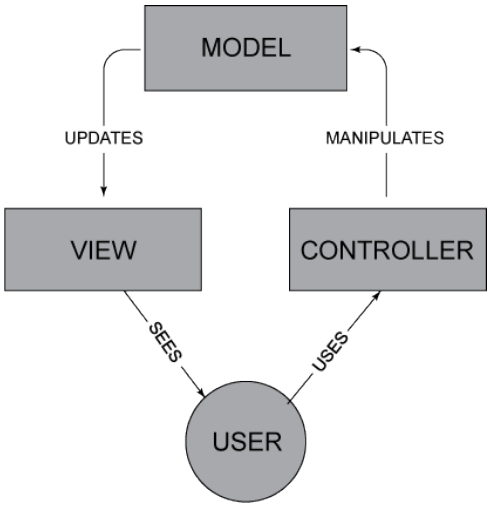


Figure 4.4: MVC figure [8]

4.2.5 User interface

Due to our lack of graphical design experience and background, using web technologies can ease some of the work needed to build a new application. We can for example, take advantage of the existence of various UI frameworks available, to compose the application. A well-known UI framework is Bootstrap, it provides pre-designed components like buttons, headers, tables and more. These components can be easily customized, with use of CSS. Bootstrap is created with mobile first principle, which means the components in Bootstrap are optimized for mobile phones[2]. However, the framework is designed to create web application running in the web browser, rather than mobile phone only.

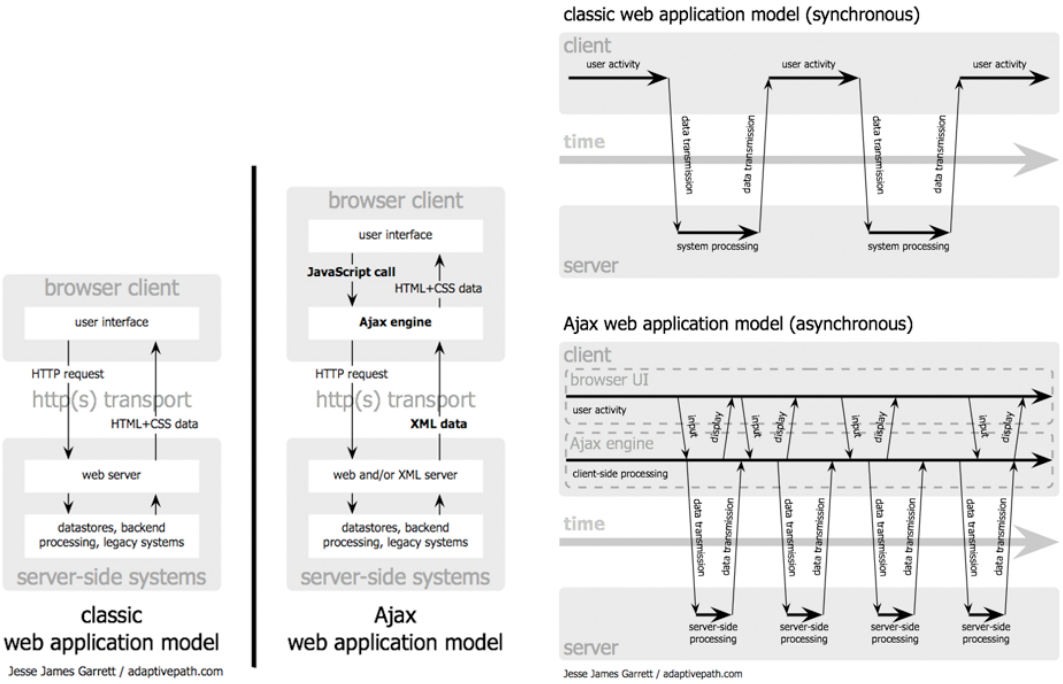
We want an UI framework that can offer native user interface, and the buttons and menus in the application to feel native. Ionic framework is a hybrid mobile application framework, that is focused on building native-feel mobile applications, by having components look and feel native [38]. The reason we picked Ionic to build our new application with, was because most of our users have iOS mobile phones, and Ionic provided UI components that look very like native UI component, and the component gave a native impression.

4.2.6 JavaScript

With HTML and CSS to present the look of the application, JavaScript is used to handle interaction between the application and the user. JavaScript can also handle communication from a client to a server by using *Asynchronous JavaScript and XML* (AJAX) programming model. On traditional web application, when the user clicks on a button, the client (web browser) sends a request to the server (web

[2]Bootstrap - `http://getbootstrap.com/`

server), and then the user have to wait for a response, and when the server is ready to respond, the client get redirect to a new URL/address. By using AJAX programming model, the user gets a feel that the page does not wait for a response or need to be refreshed/redirected [9], and make the application feel native when running on a mobile phone. An illustration of AJAX application model, and how AJAX handle the communication between a client and a server is presented in Figure 4.5, this approach is similar to what we presented in Section 4.2.3.



(a) Ajax web application

(b) Synchronous vs Asynchronous

Figure 4.5: Ajax programming model [9]

AngularJS

The UI framework we mentioned in Section 4.2.5 Ionic, is built with AngularJS, and depends on AngularJS to function. AngularJS is Open source software and a JavaScript framework, maintained by Google. Some of the features AngularJS is providing [39]:

Two-way binding gives automatic synchronization between the view(HTML) and the model(Data).

Structure for front end code by dividing the code into services, controllers and views.

Routing support we can define our own routes, and make the application feel native, and do not need to rely on a web server to provide us correct resources.

HTML template syntax we can write expression in the HTML files, and then the view is rendered based on defined conditions.

Form validation We can for example ask a number presented in a form, and then, Angular can validate the numbers written in the form.

HTML directives We can teach AngularJS new self defined HTML directives/syntax.

Dependency Injection One of the greatest feature with Angular, it provides dependency injection across all services and controllers.

It exists other frameworks that are similar to AngularJS, such as EmberJS and Backbone, but since Ionic need AngularJS to function, we did not evaluated other JavaScript frameworks, and used AngularJS as the core JavaScript framework in our application.

Supporting JavaScript frameworks

X2JS When we fetch a campaign from Ohmage server, the data from the server is specified in XML format. With a JavaScript library named X2JS[3], we can convert XML notation into JavaScript notation. This library make the processing and working with the campaign and XML a lot easier.

StringJS Working with text-strings is yet another challenge in JavaScript. StringJS[4] provides every missing string manipulation features in JavaScript. For example, native JavaScript does not support for whether a string contains in another string.

Underscore With UnderscoreJS[5] we extended the native JavaScript containers with new useful feature and functions, for example, sorting and finding various elements in big data collections.

MomentJS We used MomentJS[6] to handle time and date in JavaScript, this library allowed us to parse, manipulate, validate and displayed advanced formatted time stamps in JavaScript, and the library was very easy to use.

NVD3 [7] is a library that is built upon D3[8], and offers advanced and reusable charts with web technologies, such as HTML, CSS and JavaScript. We used this library to present all graphical charts.

ngTouch When developing a web application, most of mobile web browsers wait about 300ms after tap-and-release before sending a click event [40]. By using ngTouch library from AngularJS, we can remove this delay, and the click event will respond immediately.

Angular UI Router [9] was another core library that was used in our new application. AngularUI Router allowed us to translate URL routes into various states, instead of writing long URLs.

4.2.7 Conclusion

Our new application was built with all presented theory, design patterns, technologies and frameworks together. We decided to build a Hybrid mobile application, due to our time limit, and we needed to support Android and iOS devices. We used AngularJS as the core JavaScript framework in the application. Ionic as the HTML and CSS framework, to provide native UI. We used Single-Page-Application and AJAX programming model to make the application feel native and communicate with a server. In the next section, we present how we used these frameworks and technologies to implement the new pmSys mobile application.

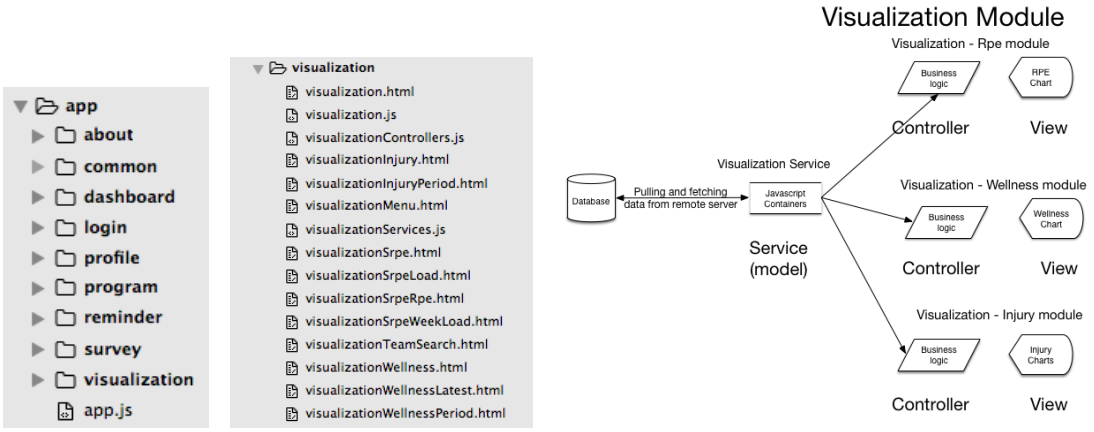
4.3 Implementation

With help of the various javascript frameworks mentioned above, and the existence of cordova plugins, we used all technologies combined to build our application. With cordova wrapping our application in a native container. The main JavaScript framework in our application was AngularJS. We then divided the file structure in the application into module-based folders. For each module in the folder, we have one or more views, a service and a controller. A service is corresponds to the model in MVC architecture and the service layer is responsible for pulling data from the server, and store it locally in a JavaScript containers.

We made a module-based mobile application by dividing each feature into separate modules and folders. In Figure 4.6, we can see that each module/feature is represented by a folder, and within each

[3]XML2JSON - `https://code.google.com/p/x2js/`
[4]StringJS`http://stringjs.com/`
[5]UnderscoreJS - `http://underscorejs.org/`
[6]MomentJS - `http://momentjs.com/`
[7]NVD3 - `http://nvd3.org/`
[8]D3 - `http://d3js.org/`
[9]AngularUI Router`https://github.com/angular-ui/ui-router`

folder have many different view files. The reason for it does not exist as many controller files and service files as view files, is because within a module, we have only one service that is distributed between all controllers and other services, all controllers are stored and defined in one file. However, we have used AngularJS's dependency injection, to injecting every service into each controller.With dependency injection enabled in AngularJS, we can reuse the services over and over again.



(a) Screenshot of the application folders

(b) Screenshot of the visualization folder

(c) Example of our MVC application structure

Figure 4.6: Module based development

4.3.1 States and routes

Usually, when building web application, the content is presented, based on what the user have requested from the server. However, when the application is running inside the Cordova container on a mobile phone. The resource is accessed as a file system. There is no existent of an address bar where we can type the address we want to access. All resources and files is resolved by the Cordova container, using the mobile browser's web view. The start page of a hybrid application in Cordova normally index.html, unless we specify something else. If we want to navigate another file, we have to create a button with a link to access the other file, and the web view will then resolve the request. The Cordova container works like a web server, serving user content based on requested resource from the user.

We wanted the application to feel native, and with Cordova handling the requests like a web server, would not provide the user a native feeling of the application. Angular has a built in module for routing and linking, ngRoute. This module allows us to display content based on which resource we want to see, and with each view presented, it also provides a controller and a service on requested resource. Figure 4.7 example[10] shows how easy it is to define routes and assign controllers and views to various routes.

When building large applications, the URL's become very long, and challenging to maintain. Especially when we sends long queries through URLs. With Angular UI-Router, we can define child views and parent views. Instead of working with URL's, Angular UI let us work on states instead long URL's. We only need to specify various states and resources, and the library will handle requested resources. An example from our application, where we used AngularUI routes, to define various routes a user can access, is presented in Figure 4.8. In Figure 4.9, we see an example of three different state and how the URL is still unchanged. Another example of AngularUI, is how the three states menu, teamSearch and wellness is inheriting from the parent view visualization in Figure 4.10. The state module is working like a tree structure.

[10]Example taken from `https://scotch.io/tutorials/single-page-apps-with-angularjs-routing-and-templati` - Accessed: 2015-03-30

```
    // create the module and name it scotchApp
    // also include ngRoute for all our routing needs
    var scotchApp = angular.module('scotchApp', ['ngRoute']);
    // configure our routes
    scotchApp.config(function($routeProvider) {
        $routeProvider
            // route for the home page
            .when('/', {
                templateUrl : 'pages/home.html',
                controller  : 'mainController'
            })
            // route for the about page
            .when('/about', {
                templateUrl : 'pages/about.html',
                controller  : 'aboutController'
            })
            // route for the contact page
            .when('/contact', {
                templateUrl : 'pages/contact.html',
                controller  : 'contactController'
            });
    });
    // create the controller and inject Angular's $scope
    scotchApp.controller('mainController', function($scope) {
        // create a message to display in our view
        $scope.message = 'Everyone come and see how good I look!';
    });
    scotchApp.controller('aboutController', function($scope) {
        $scope.message = 'Look! I am an about page.';
    });
    scotchApp.controller('contactController', function($scope) {
        $scope.message = 'Contact us! JK. This is just a demo.';
    });
```

Figure 4.7: Example of routing in AngularJS

```
var app = angular.module('pmSys');
app.config(function($stateProvider, $urlRouterProvider){
  $stateProvider
  .state('visualization', {
    url: '/visualization',
    templateUrl: 'app/visualization/visualization.html',
    controller: 'VisualizationController'
})
  .state('visualization.menu', {
    parent: 'visualization',
    params: ['urn'],
    templateUrl: 'app/visualization/visualizationMenu.html',
    controller: 'VisualizationMenuController'
})
  .state('visualization.teamSearch', {
    parent: 'visualization',
    params: ['urn'],
    templateUrl: 'app/visualization/visualizationTeamSearch.html',
    controller: 'VisualizationTeamSearchController'
})
  .state('visualization.wellness', {
    parent: 'visualization',
    params: ['urn'],
    templateUrl: 'app/visualization/visualizationWellness.html',
    controller: 'VisualizationWellnessController'
})
  .state('visualization.wellness.latest', {
    parent: 'visualization.wellness',
    params: ['urn'],
    templateUrl: 'app/visualization/visualizationWellnessLatest.html',
    controller: 'VisualizationWellnessLatestController'
});
}
```

Figure 4.8: AngularUI state routes example

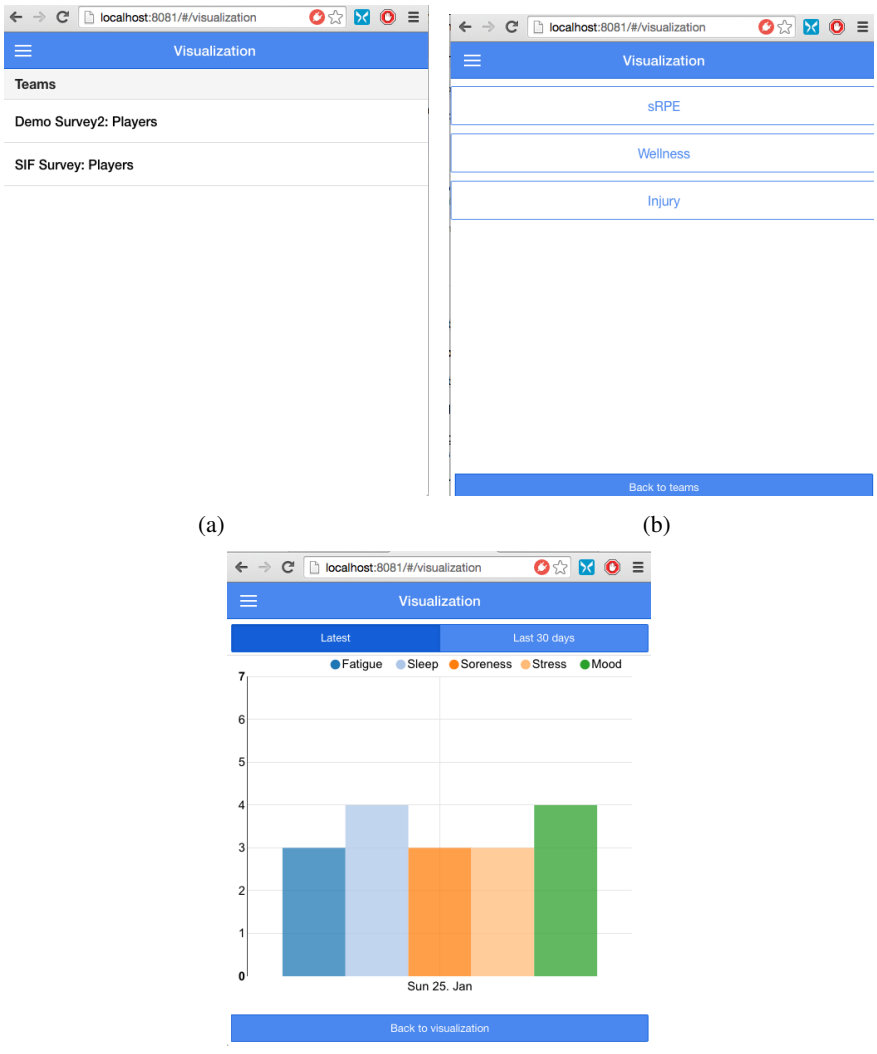


Figure 4.9: Different states, same URL

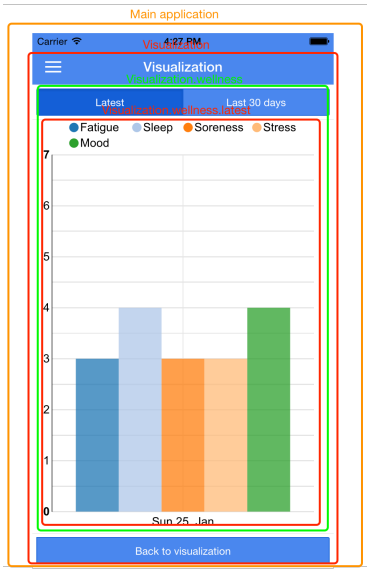


Figure 4.10: Illustration of Angular UI parent and child view

4.3.2 Campaigns and surveys

In Ohmage, we have to manually download the campaigns we wanted to conduct, and then it will be stored locally on the phone's storage. Our application, on the other hand, handle this download

process differently by downloading all available campaigns automatically, right after user have logged in. Furthermore, the campaings are stored locally on the file storage on the phone. When the user want to report data on a later point, the campaigns are always available. The user can force application to download the latest campaign and survey updates from the server. By downloading all campaigns and store if afterwards automatically, does not need the user to manually download the campaign, as we have to do on Ohmage.

The campaigns, which are downloaded from the server, are received in XML format. In the previous section, we mentioned a library that helped us convert XML to JavaScript notation *X2JS*. One downside we have with this library was how it interpreted single elements in XML. For every single element in XML, it converted the single element into a single variable in JavaScript. However, we wanted all elements in XML, to be converted to arrays in JavaScript, independently from from the XML element. As we can see on Figure 4.11, a single element is converted to an single variable, but multiple elements get converted to arrays. According to X2JS documentation, we could specify which key, we want to be converted to an array before it got parsed [41]. We were not aware of this, and we implemented our way to convert the single variable in JavaScript, to an array element in JavaScript.

Figure 4.11: X2JS converting XML to JSON

4.3.3 Report for yesterday

RPE survey should be reported as closed as possible after ended session, this is to get most accurate data on how an athlete actual felt about the session. A problem we had with Ohmage application, was when a player forgot to conduct their survey after a session, but remembered to do so the day after. With Ohmage, they could not report the day after, because the data will be submitted as the same day they are reporting. As soon as the clock passes twelve o'clock midnight, the data is reported as the day after.

We implemented a new feature allowing the player to mark the RPE or wellness report as the day before. The players can mark their response as *yesterday's* report, before they submitting the data. The reason we are allowing a player to report for yesterday is, NIH wanted to limit the maximum exceed days to be maximum one day. They meant that waiting more than a day to report data does not give accurate enough data. This option is only presented when a player executes RPE or Wellness survey, and not for injury survey. The result is presented in Figure 4.12. The reason injury does not need this feature, is because injury reporting is valid for a week, and it's not dependent to report on a certain day.

4.3.4 Injury reporting

We are following the methodology discussed in Section 2.2, where an injury or illness reporting, should be reported twice or more if he or she have multiple injuries or illness. For instance if we felt pain in both knee and arms, how could we report this in Ohmage? We told every users in person to remember twice or more, and they just have to remember it. We also added an extra prompt in Ohmage, where we

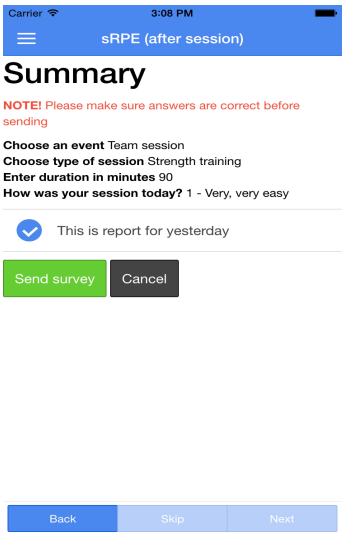
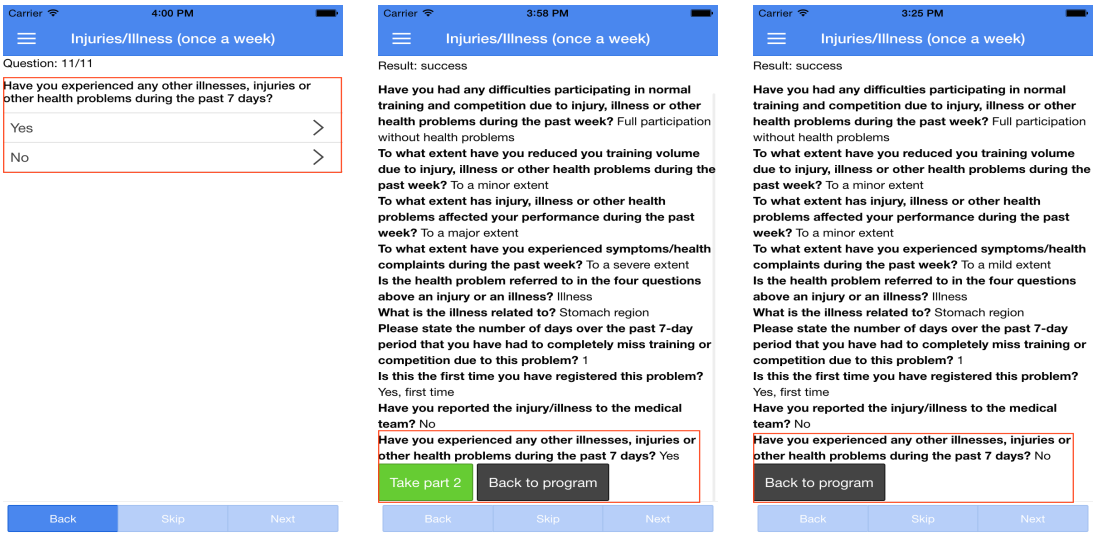


Figure 4.12: Summary page with option to report for yesterday

told every player to report twice if they have more injuries or illness. Among the injury question in the surveys, we have a question where we ask *"Do you have more injuries you want to report? Yes or No"*. It would be better if we could restart the survey based on what the users pick here.

Ohmage application and back end, have/has not support for performing an action based on a conditional logic, which means we can not tell the application to perform an action, based on what the user choose. Therefor, this feature must be implemented in the application. In pmSys mobile application, we checks the finished response, and look for whether it is an injury report or not, and then we look for a prompt with id, *ouOtherInjuries*. And based on what the user picked, yes or no, the summary page present a button to take part two after submitting (see Figure 4.13).



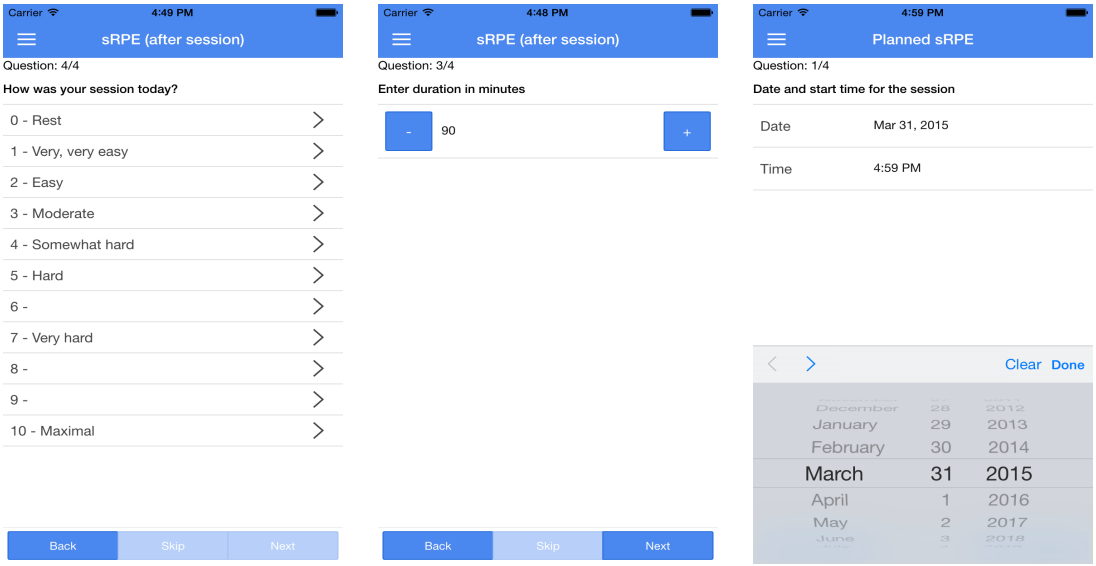
(a) Injury question (b) Take part 2 (c) No part 2

Figure 4.13: Reporting injury

4.3.5 Faster reporting

In pmSys, we are only using three prompt types, single choice, number and timestamp. An example of all implemented prompts are presented in Figure 4.14 On single choice questions we have removed the next button that existed in Ohmage applications. We can imagine a player conducting the surveys several times throughout the season, the player will over time memorize the questions and placement

of the options. Removing the next button provide less unnecessary interactions, and faster reporting. However, the next button will automatically be enabled on timestamp and number prompts. Number prompt has also been implemented differently from Ohmage, when the number prompt is presented, the number can be incremented and decremented with intervals. For example, say the minimum number is 5, and maximum is 300. When we click on plus-sign, the number is incremented by 5, to get from 90 to 5, only requires 17 clicks(85/5). In ohmage, this process requires 85 clicks. The timestamp prompt in pmSys is nothing different from Ohmage's timestamp.



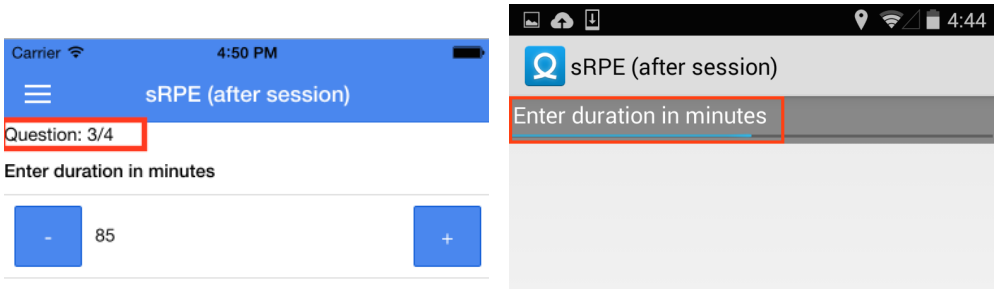
(a) Single-choice prompt (b) Number prompt (c) Timestamp prompt

Figure 4.14: Supported prompts in pmSys mobile application

4.3.6 Extra presentations

For each question presented in the application, we have created a number indicator to show how many questions we have left of the survey, and how many we have answered. This indicator is presented differently in Ohmage standalone, and did not exist in MWF version on iOS. PmSys's presentation gives the player a better indication on how many questions they have left, the result is presented in Figure 4.15.

After we have finished a survey, both Ohmage applications shows a summary text, which is defined in the campaign file on upload. Instead of showing a summary text like Ohmage do, we wanted to give the user more useful summary page, by showing a summery of their response, before uploading they upload the response, the user change the answers if necessary. We have already presented a screenshot of it in Figure 4.12 with *Report for yesterday feature*.



(a) Question's left (pmSys) (b) Question's left (Ohmage)

Figure 4.15: Presenting how many question's left

4.3.7 Filesystem

On every mobile phone and operating systems, we have limited memory available. Most mobile operating systems kills background applications as they want, to make memory available for running applications. Apple devices with iOS kills background application to free memory for running applications [42]. In order to keep the application return to the exact same state as we left it, we need to store necessary data to device storage, in order to start the application from the same state as we left.

In pmSys mobile application, we have created all necessary files and folder right after the user has logged in to the application. We are saving login information, and which server the user has logged onto. In additional, we are saving information about campaigns to present the surveys without internet connection, and after a user have conducted a survey offline without internet connection, the application stores the response into a file. This approach makes the application only require internet connection the first time we log in, and right after login, we never have to log in again, because all necessary information is stored on the file system.

Our implemented file system is also *user* based, which means, every time we have logged to the application with a valid username and password. The application creates a folder that corresponds to the username. From this point, every files created, belongs to the logged in user, and is stored within the user's folder. If the user decides to log out, all stored data that belongs to the user, will be deleted. This allows the application to have multiple user application in the future, but have not been implemented in this thesis.

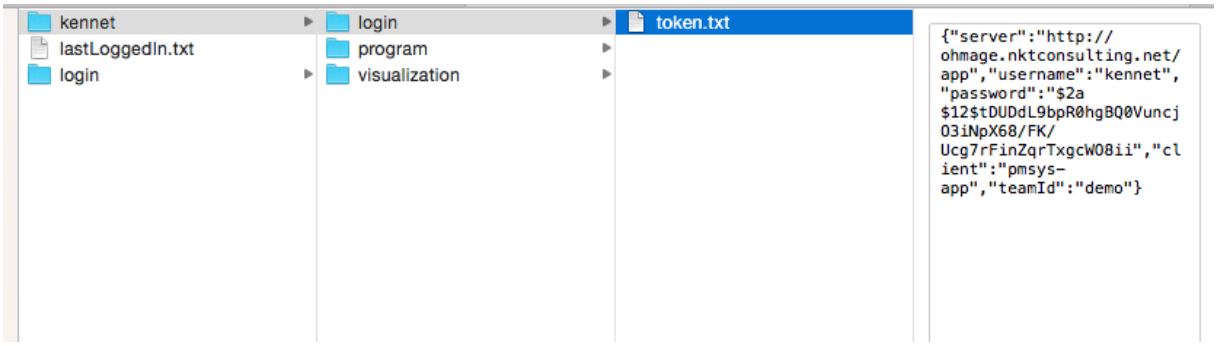


Figure 4.16: Example of pmSys file system

4.3.8 Offline reporting

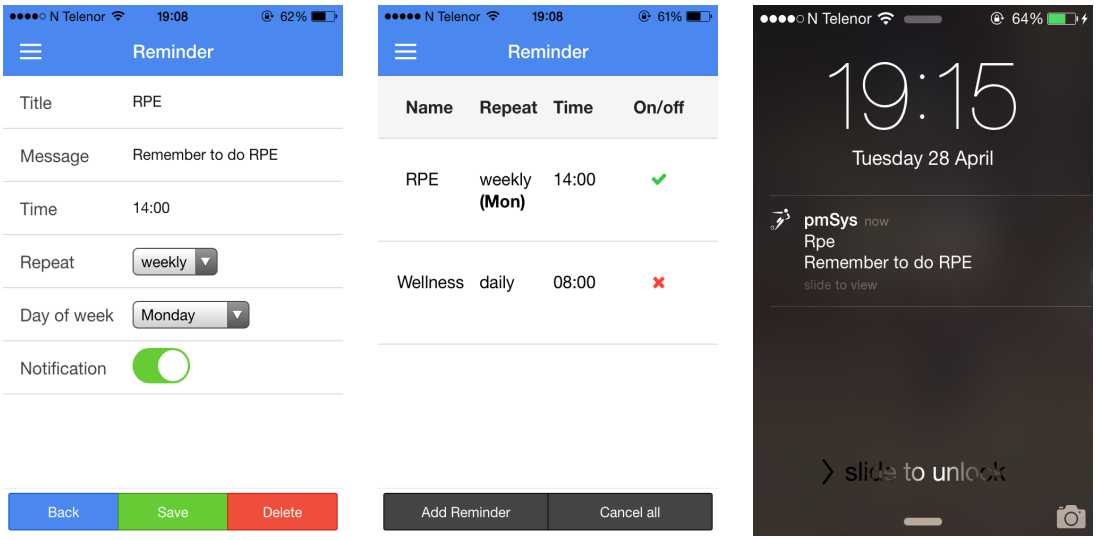
Reporting offline in Ohmage was possible, and it is possible in pmSys to, but we have done it slightly different. Since we have downloaded all campaigns the first time the user has logged in, all campaign and survey information is available in the application without internet. The reporting process and presentation of prompts is the same whether the internet is present or not. However, the only thing we have done differently from Ohmage, is to upload reported surveys that are stored offline, automatically when the application have or come in range for internet connection. On Ohmage, the user must upload the responses manually.

4.3.9 Reminder

Reminder feature was an great feature to help players report in Ohmage. In pmSys we have only support for time based reminders, and not location based reminders like Ohmage. In Ohmage applications, we could assign a survey to a reminder, but in pmSys, we have decided to left this out, and thus we provide more simple reminders. The user can set a time for when they want a reminder to be triggered, and the message to be shown, but also whether the reminder should repeat it self. For this have we used a Cordova plugin local-notification[11], which allows us to schedule a notification nativly on the platform.

[11] https://github.com/katzer/cordova-plugin-local-notifications/

A technical challenge we had with this plugin, was the API functions this plugin was providing, was not consistent. The API did not provide any option to get a list of scheduled notifications, and the information of triggered notifications where not providing correct information. So, we have to create our own containers for the reminders. We have to store reminder information at three different places and keep these three containers synchronized. We first save the reminders in-memory, to present content to the user. And then we have to save the reminder information to a file, incase the application get closed by the operating system. At last, the reminders need to be added into the native platform specific notification service to schedule a reminder. When the notifications are triggered, it will be shown on the lock screen, or in a drop down message from the top of the screen, it will be shown even when the application is not running. An example of pmSys reminder is presented in Figure 4.17



(a) Add reminder (b) List of reminders (c) Triggered reminder

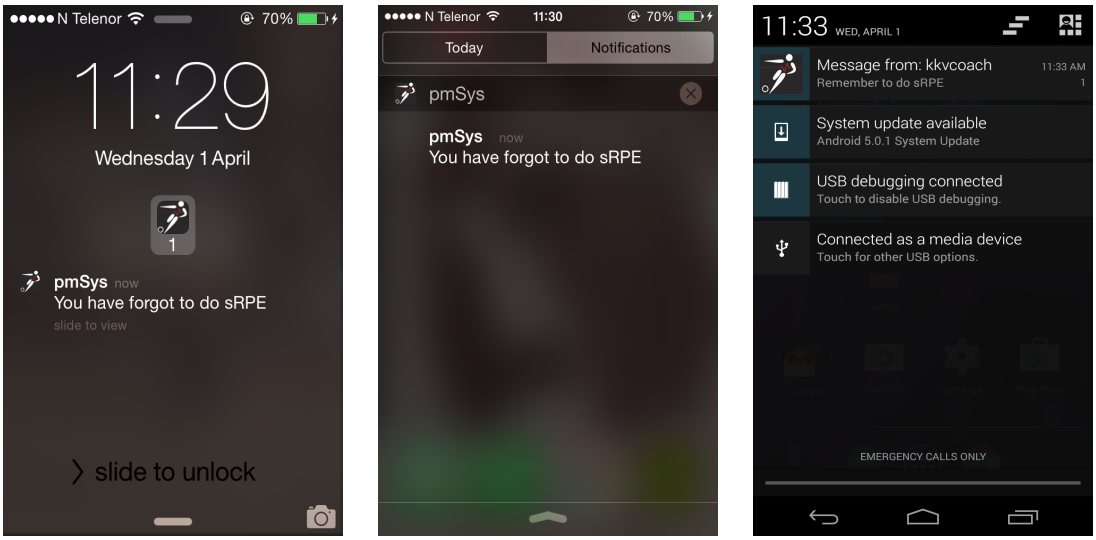
Figure 4.17: Reminder in pmSys

4.3.10 Push messages

Some players used the reminder feature, and other got reminded through co-players or coaches in person. Most of the players said that they wanted to do surveys, but they tend to forget it, they always needed someone to remind them. From the research study mentioned in Section 2.2, players got reminded by the coordinator who sent an email to remind. Another solution could have been sending a SMS to the player. Both these solutions require the message to be manually deleted after receive, in order to not fill the message box over a period. A better approach is if we could send a message that is presented to the user, and then removed itself after the user have read it.

Android and iOS devices can receive remote push message notifications through the platform's notification service. iOS devices uses Apple's push notification service [43], and Android uses Google Cloud Message [44]. Receiving push messages on Ohmage was not possible. In order for the application to receive a notification, the mobile device provide the mobile phone's UUID to a notification service. We have implemented our own push message service, which we will present in the next chapter, when a user logs on pmSys mobile application, the application automatically sends the phone UUID to the push message service, and then, the application and the user can receive push messages. We will present this more detailed in the next chapter.

As for now, it is only the coaches who can send a remote push message to a player. This allow the coach to give players instant feedback or just to remind players to conduct surveys. This communication between the coach and the player is only unidirectional from the coach to the player, but as for future work, we want to extend this to a bidirectional allowing the player to answer to the coach. Screenshots of push messages sent from a coach, is presented in Figure 4.18



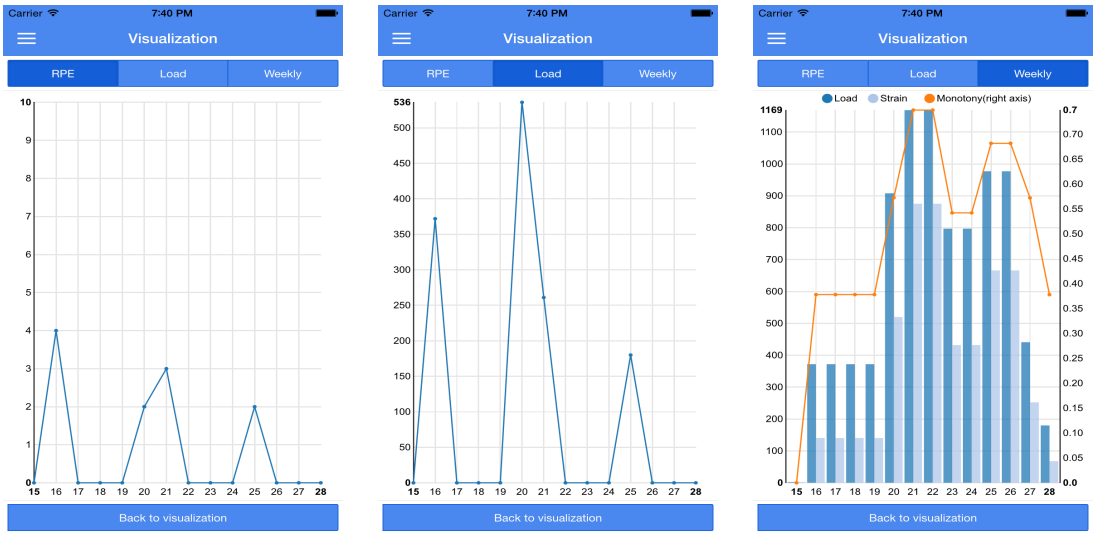
(a) iOS (b) iOS (c) Android

Figure 4.18: Push messages sent from a coach

4.3.11 Visualization

Ohmage standalone had a great feature where the user was able to view their response history, on data they have submitted to the server. We wanted to provide our users some feedback, but better and more useful feedback. This will also motivate the players do self-reporting more often. In pmSys mobile application, we visualize reported data from the players, and present it on various charts. This allows the players to monitor their weekly load, wellness and injury history. All visualization charts are based on theory from the articles we presented and discussed in Section 2.2. Our visualization feature is the best and most relevant feature for the athletes, since they can monitor their own progression.

In Figure 4.19, we can see all three RPE charts presented in pmSys. The RPE visualization only shows two weeks back in time. tThe first one is plain RPE score presented in a graphical chart. The second chart shows the load score, which is the reported session duration multiplied by reported RPE score. The last one is weekly load chart. We present three different charts plotted onto one graph, load, monotony[12] and strain[13].

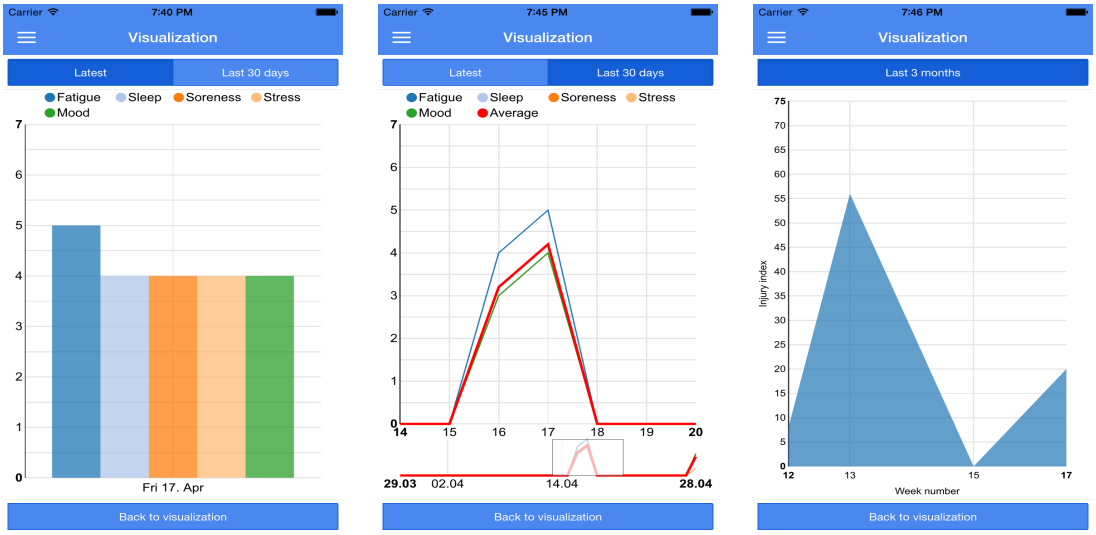


(a) Weekly rpe score (b) Weekly load (c) Weekly summary

Figure 4.19: RPE visualization in pmSys

[12]Montony = Weekly load * Standard Deviation of weekly load
[13]Strain = load * monotony

For wellness visualization, we have created two different charts, see Figure 4.20. The first one is a Histogram chart, showing the latest reported score for our wellness data. On the second chart, is a Line chart, providing wellness scores thirty days back in time, in additional, we have calculated the average score for the wellness data. The last visualization feature we have implemented, is a player's injury severity score calculated and presented with three months back in time(See Figure 4.20(c)).



(a) Latest Wellness report

(b) Wellnes data 30 days back + Average

(c) Severity score three months back

Figure 4.20: Visualization of Wellness

4.3.12 Glossary

A huge dilemma with the first version of pmSys was, whether we should have the survey presented in English or Norwegian. Ohmage did not have any support for translating questions and answers in the application. In the first version of pmSys, we ran two campaigns for each team, one in Norwegian and one in English. By creating two campaigns makes the analyzing and parsing part of the data even harder. We have to fetch data for both campaign in english and norwegian, and merge the data.

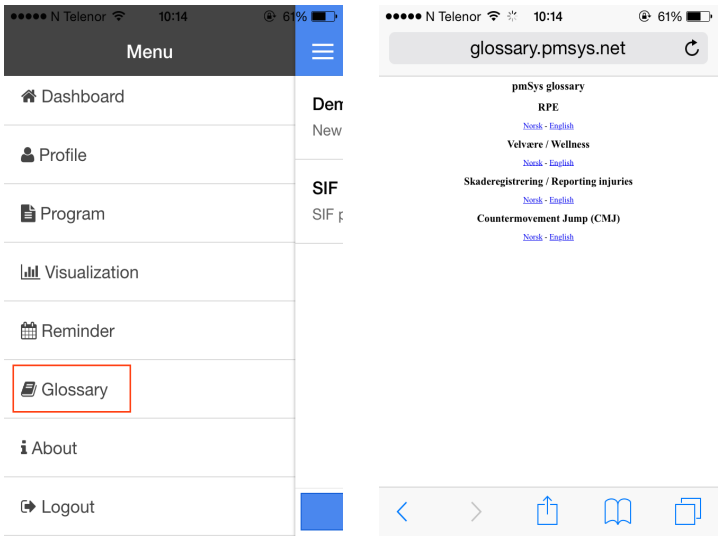
However, we have removed this approach in pmSys, and we only present one of the languages. Instead of providing two campaigns, we provide a glossary feature in the application, allowing the user to check for definition of certain words(see Figure 4.21). The glossary is hosted on a external webserver, and can easily be updated as we want.

4.4 Evaluation & discussion

4.4.1 Hybrid application

If we were not dependent on using native mobile functionalities like receiving push messages and the application to work offline. A web application hosted on a web server, would be enough and easier to develop, maintain and distribute. Most of our users have an iOS or Android device, and we limited our scope to only support these devices. Developing applications individually for each operating system, requires learning, developing and maintenance in two different languages. Synchronizing the design and features on both devices, its yet another challenge.

With a hybrid application, we can create create and develop a mobile application, and only need to deal with one language and one application that works on every platform. Hybrid application development work is great, but in the end, all hybrid applications needs to be compiled to native-code in order to be installed on various devices. This make it challenging when the development comes to iOS devices. For Android devices does the SDK work on every operating systems, whether it is windows, linux or osx. But when it comes to developing, compiling and testing iOS devices, it cannot be



(a) pmSys meny (b) Opens up in phone browser

Figure 4.21: Glossary in pmSys

done in linux or windows OS. To compile and create apple's application file, ipa, we are dependent on Xcode, which is apple's SDK to develop iOS applications. This means that even with hybrid application approach of building once, develop everywhere, we still need each operating system's SDK. An windows or Linux developer, needs apple's SDK on mac in order to compile and distribute their application on iOS platform.

When it comes to testing, each application needed to be tested separately on each platform. By following the hybrid application development approach, we saved a lot of developing time, and we only have to deal with one language, but in the end, each application need to be tested on both platforms. When we found the application usable and ready to distribute to our users, we let our beta testers(NIH) try the application before we release it, to get feedback of things we did not think of.

In order to distribute our application to our users, we needed to follow the operating system's policy and rules before we could release the application. We released many versions of our application through Google Play and Apple Store. Google play used 1-4 days to approve the application before it released the application, but Apple used up to two weeks and more to accept our application. Because of the application stores used time to accept the application, we have to set deadline for developing new features and bug fixes, to two weeks before actual release day.

The conclusion of using hybrid approach is that in the end, we still need all SDK's to build and test on the devices, and we need the SDK to build and distribute the application to the store. Hybrid application saves a lot of time in the early stages, because we have to deal with only one language, but as the application escalate and become bigger, we have to deal with the application separately by device in the end.

4.4.2 User study: Ohmage Mobile Application vs pmSys Mobile Application

We performed an user study to confirm whether our application was more user friendly and better than Ohmage. We invited 26 persons in age's from 16 to 35, to perform the study. We started with presenting the purpose of the study, without telling which application we have created, and then we told the users to put their self in an athlete's perspective before we started to test the application. We started with Ohmage mobile applications on both Android and iPhone first. Then we gave everyone an username and a password, and told them to log on, and try to perform a survey in the application, and try to navigate around the application, and use it in general. After users were done with testing of Ohmage applications, we moved to our application, pmSys. Here they did the same procedure of logging in, conducting a survey, and navigate through the application. We did not tell them which application was ours, and neither did we give them any hints, or directed their thoughts to prefer pmSys over Ohmage.

After testing both application, we provided them a survey, where we asked for their feedback on

usability, user interface/design(how the applications look), navigation(How it was to navigate within the applications) and how the content was presented. The questions presented were applied for both application. For each question they got five alternatives, (1) Poor, (2) Ok, (3) Good, (4) Very good, (5) Excellent. We ended the survey with a question on which application they prefer, ohmage, pmSys or both(the application is equal).

In Figure 4.22, we can confirm that the majority found pmSys application better than Ohmage. Most of the users found pmSys very good and excellent on the four fields we asked them. But with Ohmage, the majority found Ohmage Okey, which is less great than Good. 20 of 26 our testers said they prefer pmSys over ohmage(See Figure 4.23). We could have invited more people to get a better result, but due to our time limit, we only reached out to 26 people. We got overwhelming comments from our test users, such as:

- pmSys provided better questionnaires than Ohmage, and it was easier to navigate through the application.
- pmSys provided a summary page of the responses, which Ohmage did not have.
- pmSys was simpler to use than Ohmage, and provided more advanced content compared to Ohmage.
- Visualization was a great feature to monitor activity.
- pmSys provided more modern design than Ohmage.
- pmSys provided simple features, which resulted gave better user experience.

4.4.3 User study: pmSys as reporting tool

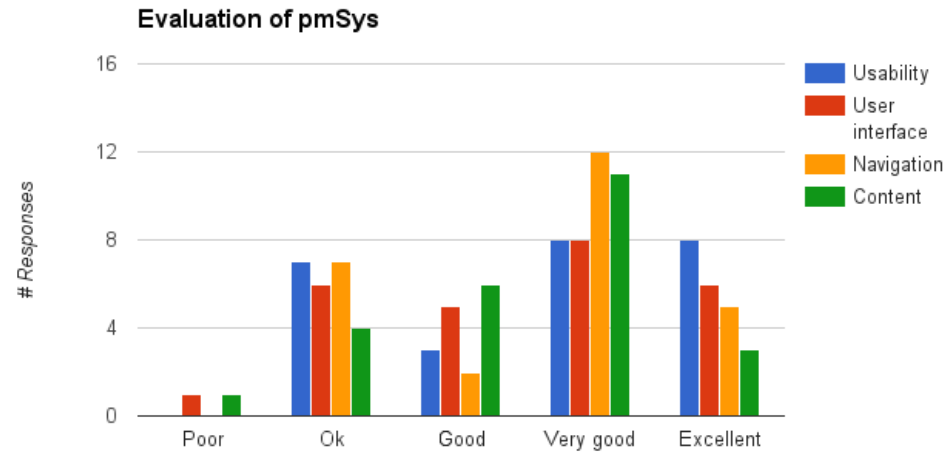
We did not reach out to the national team and Sarpsborg 08, due to ours and their time limit. But Lørenskog IF and Strømsgodset IL were positive to perform our user study to the system. This user study was performed to get a confirmation on whether pmSys was a better reporting tool than other tools such as Excel, Pen and paper or Web surveys. And how they experienced the usability of pmSys mobile application.

We got in total 30 responses, and we used the scale we discussed in previous section, from 1 to 5. We asked how they felt about RPE, Wellness and injury reporting process, and which reporting tool they like to use?. Before using pmSys, Lørenskog IF reported RPE and wellness through Excel sheets, and sent the response over email. The outcome was that everyone preferred pmSys over excel and pen and paper. We also got confirmed that pmSys saves time compared to the other tools. Thus the result of preference and time saving is not presented on the graph. The result of our study is presented in Figure 4.24, here we asked them to rate the usability, design, navigation, how he content is present, usefulness of pmSys and benefits of using pmSys.

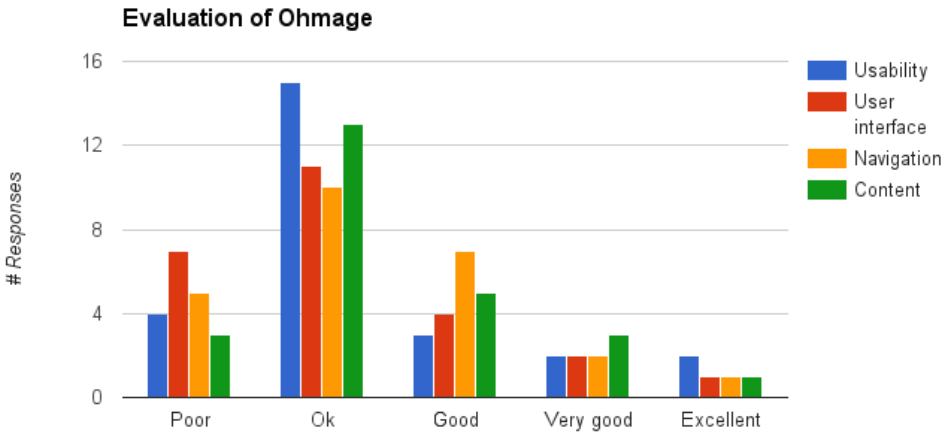
We got some great feedback from our testers, where some comment was:

- Great app! Saves me a lot of time
- pmSys is a great application to self-report with.
- pmSys works phenomenally for me, and I'm pleased with how it has affected my life(Positively).
- pmSys help me to track my training, sleep and injury.

Based on the result presented and the great comments, we conclude that our application is a great reporting tool for sports athletes. We did not get many participatory to our study due to our time limit, but we are pleased with the result, and can confirm that our application is a great reporting tool for sports athletes.



(a)



(b)

Figure 4.22: Result of user study Ohmage vs pmSys

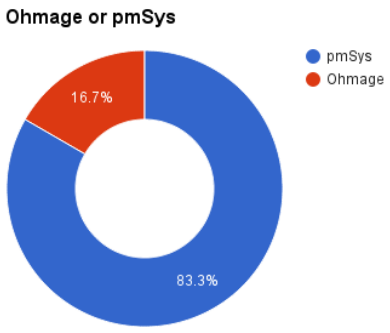


Figure 4.23: Application preference among our test users

4.4.4 Results

We wanted to perform a benchmark test, to find which application could provide the fastest reporting process in both Ohmage and pmSys. We reported reported each survey ten times on both Ohmage and pmSys, and executed the survey as fastest as we could. The reason we removed the next button was, throughout a football season, a football player will get used to the questions and answers presented in the

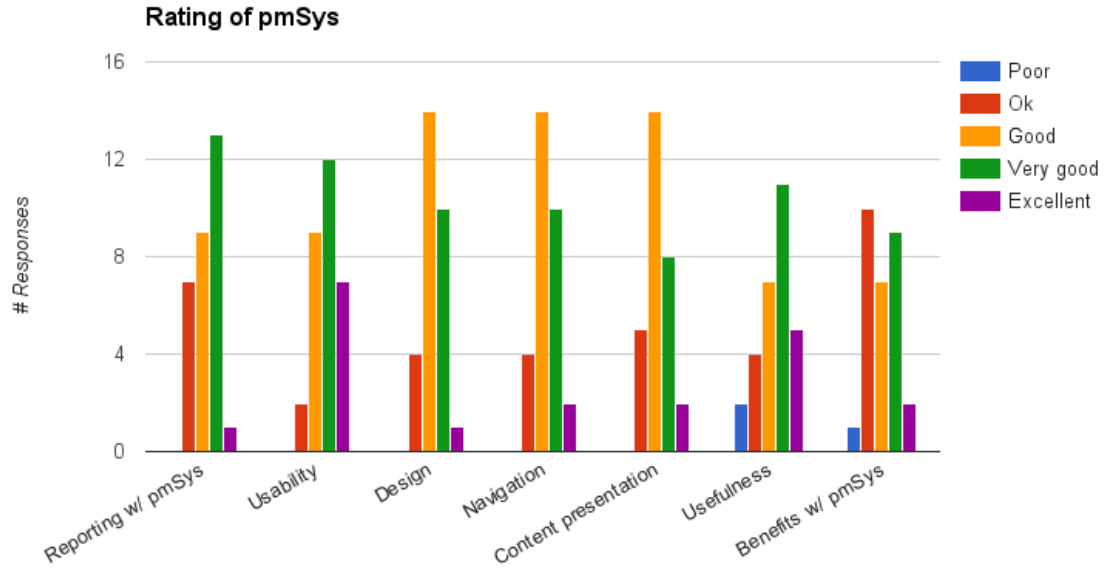


Figure 4.24: User study, pmSys ratings

application, and over time, the player will memorize and create their own pattern for reporting, knowing where each answers are located. Based on the performance test results in Table 4.1 and Figure 4.25, is our application faster when it comes to conducting a survey.

Survey	pmSys(Avg. Seconds)	Ohmage(Avg. Seconds)	pmSys(Clicks)	Ohmage(Clicks)
RPE	7,052	12,498	12	39
Wellness	8,945	11,9	13	19
No injury	4,384	6,103	6	10
Knee injury	12,313	15,543	14	23

Table 4.1: Performance test, pmSys vs Ohmage

4.4.5 Conclusion

The whole purpose of building a new application was to replace Ohmage's mobile applications, and see if we could create a improved and customized application to better fit an athlete. We have optimized the application with new features and provide more useful feedback than Ohmage. We have removed unnecessary user interactions and features, and created a faster and useful reporting tool for the sports context and athletes. We have developed an application that is distributed on two different platforms, but provide the same functionalities on both platforms. Being able to receive push messages give the user possibility to have a unidirectional communication with the coach.

With the new application have increased the number of responses tremendously. With the four clubs from the Norwegian top division, which used Ohmage application in the first version of pmSys, was total collected responses only 467, and this was numbers for the whole 2014 season from four different teams. With the new application pmSys, the total responses from two top division teams are over 5000 responses in the period March to May, 2015. And this is the numbers from two different teams, with 15-20 players. These numbers proves that our application, is a better application customized for an athlete, and motives users to report frequently with the new application. Total responses of old version of pmSys with Ohmage versus the new version of pmSys with new mobile application, is presented in Table 4.2

Based on the numbers from Apple store and Google Play store, is our application downloaded over 166 times on iOS, and 28 times on Android. Screenshot of total Apple and Android downloads in the period September to April, are presented in Figure 4.26

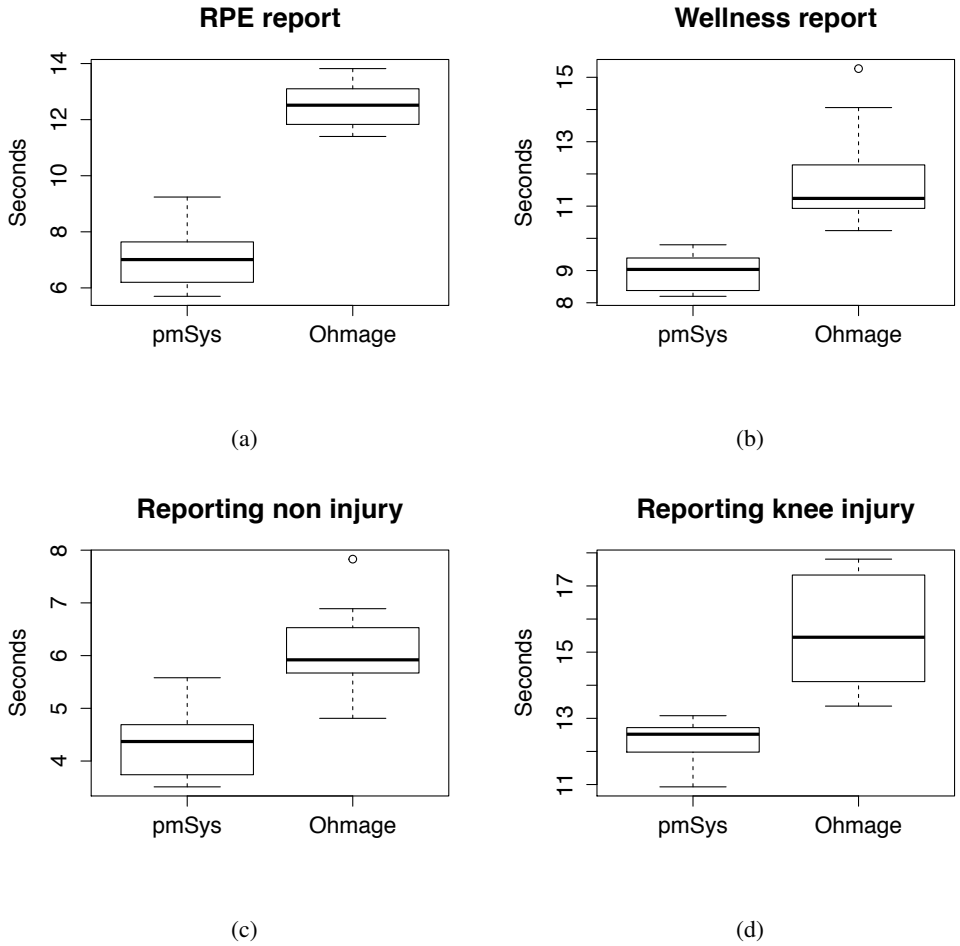


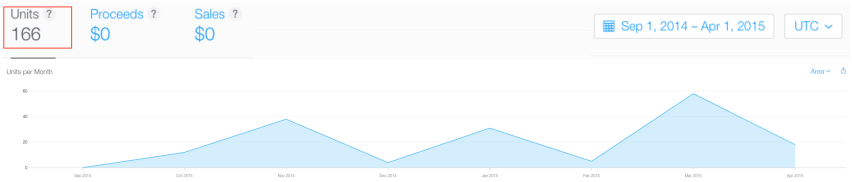
Figure 4.25: Performance plot in seconds, pmSys vs Ohmage

Version	Teams	Total responses
pmSys 2014 (ohmage application)	4	467
pmSys 2015 (new application)	2	5005

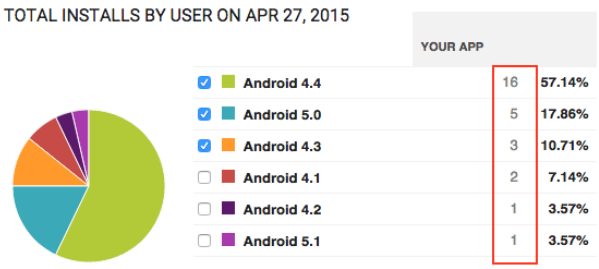
Table 4.2: Total responses, old pmSys vs new pmSys

4.5 Summary

Based on user feedback from user study and the result of our application, we have created an application that suits a sport athlete better than Ohmage. The application is easy to use and is can easy be downloaded from the application store. The application is only a reporting tool, but it contains extra tools to motivate the user to use the application. The application can receive push messages from the coach. And the application provides better feedback than Ohmage, with visualized charts. We present the new version of pmSys with our new mobile application (see Figure 4.27). In the next chapter, we present the web portal created for the coaches to use with pmSys, and evaluate this web portal against the ohmage portal presented in Chapter 3. In addition, we present our implemented push message service.



(a) Apple store



(b) Google play store

Figure 4.26: Total downloads of pmSys

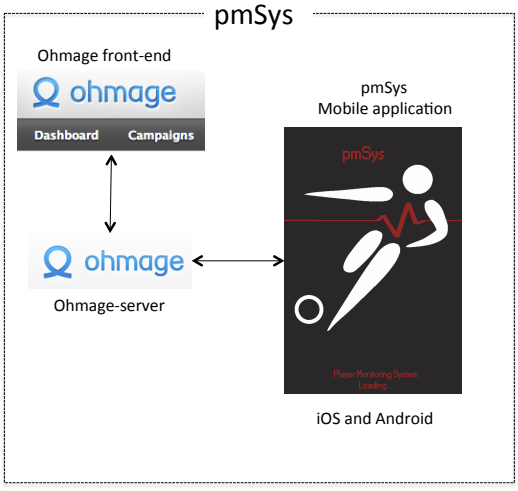


Figure 4.27: pmSys with new application

Chapter 5

The pmSys web portal

From the previous chapter, we got a detailed view on how we implemented a new pmSys application for data collecting. In this chapter, we are moving the focus from players to coaches. We are going to present a web portal we implemented, to replace existing portal from Ohmage. This portal can be used to analyze and monitor an athlete's wellness, load and overuse injury, based on the collected data from the players. We are going to present the technologies and frameworks we used to implement a such portal. And then talk about the push message service we created for the coaches, to send real-time messages to the players. We end the chapter with a summarize.

5.1 Motivation

A mobile phone have limited screen to display content and power dependent. Thus, there is limited how much content we can present, and what is optimal to present on a mobile phone. Already, players can monitor their own progression on load, wellness and injury in application. Have the coach monitor and analyze data through a mobile phone is not optimal. From Section 3.4 we presented an Ohmage web-portal, and concluded that the portal was not created for the coach to use as an analyzing and a monitoring tool.

In the first version of pmSys, coaches exported the data from Ohmage portal, and then, structured and edited the data manually in Excel. From here, they made calculations and presented the result in graphical charts with Excel. We want to replace and ease this method of analyzing and processing data, by implementing a web portal that presents the data captured from players, parsed and calculated, and then, plotted onto graphical charts. And have the coach focus on analyze and provide feedback to players.

From here, the coaches or the medical staff can detect suspicious reports from their players, and then, the staff can contact the player and provide further guidance. We want to provide the coach, a way to notify a player with a custom message, while the coach analyze the players. An example would be if the coach detects that the reported load to a player was to high, and then, the coach can contact the present player and make an appointment, or just tell him to relax for few days.

By implementing this web portal, we hope to ease the process of monitoring and analyzing for coaches and coordinators. We want to replace Ohmage-portal as a analyzing portal, but keep Ohmage-portal as a user management portal.

5.2 Design

This new web portal for analyzing and monitoring must work with Ohmage-system, in order to fetch various data from the players. A web application that have directly access to the database, fetching and process data through SQL queries, could have been a solution. But we do not want directly access to the database, because Ohmage backend already have input validation and processing, before the server stores the data, the server also handles data access restrictions and user authentications.

We wanted to follow Ohmage's system model from Section 3.2, where applications are built around, and communicating with the Ohmage server backend. This way, we also keep the database secured and

not manipulated or destroyed by unauthorized users. Neither do we need to handle database transactions, and follow the ACID[1] properties, because Ohmage-server already handle this for us.

However, with a portal built around Ohmage backend, we get a portable and independent web portal, that is not server dependent. We can install and run the web portal anywhere we want, and allow the user to pick which ohmage-server they want to log onto. If we look back at the Ohmage-portal, not only was the API endpoints hard-coded to a specific path and url, but the portal must also be hosted at the same server as Ohmage backend.

5.2.1 HTTP Client

Ohmage's web portal runs in a java servlet container, and fetches and sending data to Ohmage server through HTTP requests. We want to implement a such web application with NodeJS, instead of a Java. *"Node JS is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications. It uses an event-driven, non-block I/O model that makes it lightweight and efficient, perfect for data-intensive real-time application that run accross distributed devices" [45].* NodeJS is very easy to get started with, and as we can see in Figure 5.1, with minimal lines of JavaScript code, a HTTP server is created, this example is taken from NodeJS website [46]. NodeJS is very portable and can be installed on every desktop platforms, Windows, Linux or OSX. NPM [47] is NodeJS's package management to handle library and package dependencies.

```
var http = require('http');

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello World\n');
}).listen(1337, "127.0.0.1");

console.log('Server running at http://127.0.0.1:1337/');
```

Figure 5.1: HelloWorld code in NodeJS, listening for HTTP connections

The idea is to have NodeJS as a server instance running and fetching data from Ohmage-server over HTTP protocol, and serve like a proxy instance between the client and the server. In NodeJS, we can also have HTML server side rendering to present content. This approach is different from normal single-page-applications, where the HTML rendering and HTTP fetching is managed by the client. A system model of our envisioned web portal is presented in Figure 5.2.

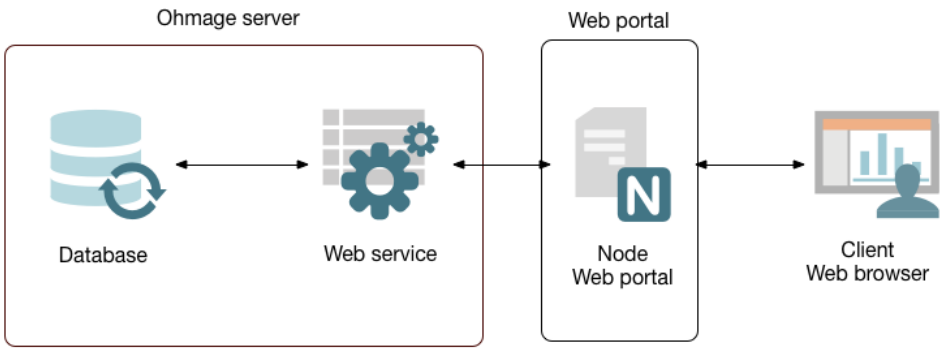


Figure 5.2: pmSys web portal - System overview

5.2.2 Web application

According to NPM's homepage it exists about 138487 packages(libraries and frameworks [47] to NodeJS. Like every other programming languages projects, every project have a project file to resolve

[1] ACID - Atomic, Consistent, Isolated, Durable

dependencies. NodeJS also have a project file structured in a JSON format. To have a project file, where we define all dependencies our project need in NodeJS, is one of the benefits of using NodeJS.

ExpressJS[2] is a web framework to NodeJS, and serves as a layer on top of NodeJS. ExpressJS ease the web application development and let the developer write less and cleaner code. With ExpressJS, we do not need to handle HTTP requests in detailed, such as manage HTTP headers on how the packets should be structured. Hence, we can focus on the content and how the content should be presented. In Figure 5.3 we see an example of ExpressJS server that does exactly the same as plain NodeJS server from Figure 5.1.

```
var express = require('express');
var app = express();

app.get('/', function(req, res){
  res.send('hello world');
});

app.listen(3000);
```

Figure 5.3: HelloWorld in ExpressJS [10]

Another advantage we found with ExpressJS was that it provides routing support. Without ExpressJS, we must handle all incoming requests manually, and check the request object for what content the client requested, and respond to the request. This is handled automatically in ExpressJS. Yet another powerful feature in ExpressJS, is that we can pick any HTML rendering library we want. We decided to use Nunjucks[3], which is a template engine for JavaScript to render HTML content. Nunjucks have support for inheritance of views, where a child view can inherit from a parent view. Nunjucks also provides conditional rendering on the HTML pages, we could for example send variables to the view, and the HTML page will be rendered based on the value we sent.

In order fetch data from Ohmage server, we need a HTTP client in NodeJS. Request is a simplified HTTP client in NodeJS[4] which we used as to fetch data from Ohmage server. We wanted our users to use our portal, without need of authenticate all the time. In order to fulfill this, we need a session storage to cache all data from the users throughout a session. Instead of saving everything in memory, we have stored every user data for a session, in a redis-database[5]. Redis is a open-source, lightweight, key-value database.

5.2.3 Push message service

We wanted to provide a way for the coach to communicate with their players, by sending a message to announce something, or just to remind the players to conduct their survey. A solution we mentioned in Section 2.2, was to send an email. We wanted to take advantage of the smart phone, by sending remote push messages directly to the players phone.

It exists a lot of providers that offer various push message services, some of the services are: Amazon SNS[6], Parse[7] and Appoxee[8]. A disadvantage we found with these services, we have to pay the services use them. We also wanted the coaches to send messages from our portal and use the already existing user system in Ohmage. With these services, the portal is also dependent on the third part service. Some of services offered API access, so we could integrate their service into the portal. But again, these services come with a cost. Thus, we wanted to control and create our own service.

Our first goal with our push message service, was to allow a coach to send push messages to their players, which means a unidirectional one to many communication. Since we have build wanted to build

[2]ExpressJS - `http://expressjs.com`
[3]Nunjucks - `https://mozilla.github.io/nunjucks/`
[4]Request - `https://github.com/request/request`
[5]redis - `http://redis.io/`
[6]Amazon SNS - `http://aws.amazon.com/sns/`
[7]Parse - `https://www.parse.com/`
[8]Appoxee - `http://www.appoxee.com/`

the portal with NodeJS, we decided to implement the push message service in NodeJS as well. A web service that works besides the web portal, to handle push messages. In NodeJS, it exist two libraries we could use to send remote push messages to iOS and Android devices, node-apn[9] and node-gcm[10].

In order to send push messages messages, we are also dependent on the mobile device's associated UUID[11]. Once we have the UUID, we could send push message with the libraries mentioned above. The NodeJS libraries provides an easy interface to send messages to the platform message service, and we does not have to read the APN and GCM documentation to set up a push message service. An example of the portal can be found in Figure 5.4

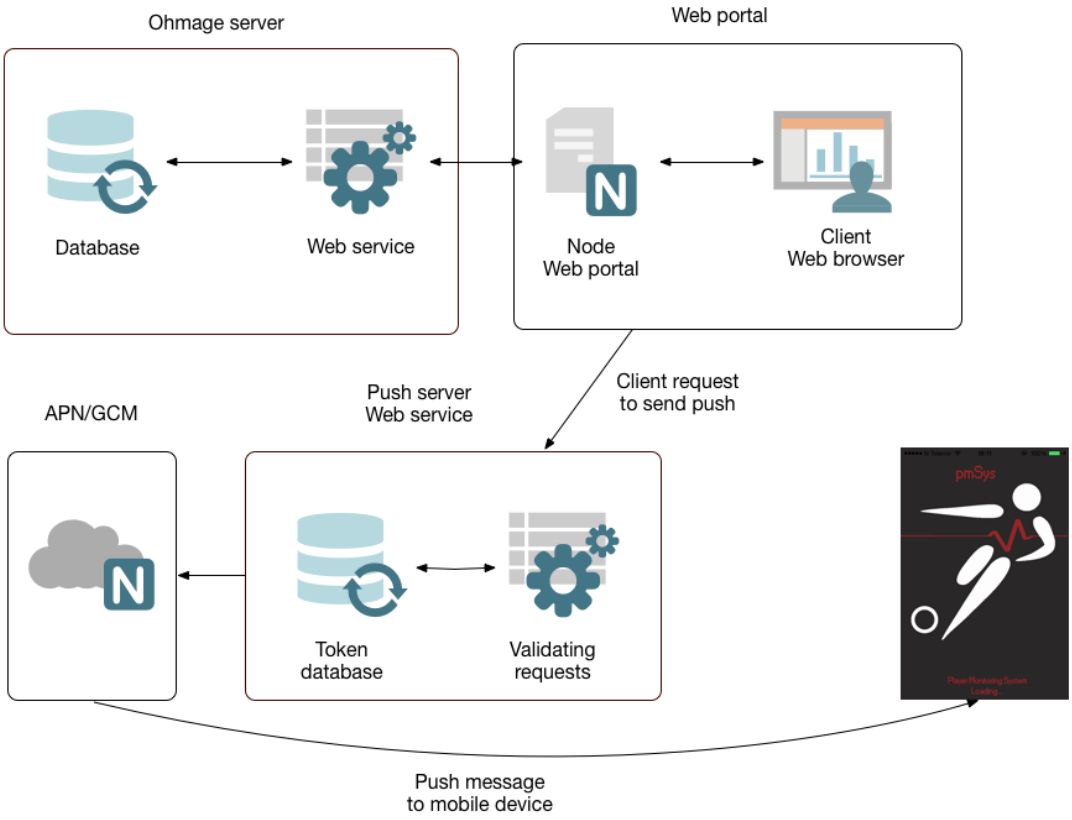


Figure 5.4: pmSys with push message service

5.3 Implementation

5.3.1 Login

Since we wanted our portal to be used with diffrent Ohmage back-end servers, we have provided the user an option to pick a server on login. This allows different teams the possibility to host their own ohmage server. In case their want own and control the reported data, and then use our portal to analyze and monitor their players. A screenshot is provided in Figure 5.5. We have limited the access to only allow coaches to use the portal.

5.3.2 Session storage

We mentioned a redis database in the previous section. The reason for why we used a redis-database, is because we can save users data throughout a session, for example after logging in to the portal, we can save the password and username onto redis-database, to be used later when user fetches data from

[9] Apple push notification module for NodeJS `https://github.com/argon/node-apn`
[10] Google cloud messages for Android module for NodeJS - `https://github.com/ToothlessGear/node-gcm`
[11] Universally unique identifier

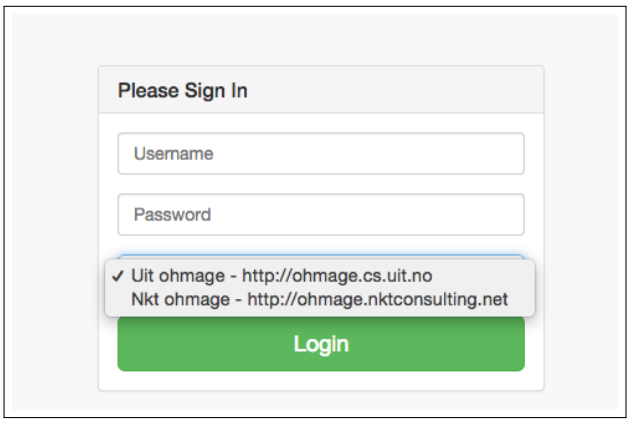


Figure 5.5: Login screenshot of pmSys portal

the Ohmage server. And then, we do not need to ask for the user password all the time. The idea of using a redis-database to store session data, is taken from Modulus [48], a NodeJS hosting platform. The redis-database is hosted on the same platform as the web portal, and is not accessible from the outside. A system overview of pmSys portal can be viewed on Figure 5.6

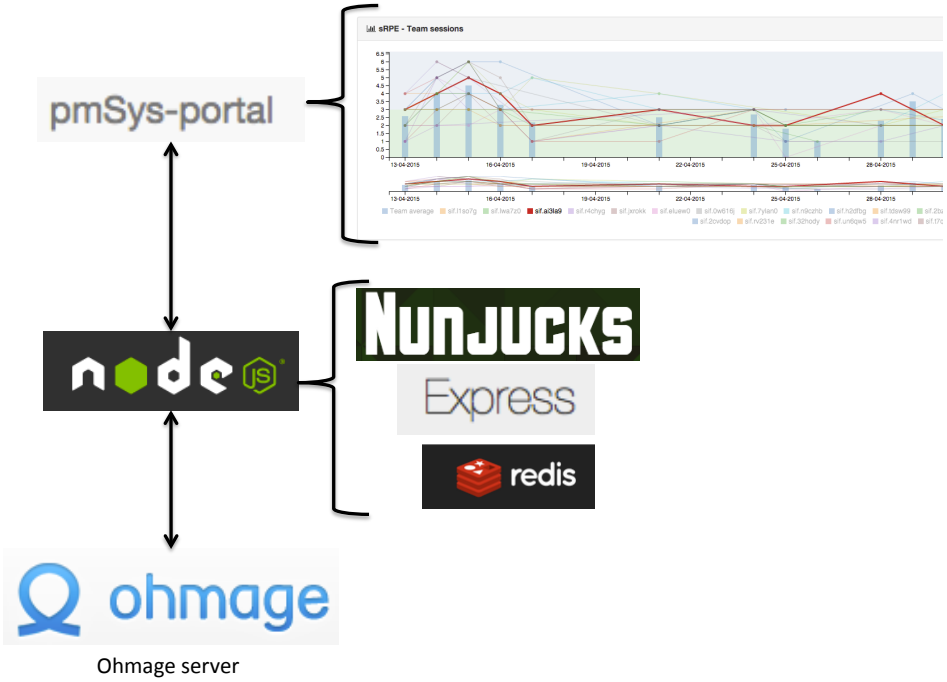
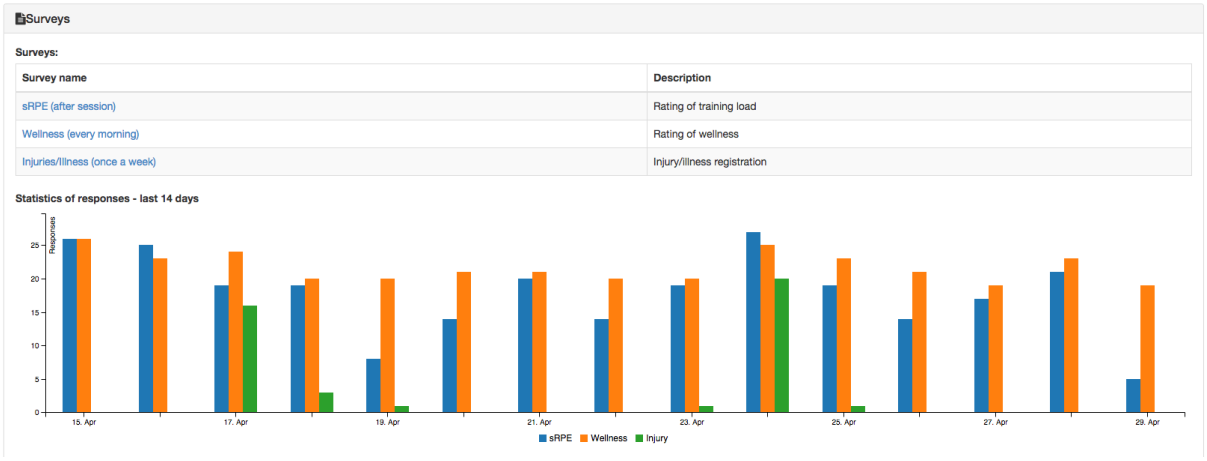


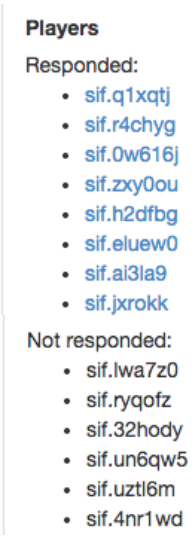
Figure 5.6: An overview of pmSys portal

5.3.3 Responses

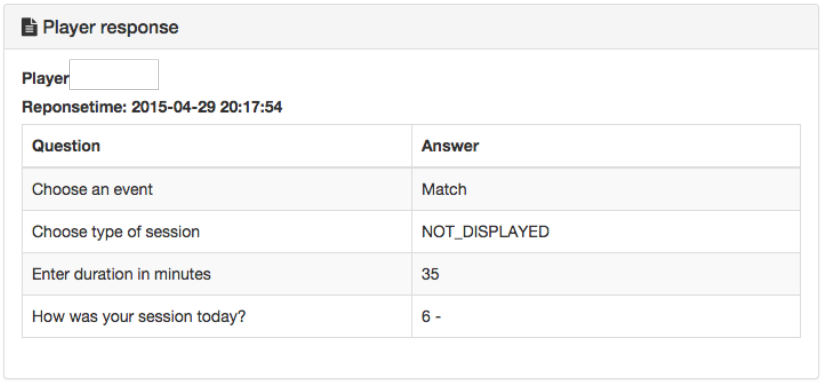
How the web portal presents the responses is differently from Ohmage web portal, instead of a filter to view the responses like Ohmage provides. We first present a list of accessible campaigns, secondly we present a list of available surveys, before pick a survey, we present a statics over reported data the last 14 days (See Figure 5.7(a)). And after the user have picked a date to view responses, the portal presents a list over who has reported that day and who have not (See Figure 5.7(b)). And then the response is presented to the user in Figure 5.7(c).



(a) Available surveys + statistics



(b) Player list



(c) Response data

Figure 5.7: Viewing a response

5.3.4 Visualization

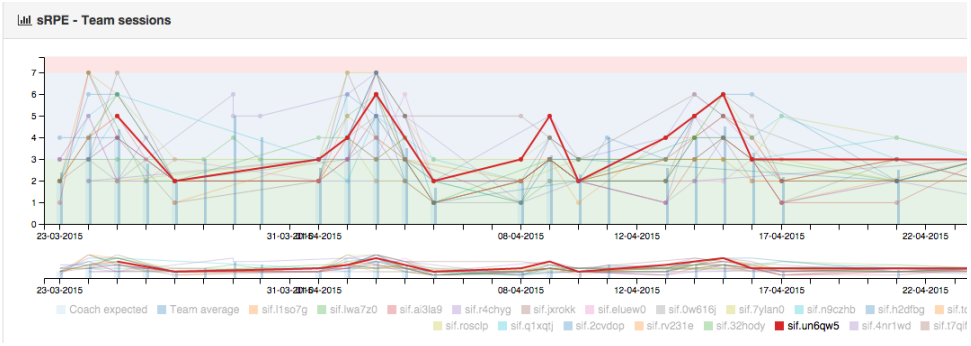
Ohmage front-end offered some basic statistics visualization. However, calculating reported data, such as load, strain or monotony was not possible in Ohmage. Coaches have to export the data and visualize and analyze in third part application. We wanted to create interactive charts, and visualize reported data, that have been processed before presenting to coaches. The pmSys web portal visualization, provides two types visualization groups, the whole team and a single player. Both the visualization groups are based the three surveys, RPE, wellness and Injury. All visualization charts are interactive(see Figure 5.8), which allows the coach to view detailed data on a specific player.

RPE visualization

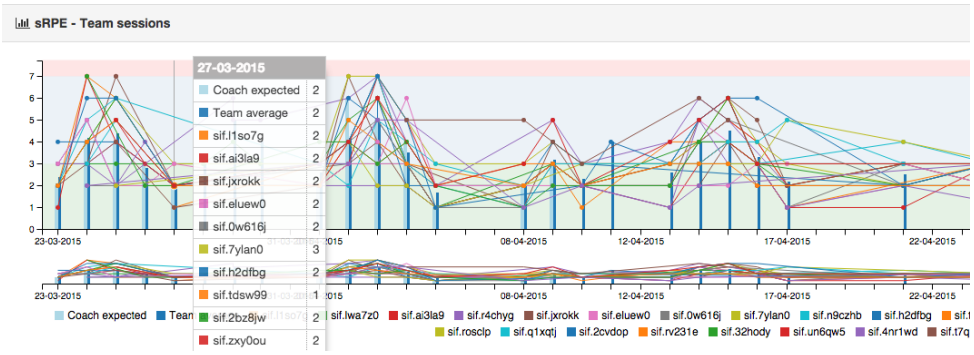
We visualize all players reported RPE score and load. We have also provided a team average score on load and RPE for the coach, the planned RPE score is also presented in the charts (See Figure 5.9).

Wellness visualization

On wellness visualization, we can choose to visualize Average, Mood, Sleep Amount, Sleep quality, Soreness, Stress, Fatigue or Readiness. These variables are visualized based on what the player have reported (See Figure 5.10).

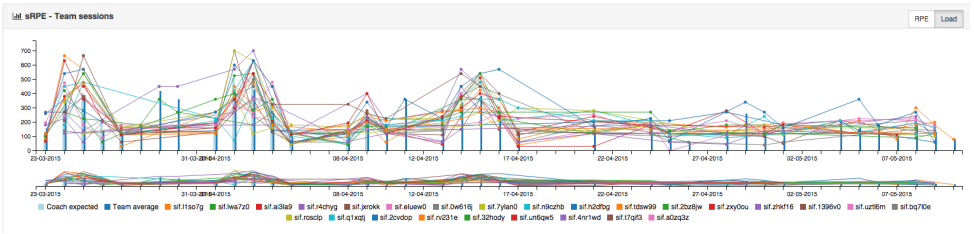


(a) Highlight single player

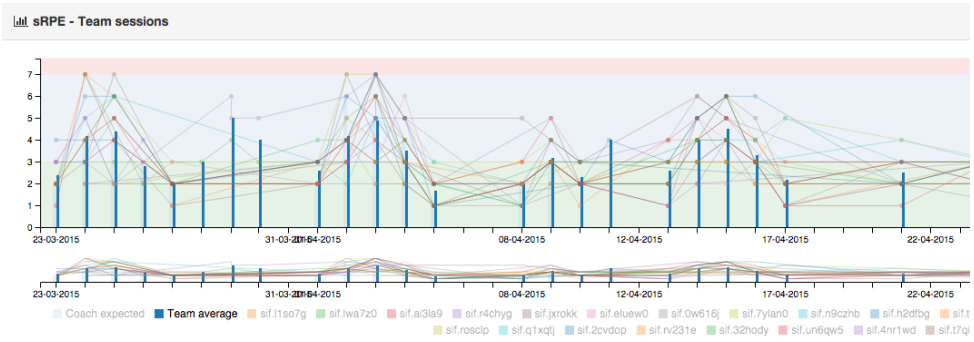


(b) Data on date

Figure 5.8: Interactive graphs



(a) Load

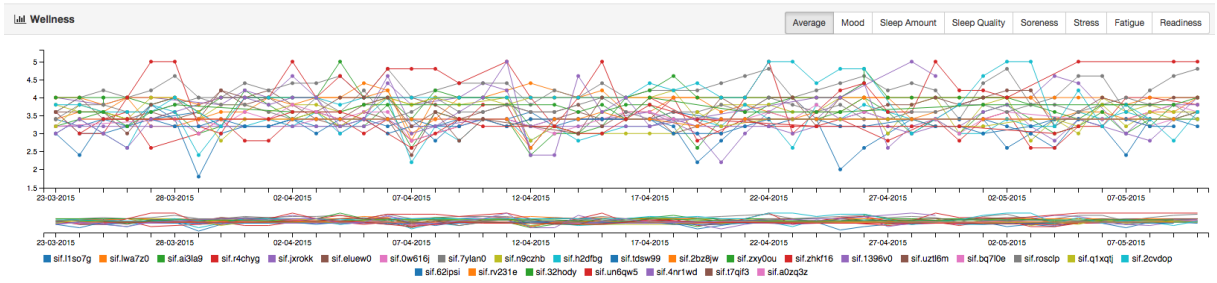


(b) Team average

Figure 5.9: Team visualization of Load

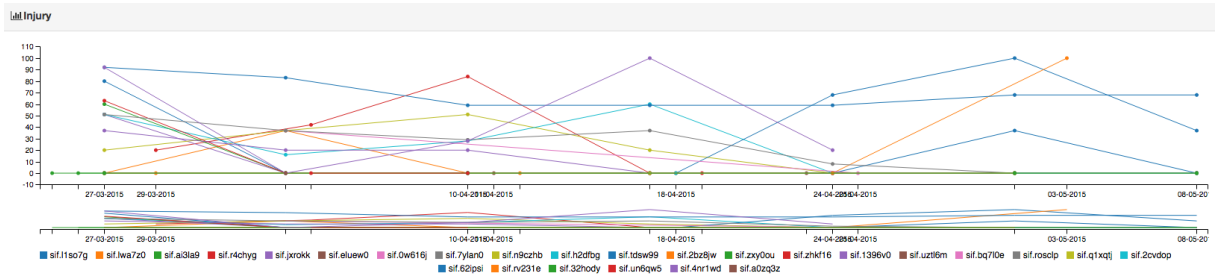
Injury visualization

On injury visualization, we have presented the severity score based on the four key questions from Section 2.2 (See Figure 5.11).



(a) Average wellness visualization

Figure 5.10: Example of wellness visualization



(a) Severity score

Figure 5.11: Team visualization of Injury

Individual visualization

The coach can also visualize and monitor each player individually. Here is all visualization features presented on one page. A coach view the players Readiness to play, load, wellness and injuries. The charts here are also interactive. See Figure 5.12

5.3.5 Push messages for coaches

It does not exist any features in Ohmage-system, that could provide the coach a view of who have not reported for a day. PmSys web portal provides a statistics over last 14 days reports, but that does not help the coach to reminder the players who have not reported any easier. Thus, in pmSys web portal, we present a overview for who have reported, and who have not. In Figure 5.13 we see a overview of who have reported RPE and Wellness for the day. From here, the coach can mark all or the players who have not reported, and then send marked players a custom message.

Have the coach send push messages manually every day to remind the players, is impractical in long run. We want to ease this process by implementing a way to schedule a message, and then be sent on certain time. This way the coaches can for example, schedule a reminder to be sent every morning, to remind the players to conduct wellness survey. With this feature, we are not only helping the players to report, but we have relieved the coach from remembering to remind the players to report every day. In Figure 5.14, we can see a screenshot of scheduled reminders, and how to add a scheduled reminder.

5.3.6 pmSys: Push message service

Our push message service consist of a MySQL database to store username and device tokens(UUID), and above the database, we have a web-service layer, which communicate over HTTP protocol. The web service is built with NodeJS, and we use the plugins we mentioned in design Section 5.2.3. These plugins ease the process of setting up connection to a Apple Push Notification(APN) service or Google Cloud messages (GCM), and provide an easy way to send push messages. The only requirements to use these plugins are, we must have signed certification from Apple and Google, and these certifications can be requested from Apple and Google with a developer account.

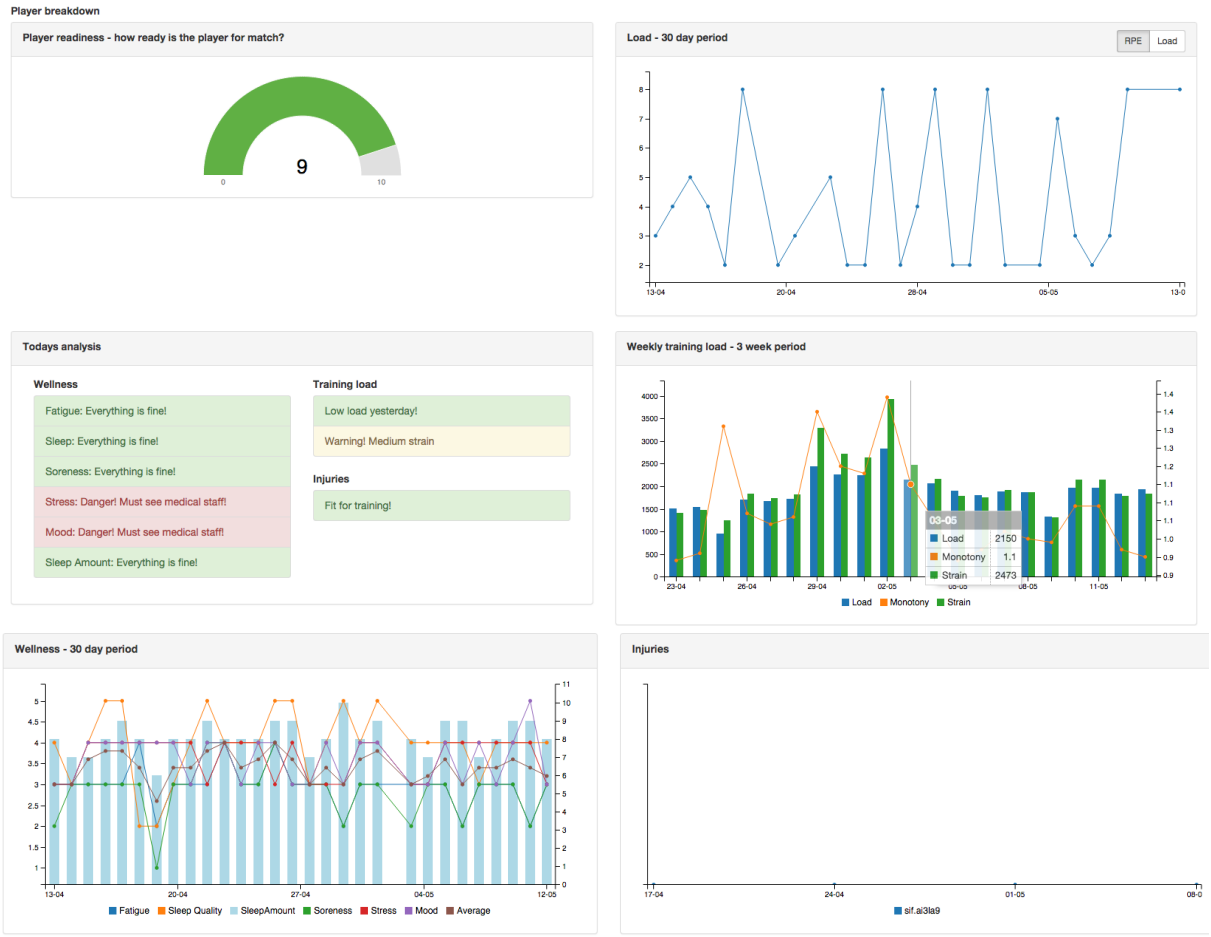


Figure 5.12: Individual visualization

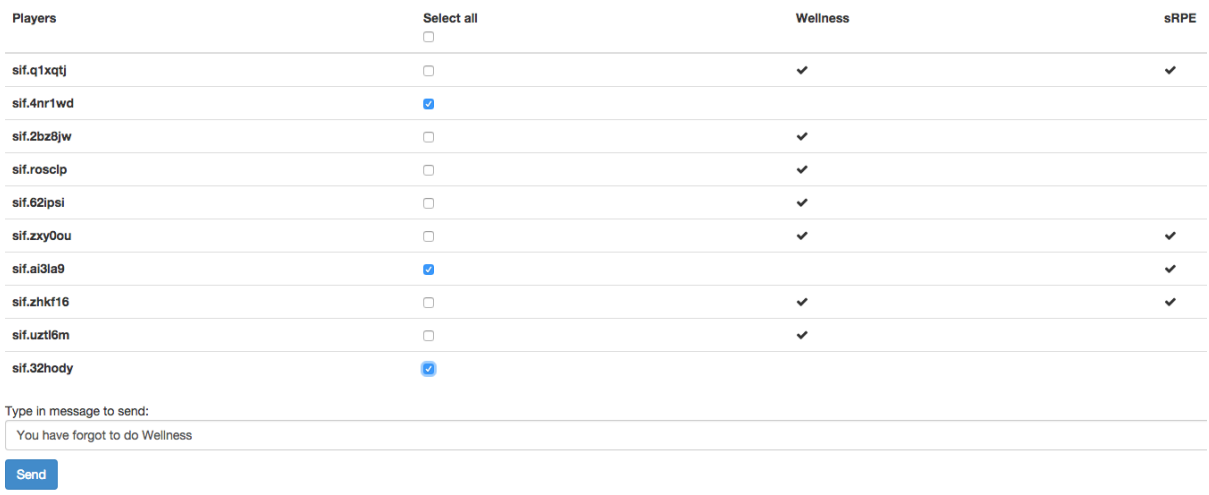


Figure 5.13: pmSys push messages

When the coach sends a push message through pmSys web-portal, the portal sends a HTTP request to our push message service, and then, our service sends a request to APN or GCM, based on which device was requested. And then, APN and GCM send a push messages to the requested devices. The devices subscribe it selves with the new application, by sending a subscribe request to the push message service after the user have logged in.

As we discussed from previous section, a coach could schedule a push message for when he or she want the message to be sent. We solved this by extend the push service to support cron jobs by creating

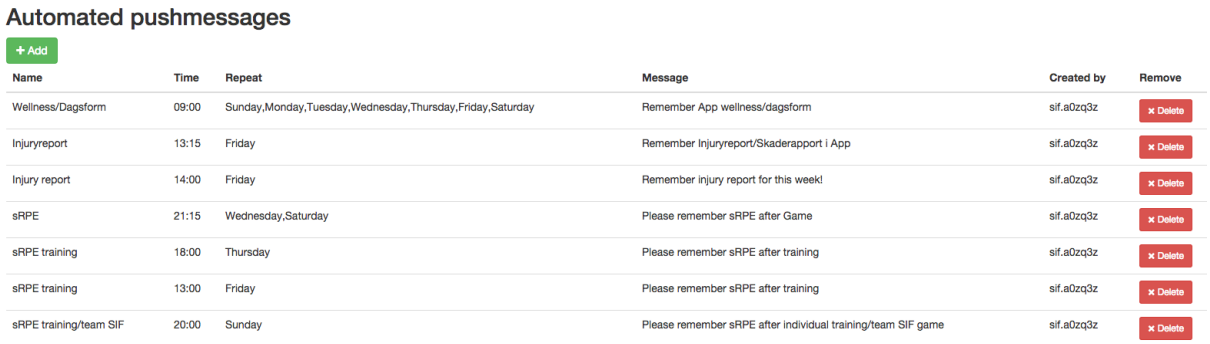


Figure 5.14: Screenshot of scheduled messages

Endpoint	HTTP Method	Description
/subscribe	POST	To subscribe a mobile device to pmSys push service, must provide username, device platform(iOS/Android) and device UUID/token
/unsubscribe	POST	To unsubscribe a mobile device from receiving push messages, must provide username.
/send	POST	To send a push message, must provide a user list of receivers.
/users/:userId	GET	To get device information for a user. Username must be provided
/users	GET	Provides a list of all registred users.
/teams/:teamID	GET	Provides a list of all registred user for a team.

Table 5.1: pmSys push message service

new HTTP endpoints. The new endpoints can be found on Table 5.2. We implemented this feature by using a NodeJS plugin with name node-cron[12], which allows us to schedule any task we want in NodeJS.

Endpoint	HTTP Method	Description
/cron/create	POST	Creates a scheduled event, using cron-job syntax, returns a cron-job id
/cron/stop	POST	Stops a scheduled event, need to send cron-job id
/cron/getAllCronJobs	POST	Returns a list of all scheduled cron jobs for a team.

Table 5.2: pmSys push message service

5.4 Evaluation & discussion

5.4.1 Visualization bottleneck

In the beginning with the web portal, our vizualization feature worked excellent when it had small number of data points to present. But when it had more data points to render it became slow. The reason for the slow rendering, was how we parsed, rendered and visualized the data. When the coach requested for visualization, the portal sent a request to Ohmage-server to fetch necessary data, and when the portal got response from Ohmage-server, the portal sends the data unparsed clients web browser, and let web browser render and present the HTML and charts. We thought that having the client render the charts, could relief the portal from server-side rendering. But we were wrong and we decided to move the rendering and parsing over to server site, fixed the problem. We have NodeJS both fetch and render the content from Ohmage and then present it to the user, turned out to remove the slowness of visualization rendering. After we moved the client rendering to server side rendering, the visualization was not a bottleneck in our portal anymore.

5.4.2 Push message service

Push message service have increased the total responses. A huge reason for this, was that pmSys application could receive push messages, and the coach could help the players to do their survey but

[12]node-cron `https://github.com/ncb000gt/node-cron`

also send push messages at any time. Another factor that increased the data reporting, through the web portal was the coaches able to schedule a push message. Right after we released this feature, with the coach able to add schedule reminders. The amount of responses reported increased. In figure 5.15, we can see the result of scheduling a push message, increased the reporting significantly right after the release of this feature.

Our push message service is not production ready, even though it being use in production now. When we implemented our push service, we did not consider security aspects of the system. Hence, we are validating every HTTP header and request for a certain value to be defined. This is to prevent unauthorized users to spam our service with push messages. Our push message service is still open to be abused, if the user know the endpoints, and use the service outside the web portal.

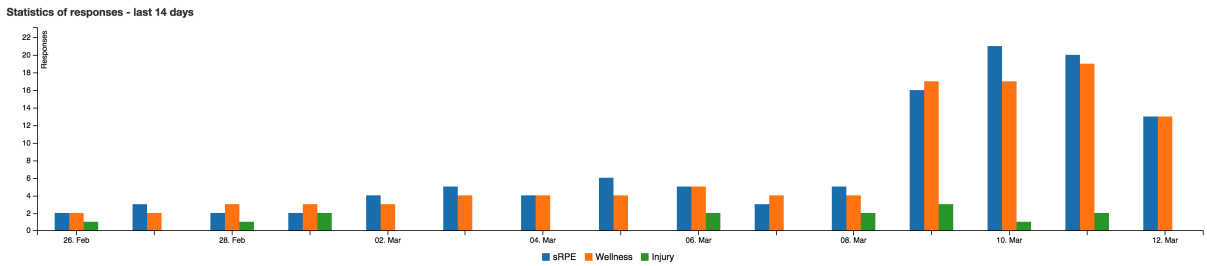
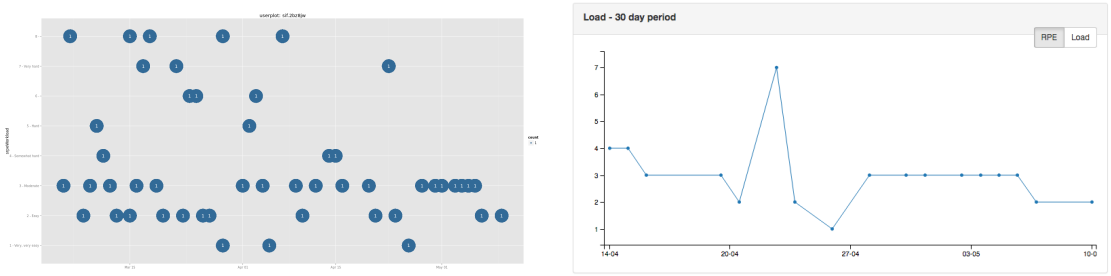


Figure 5.15: Increased reporting after scheduled push messages

5.4.3 pmSys vs Ohmage frontend

Visualization

Our visualization feature in the web portal provides more useful feedback for monitoring and analyzing reported data than Ohmage web portal. Our Web portal provides customized charts based on the three survey's the players are conducting, and the numbers are calculated and processed before presenting. In figure 5.16, we have visualized a player's RPE value in Ohmage web portal and pmSys web portal. As we can see, Ohmage only provide a statics on how many times a player have picked a value. However, we can visualize a players RPE score and load value over a week and more. We provide more specific visualization, for example based on load, wellness and injury, but Ohmage provide statistical visualization. In addition, the portal can present the statistics of reported data the last 14 days. Comparing this to Ohmage, we have more clean and easier to understand feedback than Ohmage's total responses. See Figure 5.17.



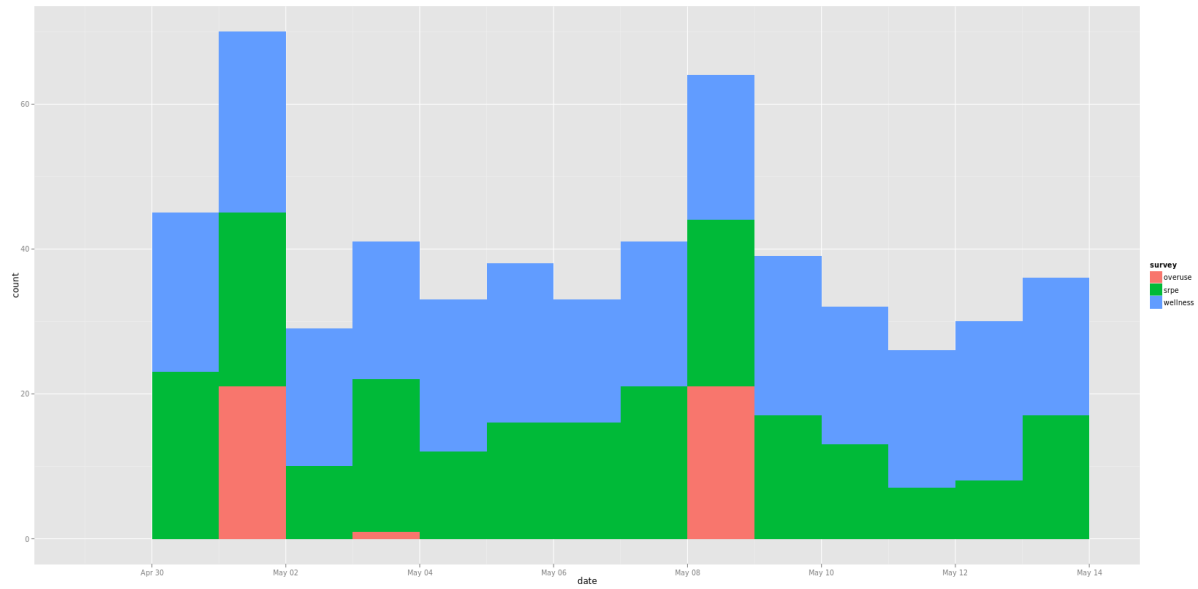
(a) RPE in Ohmage portal

(b) RPE in pmSys portal

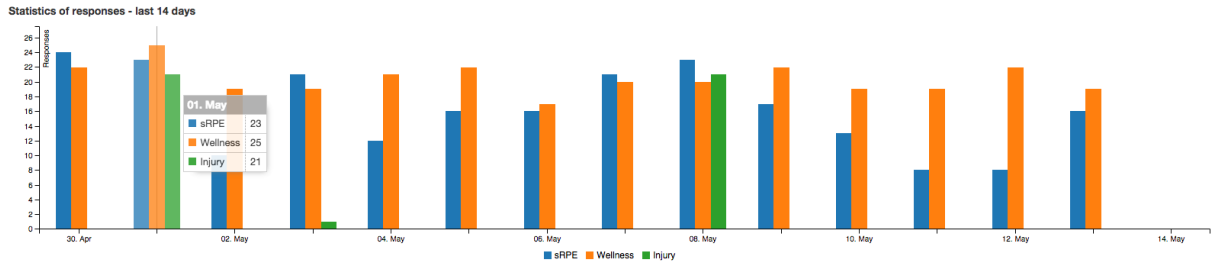
Figure 5.16: Visualizing RPE in Ohmage and pmSys

Response

pmSys web portal is more focused on presenting the reported data and provide more useful feedback than Ohmage web portal. Ohmage did not provide a way to view who have reported and who have not. In pmSys, we have presented a overview of who have reported and have not reported (see Figure 5.18),



(a) Statistics of responses in Ohmage portal



(b) Statistics of responses in pmSys portal

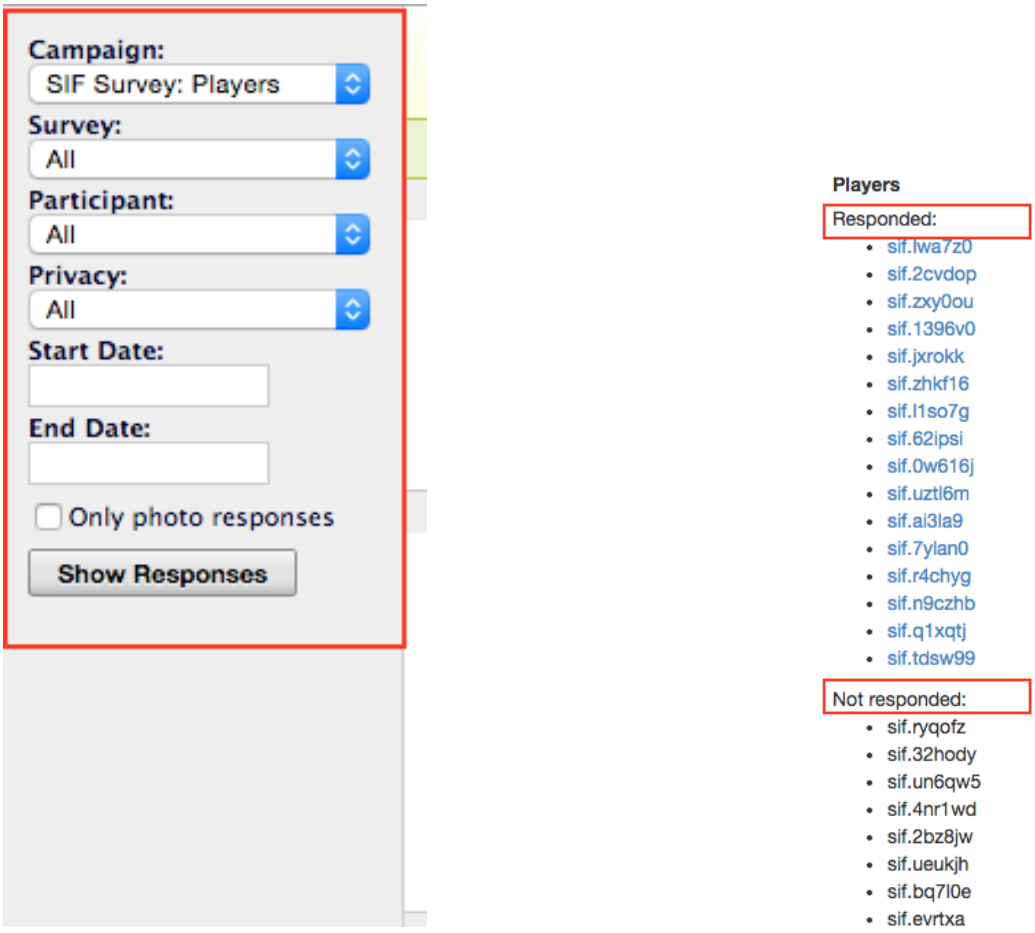
Figure 5.17: Visualizing total response last 14 days in Ohmage and pmSys

Ohmage on the other hand, let the user filter the responses, this filter is also supported in pmSys as well. This feature was highly demanded, thus, a coach can contact the players who have not reported, and tell them to do so.

5.5 Summary

In this chapter, we have presented our implementation of pmSys, web portal. We have proved that our portal, provides more useful features than Ohmage. With features such as visualized feedback, and this allows coaches to monitor and analyze their players. The visualization is also heavily focused on a player's load, wellness and injury, rather than statistics. We have increased the amount of reported data with our push message service, and this feature was highly demanded from the first version of pmSys. The portal have also made it easier for coaches to monitor and analyze their players in general. We also got great feedback from NIH, where they said pmSys was a very useful and good system to monitor the players, and the system have helped their research a lot. pmSys with the new portal and the application is illustrated in Figure 5.19

This web portal is not considered production ready, even though the portal is used by the various coaches in reality. But the portal have a great potential to be something big, and to be extended with new features. We can for example build a bidirectional communication channel between the coach and the player. In next chapter, we present the Ohmage-server, and our study hosting Ohmage-server in the cloud. We have also improved the back-end server with new endpoints to reduce latency and data transfer between the client and the server.



(a) Ohmage response filter

(b) Player list

Figure 5.18: Responses in Ohmage and pmSys

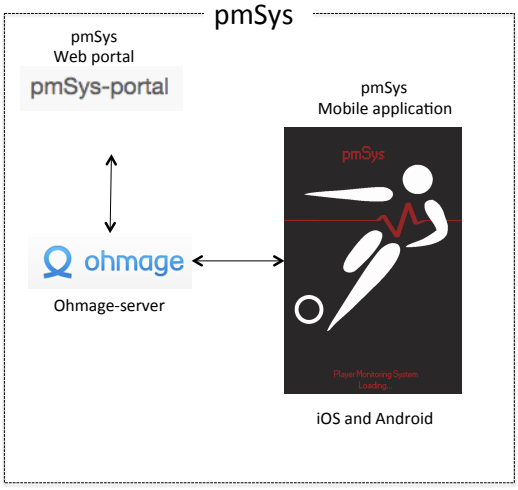


Figure 5.19: pmSys with new portal

Chapter 6

The pmSys backend server

In this chapter, we present technologies used in Ohmage-server. And then, we study and analyze whether we can find any advantages of hosting Ohmage-server in cloud with focus on Latency. We only analyze network latency to the various cloud services and compare it to our self-hosted environment in Tromsø. At last, we present possible improvements to Ohmage-server, and how these improvements can reduce the latency and data transfer between the server and client.

6.1 Motivation

Until now, we have presented our implemented pmSys mobile application and web-portal. In this chapter, we move the focus over to the server. The server is the place where all data is stored and processed One of the reasons we wanted to use Ohmage and built our system around it, was because Ohmage provided a server backend that could handle reported surveys, but also present surveys and captured data. We could just build applications to work around Ohmage. Ohmage also offers a great user management system with clear user roles, that help us separate a player role from a coach role, and keep data from unauthorized users.

When the pmSys applications were built, we saw that Ohmage-server provided unnecessary data, and the application needed to do many unnecessary server requests, to get enough data before we could visualize and present it. We started to look at the server implementation, and wanted to find optimizations and improvements towards the server.

Throughout the thesis, we hosted our Ohmage-server at University of Tromsø (UiT). They were responsible for maintaining and keep the server secured. We never experienced any problems with their hosting. Until it came to a point, where the servers at Tromsø needed an upgrade. UiT performed a hardware and UPS(uninterruptible power supply) upgrade, and the server was taken down for few hours. We had another scenario where the electricity at Tromsø went down, and the server were unavailable for the days. We did not consider that the server could go offline, and neither did we have any backup plan. Due to server offline, players where unable upload responses to the server, and the coaches could not fetch data from the server. Hence, we started to consider cloud hosting, since the cloud infrastructure upgrade is a less problem at the cloud services, and they can guarantee up to 99% up time.

6.2 Ohmage server

In this section, we are going to look at the server, and see how the server is built. Ohmage back-end consist of various applications and built like stack with application layers. All information about the users, teams, user roles, campaigns and responses, are stored in the database layer of Ohmage back-end.

The database layer in Ohmage is a MySQL database. MySQL [49] is a Relational Database System, which means the data is stored in rows and tables. Ohmage also supports MariaDB, which is a improved version of MySQL[1]. Ohmage database contains many different tables, and we will not present them all here, but some of the most important tables we are using, is presented and described in Table 6.1.

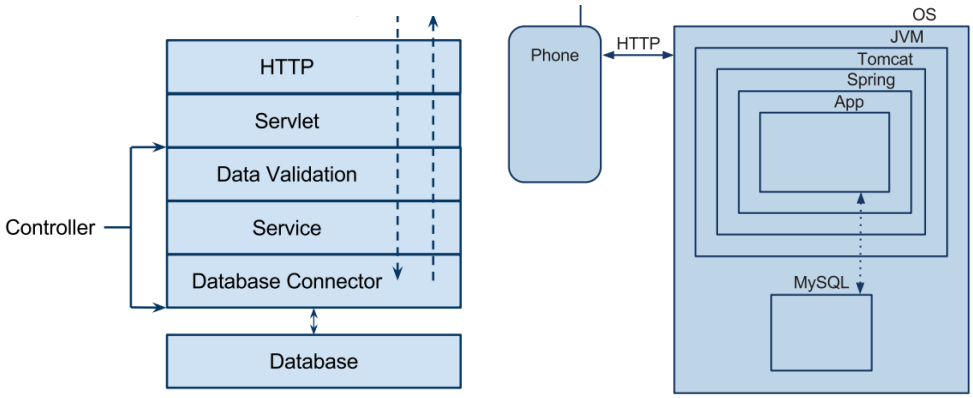
[1]MySQL vs MariaDB - `https://mariadb.com/kb/en/mariadb/mariadb-vs-mysql-features/`

Table	Description
preference	Meta information about the back-end instance
user	User information such as username and password
class	Information about a class(team)
user_class	Connection between a user(player) and a class(team)
survey_response	All collected responses
campaign	Information about campaigns and surveys
campaign_class	Connection between a campaign and a class(team)

Table 6.1: Few tables from Ohmage database

Above the database layer, we have an application layer. The application layer is built to prevent users from accessing the database directly. Allowing a user to insert directly to the database, without validating the user, and validating the data from the user, can damage the database. With the application layer, we can validate, and define business logic, before the data is inserted or fetched from the database.

Ohmage server provides a public API over HTTP protocol. Using HTTP methods such as GET and POST, to access and provide data from the server. The response from the server is presented in JSON format. In short, Ohmage server provides a public web service over HTTP protocol, using JSON [50]. The web service in Ohmage is built with Java, and running in a java servlet[2] container within the Java Virtual Machine(JVM). Ohmage is recommended to run with Apache Tomcat [51], Apache Tomcat is an open source implementation of a java servlet container running in Java Virtual Machine. Ohmage server application is dependent on Spring Framework[3] for dependency injection, and Java Database Connectivity(JDBC) to communicate with the database. Illustration of Ohmage-server can be found on Figure 6.1



(a) Ohmage module layers and how HTTP requests is handled

(b) Ohmage server architecture

Figure 6.1: Figure of Ohmage web application [11]

Ohmage provides several end points, which offer various services. Our system is only using four API categories in Ohmage, class API, campaign API, Authentication API and Survey response API. Most of the required parameters to be sent within a request is username, client name and hashed password, or an authentication token. The data is responded in JSON (JavaScript notation). If a requests fails, the server respond with a detailed message on what went wrong.

The *authentication API* provides two types of authentication methods, stateless and statefull. Statefull authentication returns a token from the server, which can be used to request other API endpoints. *Class API* is used to get information about which class(team) a user belongs to, and which privileges a user have. For coaches who is privileged, can view information about the whole class(team), which is a

[2]A servlet is a Java programming language class that is used to extend the capabilities of servers that host applications accessed by means of a request-response programming model. `http://docs.oracle.com/javaee/6/tutorial/doc/bnafe.html`, Accessed= 2015-05-11

[3]Spring - `http://spring.io/`

list of members of the class(team). For uploading and pulling surveys, we use the *Campaign API*, which provides information of about a campaign and the all surveys in the campaign. The response object is presented in JSON format, but the campaign and survey data is in XML format. Clients who fetch campaign data from the server, needs to interpret the XML formatted schema. For response uploading, we are using Ohmage's *survey manipulation API*, here we can upload our responses, and read uploaded responses. A player can only read their own responses, but a coach can fetch and read all players data. Other endpoints can be found in Ohmage's documentation [52].

6.3 Cloud services

Throughout this thesis, we have been hosting our Ohmage-server at University of Tromsø. We never experienced any problems with their hosting through out thesis, until the problems mentioned in Section 6.1. Under application development, we did not think that the server could go offline. Hence, we wanted to explore and try cloud hosting, and we think that problems we had with UIT, or other self hosted environment, can be solved with cloud hosting. Cloud computing is hardware, systems software and application delivered as a service over Internet [53]. In other words, cloud computing is just another computer service, manged over the internet.

One of the benefits of picking cloud providers, to host our ohmage server application, is that most of the cloud services can provide 99% uptime[4]. Another beneficial with using cloud services, they offers scalability, which means we can scale our system and increase hardware specifications to our needs. Let the cloud service handle infrastructure and hardware upgrade, might even be more cost efficient than self hosting.

Within cloud computing, we have three cloud service models [54]:

- Software as a Service(SaaS): where we can deploy our application to the provider's cloud infrastructure, but we can not control anything else like operating system, hardware, storage or underlying infrastructure. An example of this is web servers, which serves HTML pages.

- Platform as a Service(PaaS): where we can host our application using the supported build environment from the provider, allow the developer control the application and create the developer's own configurations to the application. But the developer can not control the operating system and hardware, and the underlying infrastructure.

- Infrastructure as a Service(IaaS): where we are allowed to manage and control the operating system, storage and applications, but not control underlying infrastructure cloud structure.

According to Computer Business Revieew[5] the most popular IaaS is Amazon Web Services and Microsft Azure. In additional to Amazon and Azure, we wanted to try the cloud hosting service, Digital Ocean. The requirements for running Ohmage application was Tomcat servlet application, and a MySQL database to store data. Following java EE(Enterprise Edition) documentation, recommended system requirements for running a Java server, is 1 GB memory for non-windows platforms, 2GB for windows platforms, and at least 500 MB disk storage [55]. MySQL did not have any requirements, but we only wanted to measure the network latency to these cloud services, and therefore, data storage capacity it's not considered. Since, we want to measure network latency to various cloud services and compare to Tromsø's server, we should pick the servers that are closest to Tromsø to get lowest possible latency, which means European servers is preferred.

6.3.1 Cloud providers

Amazon web services(AWS) is a cloud services who offers various cloud services, all the services can be found on Amazon's home page [56]. We mainly focused on Amazon Elastic Compute Cloud(EC2).

[4]Networkworld, accessed 2015-04-13 - http://www.networkworld.com/article/2866950/cloud-computing/which-cloud-providers-had-the-best-uptime-last-year.html

[5]accessed=2015-04-14 - http://www.cbronline.com/news/enterprise-it/it-services/10-top-cloud-computing-providers-for-2014-4401618

Which is a realizable virtual computer environment, customized through a web interface [57]. Here, we can pick any operating system we want, and install any application we needed. Amazon server is available in Ireland and UK [58].

Microsoft Azure is another known cloud service, that offer various of cloud services. On the same level as Amazon, we are only interested in their virtual machine, hosted in their cloud environment. Azure is offering most of the same features as Amazon. With a virtual cloud machine, we can pick between various operating systems, and install applications as we want. In Europe is Azure providing two hosting locations, Ireland and Netherlands [59].

The last cloud services we want to study, is Digital Ocean. Digital Ocean is not as known as Azure and Amazon. They provide Solid-state-Drive(SSD) only Hard drives, and have a simple control panel. They are saying that, everyone can get a virtual machine up and running in as little as 55 seconds [60]. Digital Ocean is offering more hosting locations, than Azure and Amazon. Digital Ocean provides hosting in Netherlands, UK and Germany [61].

Hardware specifications and prices is presented on Figure 6.2(a),6.2(b) and 6.2(c). We have only presented prices and hardware on the minimal requirement our hosting. Our goal with testing Ohmage in cloud, was to find which cloud services who could provide lowest latency,and we tested the latency from Simula Research Labaratory and Telenor's Mobile Network on 3G.

6.4 Cloud hosting latency

In our test environment in the cloud, we ran Ubuntu 14.04, with tomcat 7 and MySQL server version 5.5. On amazon, we ran t2.micro instance, with 1 core CPU and 1 GB ram, located in Ireland. Azure ran on A1 Standard tier, with 1 core CPU and 1.75 GB RAM in Ireland. And Digital Ocean ran on 10$/month, with 1 GB memory, and 1 core CPU.

We measured the average latency and compared the results, with the Ohmage-server instance hosted at University of Tromsø, Amazon, Azure and Digital Ocean. We used Apache JMeter as our testing tool, and we performed the latency test with 10 concurrent connections, and sent 100 requests for each connection thread, we ran the test from Simula Research Laboratory[6]. And on the mobile network, we ran less requests due to limited available mobile network resources. We only sent 10 requests in one single connection thread. The test were done on an iPhone 4S with Telenor Mobile Network, using 3G. We measured the latency for three different Ohmage endpoints, (1) Server configuration information, (2) Stateless authentication, (3) Fetching one RPE response from a test user.

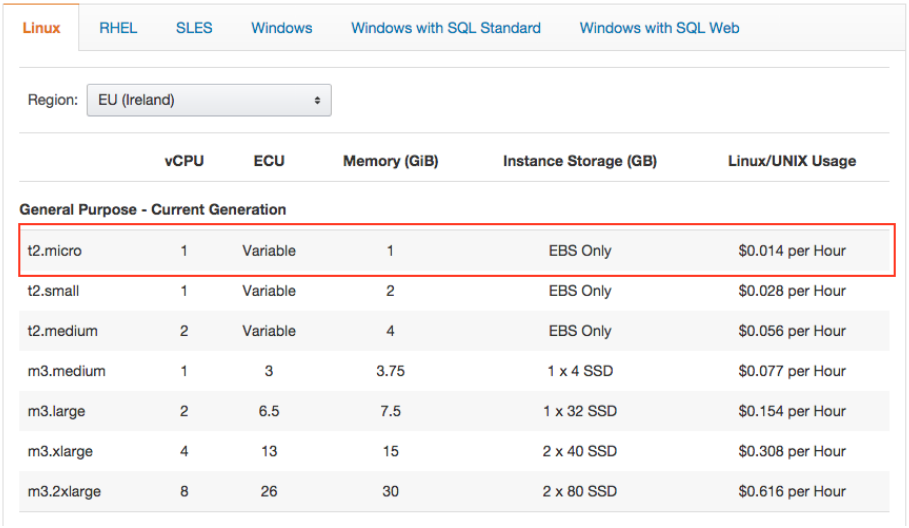
6.4.1 Wired network

From wired network, we see that Tromsø provided outstanding and lowest latency. The primary goal with this study, was to compare the cloud services, and have Tromsø as a basis point. Amazon and Digital Ocean provided lower average latency than Azure overall. Digital Ocean have lower latency on configuration read. But on stateless authentication, which is more important for our applications, Amazon provided lower latency than the others. For our system, this means the users can authenticate faster. For reading responses, were both Amazon and Digital ocean very close on providing lowest latency. See Figure 6.3 and Table 6.2 for result.

Host	Configuration read	Stateless authentication	Read one response
Amazon	92 ms	2987 ms	73 ms
Azure	108 ms	4503 ms	129 ms
Digital Ocean	63 ms	3680 ms	75 ms
University of Tromsø	48 ms	543 ms	43 ms

Table 6.2: Average latency on wired network

[6]Fornebu, Norway

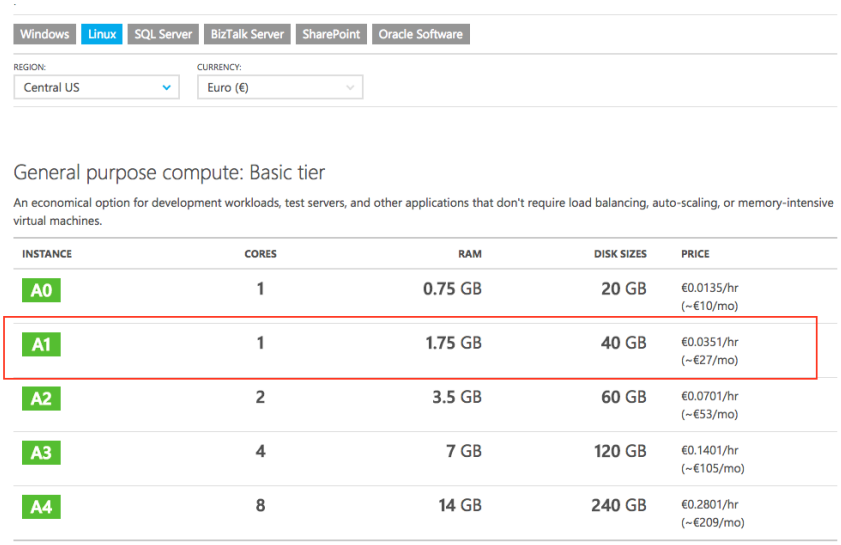


Linux | RHEL | SLES | Windows | Windows with SQL Standard | Windows with SQL Web

Region: EU (Ireland)

	vCPU	ECU	Memory (GiB)	Instance Storage (GB)	Linux/UNIX Usage
General Purpose - Current Generation					
t2.micro	1	Variable	1	EBS Only	$0.014 per Hour
t2.small	1	Variable	2	EBS Only	$0.028 per Hour
t2.medium	2	Variable	4	EBS Only	$0.056 per Hour
m3.medium	1	3	3.75	1 x 4 SSD	$0.077 per Hour
m3.large	2	6.5	7.5	1 x 32 SSD	$0.154 per Hour
m3.xlarge	4	13	15	2 x 40 SSD	$0.308 per Hour
m3.2xlarge	8	26	30	2 x 80 SSD	$0.616 per Hour

(a) Amazon EC2 Basic tier, prices [62]



Windows | Linux | SQL Server | BizTalk Server | SharePoint | Oracle Software

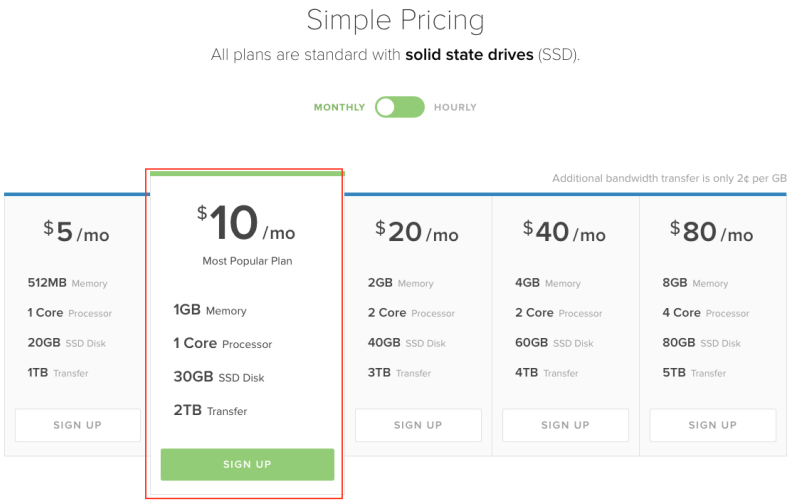
REGION: Central US CURRENCY: Euro (€)

General purpose compute: Basic tier

An economical option for development workloads, test servers, and other applications that don't require load balancing, auto-scaling, or memory-intensive virtual machines.

INSTANCE	CORES	RAM	DISK SIZES	PRICE
A0	1	0.75 GB	20 GB	€0.0135/hr (~€10/mo)
A1	1	1.75 GB	40 GB	€0.0351/hr (~€27/mo)
A2	2	3.5 GB	60 GB	€0.0701/hr (~€53/mo)
A3	4	7 GB	120 GB	€0.1401/hr (~€105/mo)
A4	8	14 GB	240 GB	€0.2801/hr (~€209/mo)

(b) Microsoft Basic tier for Linux hosting, prices [63]



Simple Pricing

All plans are standard with **solid state drives** (SSD).

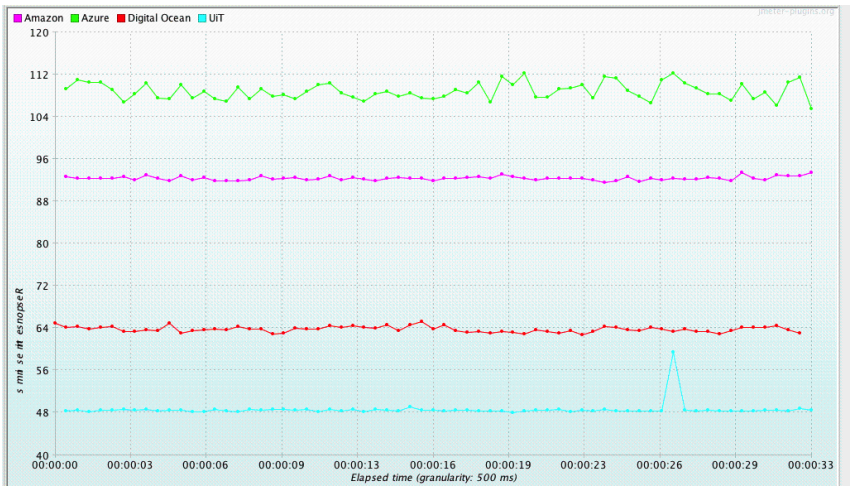
MONTHLY HOURLY

Additional bandwidth transfer is only 2¢ per GB

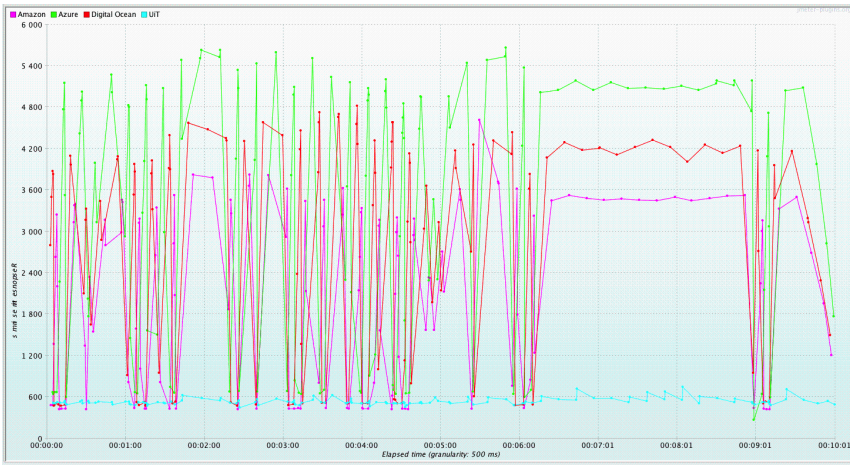
$5/mo	$10/mo	$20/mo	$40/mo	$80/mo
	Most Popular Plan			
512MB Memory	1GB Memory	2GB Memory	4GB Memory	8GB Memory
1 Core Processor	1 Core Processor	2 Core Processor	2 Core Processor	4 Core Processor
20GB SSD Disk	30GB SSD Disk	40GB SSD Disk	60GB SSD Disk	80GB SSD Disk
1TB Transfer	2TB Transfer	3TB Transfer	4TB Transfer	5TB Transfer
SIGN UP	SIGN UP	SIGN UP	SIGN UP	SIGN UP

(c) Digital Ocean prices [64]

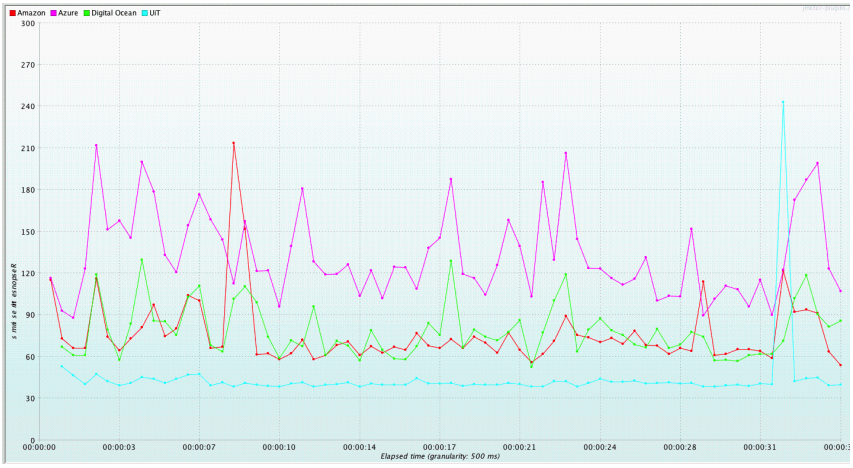
Figure 6.2: Cloud services



(a) Configuration read



(b) Stateless authentication

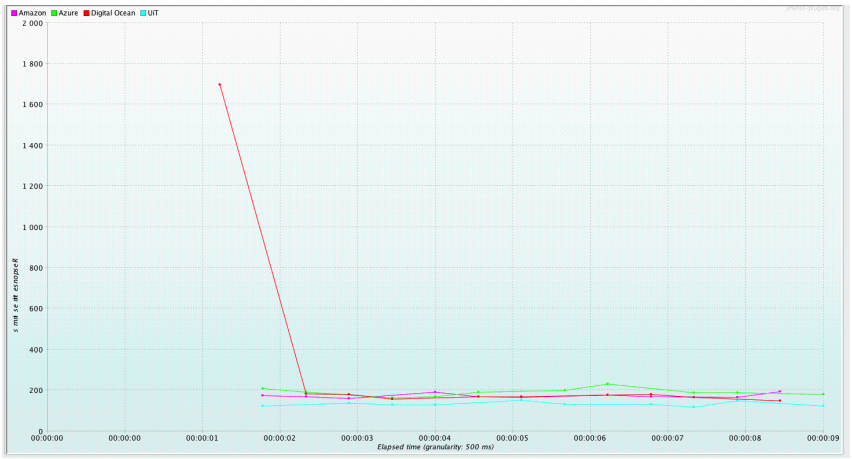


(c) Response read

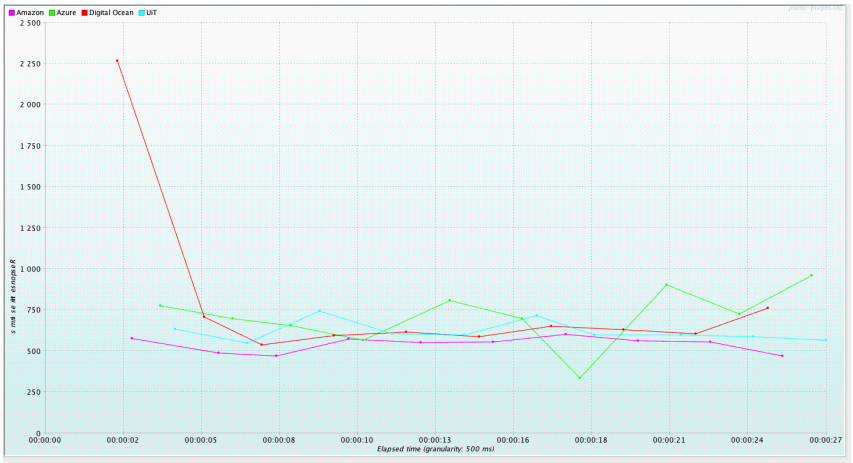
Figure 6.3: Latency on wired network

6.4.2 Mobile broadband network

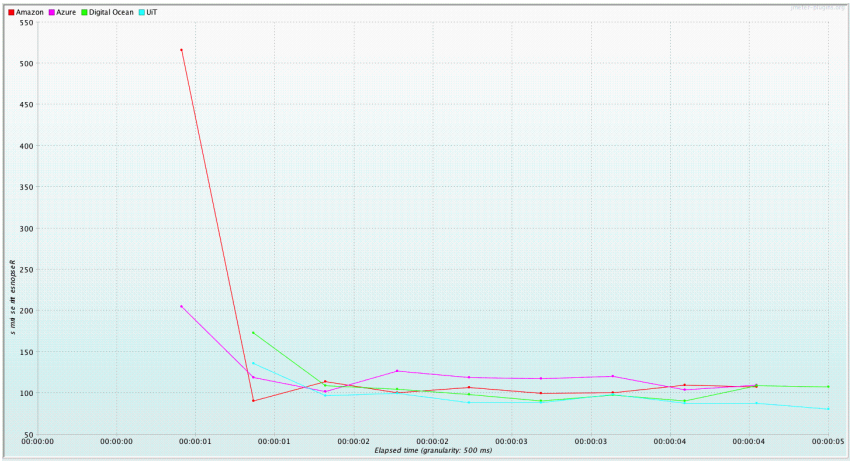
Latency through the mobile network, was significantly lower on stateless authentication, than wired network. But, configuration info had higher latency than wired network. The most surprised findings was that, Amazon had lower latency than Tromsø on stateless authentication. In additional, we could see that cloud services and University in Tromsø, have slightly latency difference over the mobile network, compared to wired network. See Figure 6.4 and Table 6.3 for result.



(a) Configuration read



(b) Stateless authentication



(c) Response read

Figure 6.4: Latency on mobile broadband network

6.4.3 Conclusion

Our goal with the latency measure, was to see which of these services could provide lowest latency from a wired network access, but also from a limited mobile broadband network. Since most of our users are using their mobile phone to report data, providing them low latency, make the system in general faster. Based on the results, we conclude that cloud service is a good solution when we are reliant latency. In additional cloud services can guarantee up to 99% up time. University of Tromsø provided the best latency from wired network, but if we had to pick a cloud service, would Amazon the one who provided

Host	Configuration read	Stateless authentication	Read one response
Amazon	173 ms	540 ms	144 ms
Azure	191 ms	713 ms	123 ms
Digital Ocean	323 ms	796 ms	108 ms
University of Tromsø	132 ms	620 ms	95 ms

Table 6.3: Average latency from Mobile network

lowest latency overall.

6.5 Server improvements

6.5.1 Authentication token

One of the biggest struggle we had in the beginning with pmSys, we wanted to change the expire timer for the authentication token. The expiry timer for the authentication token was hardcoded into the server application (see Figure 6.5), and sat to 15 minutes.

```
/**
 * User storage. User objects are mapped to unique ids. Avoids dependencies on
 * JEE session management. The lifetime param set on construction controls how
 * long User objects stay active.
 * @author Joshua Selsky
 */
public final class UserBin{
    /**
     * This is the length of an authentication token.
     */
    public static final int LIFETIME = 1000 * 60 * 15;

    // Source: https://github.com/ohmage/server/blob/master/src/org/ohmage/cache/UserBin.java
    // More ..
}
```

Figure 6.5: Auth token timer was hardcoded to 15 minutes.

The reason we wanted to extend this timer, is to relief the user and applications from refreshing the authentication token every 15 minutes. When the timer expires for a token, the user has to re authenticate to get a new token. The only way to refresh the authentication token was to do a random request to the server with the authentication token, and then, the timer will be refreshed. We moved the timer value from the source code, into the preference table in the database. When the server starts, the authentication will be set, by fetching the timer from the database. With the new solution, web applications and tokens is valid longer than 15 minutes. Result is presented in Figure 6.6

```
select * from preference where p_key='auth_token_time';
+-----------------+----------+
| p_key           | p_value |
+-----------------+----------+
| auth_token_time | 10800000|
+-----------------+----------+
```

Figure 6.6: Auth-token in database

6.5.2 Validating users

When we created the pmSys mobile application and pmSys web portal, we also implemented a push message service. The problem with the push message service was it did not provide any form for validation when the user subscribe to the service. We could not check if the user was an actual user

in Ohmage. Ohmage does not provide a way to validate whether the username and password is valid or not. The only to validate a user, was to do a stateless authentication with the username and clear-text password to the server. But we did not want to store the password in clear text in the applications, because we were afraid of possible exposures. We extended the endpoint on the server, to check whether the sent password was connected to the provided user. The new extended endpoint provided significant lower latency compared to stateless authentication. With the new validation endpoint, the latency was low as 2 ms, on the same server was the stateless authentication latency over 300 ms. Providing a validating endpoint might leads to brute force attacks, but considering security issues, were not considered in this thesis. However, a suggestion to a solution would have been to limit the total times a user perform a validation.

6.5.3 Improved data fetching

We wanted to improve the way data is sent from the server. In our web portal and mobile application, we have done a lot of necessary parsing on responses from Ohmage, such as organizing the responses, and connect the correct prompts to the correct values and date's, see example of the old response in Figure 6.7. In Figure 6.8, we have illustrated our new implemented response read. Here we can see that the data is presented in a much cleaner structure, and make the parsing and interpreting for a developer much easier.

6.5.4 Conclusion

We ran a network measure on our new implemented endpoints, and the result is presented in Table 6.4. We can see that our new way validation of username and password only used 2 ms. In addition, the new improved response read, reduced the amount of data transferred by over 50%. The tests were performed on the same machine as the testing environment.

Endpoint	**Latency**	**Bytes**
Stateless authentication	348 ms	346 bytes
Validate username and password	**2 ms**	280 bytes
Read responses	17 ms	2335 bytes
New improved read responses	16 ms	**1080 bytes**

Table 6.4: Average latency and bytes transferred with new endpoints

6.6 Summary

We conclude that hosting in cloud gave lower latency on mobile network than wired network. Reasons to pick cloud services over self-hosting are, possible cost efficient, guaranteed up time, scalability, backup, infrastructure transparency. The only time we should consider self-hosting is when we need to host the data ourselves and be the owner of the data, or if we want to have low latency by having the server close to our location. We have also presented four new improvements for the server, and this was done to prove that it exist many improvements for the server.

```
{
  "result" : "success",
  "data" : {
     "urn:ohmage:context:timestamp" : {
      "values" : [
        "2015-04-25 21:54:28",
        "2015-03-31 18:26:20",
        "2015-03-24 22:27:52"
      ]
    },
    "urn:ohmage:prompt:id:srpeLength" : {
      "values" : [
        90,
        90,
        93
      ],
      "context" : {
        "display_label" : "Duration",
        "id" : "srpeLength",
        "prompt_type" : "number",
        "default" : 90,
        // ... and more
      }
    },
    "urn:ohmage:prompt:id:srpeWorkload" : {
      "values" : [
        0,
        1,
        4
      ],
      "context" : {
        "display_label" : "Workload",
        "id" : "srpeWorkload",
        "prompt_type" : "single_choice",
        // ... and more
       }
    },
    "urn:ohmage:prompt:id:srpeType2" : {
      "values" : [
        "NOT_DISPLAYED",
        "NOT_DISPLAYED",
        "NOT_DISPLAYED"
      ],
      "context" : {
        "display_label" : "Session",
        "condition" : "(srpeType1 == 1) or (srpeType1
              == 2)",
        "id" : "srpeType2",
        "prompt_type" : "single_choice",
        // ... and more
      }
    },
    "urn:ohmage:prompt:id:srpeType1" : {
      "values" : [
        0,
        0,
        0
      ],
      "context" : {
        "display_label" : "Session",
        "id" : "srpeType1",
        "prompt_type" : "single_choice",
       // ... and more
      }
    },

  }
}
```

Figure 6.7: Example of response read in Ohmage

```
{
  "result" : "success",
  "data" : {
    "srpe" : {
      "kennet" : [
        {
          "time" : "2015-04-25",
          "srpeType1" : "0",
          "srpeWorkload" : "0",
          "srpeType2" : "NOT_DISPLAYED",
          "srpeLength" : 90
        },
        {
          "time" : "2015-03-31",
          "srpeType1" : "0",
          "srpeWorkload" : "1",
          "srpeType2" : "NOT_DISPLAYED",
          "srpeLength" : 90
        },
        {
          "time" : "2015-03-24",
          "srpeType1" : "0",
          "srpeWorkload" : "4",
          "srpeType2" : "NOT_DISPLAYED",
          "srpeLength" : 93
        },
      ]
    }
  }
}
```

Figure 6.8: Optimized response read response

Chapter 7

Conclusion

In this chapter, we summarize and present our achievements in this thesis. Then, we present possible future work.

7.1 Summary

In this thesis, we have presented an implemented monitoring system for collecting, processing and presenting data. The system has proved to be an easier approach towards monitoring football athletes, by collecting subjective self-report data. The combination of mobile application to report data, web portal to monitor and analyze collected data, and a back end server to handle and process reported and present data have formed our Player Monitoring System. The players do not need to report by paper or excel sheets anymore. Thus, they can self-report at anytime from anywhere, and report as many times as they want, using their mobile application. The coaches can focus on analyzing and interpret the visualized data through a web portal, rather than use time on processing and organize reported data, and use third part programs such as excel to visualize collected data. We have also presented a unidirectional communication channel, where a coach can communicate with a player. A coach can also schedule custom messages to be sent to a player at certain time. At last, we have analyzed and studied the possibility of deploying the server in cloud.

7.2 Main Contributions

The main contribution for our system, we have implemented a working monitoring system that is in use by football teams in Norway. We had four teams using the first version of pmSys, but due to lack of missing features and self-report a new system was needed. With the new version of pmSys and five football teams using the new version, we have increased the total response tremendously with the new system. Our system got media's attention after the Norwegian national team started to use it. In addition, the mobile application has been downloaded over 190 times on iOS and Android. Our system is in use daily and we have over 50 players reporting every day. We have also studied the possibility to deploy our system in Cloud to provide users high up time.

We conclude that our system made it easier for a team to monitor a players load, wellness and injury. We have digitized the approach to collect data through questionnaires by using a mobile application to self-report. The submitted data uploads automatically to a distributed server, where the data is processed and stored securely. The server has removed a coach's need for processing and organizing collected data from players. The coaches no longer need to interpret collected data through pen and paper anymore. The system automatically organizes and processes the reported data. The coach only needs to deal with the web portal, where the coach can view reported data presented in graphical charts, and focus on analyzing and monitoring. The coach can also send instantly push messages right to the players. We have made a Player Monitoring System, pmSys.

7.3 Future work

In this section, we present possible future work and further improvements for our system.

7.3.1 Mobile application

Chat system The mobile application is the core of pmSys. In our application, we wanted to implement a bidirectional communication channel between the coach and the player. This way, the player can respond to the coach's message. We only want the communication to be Coach to player, and not player to player.

Custom message prompt We wanted to let the player provide a custom message after the survey is done, this is to describe reported data further.

Multiple language support Language support is highly demanded for our application. The users were both English and Norwegian speaking. Some words in the surveys were hard to understand. To solve this problem, we want to have a localization file, where every word is translated and then presented to the user.

Encryption and localstorage In pmSys now, we are storing all information and data in JSON files. A better approach would be to save data in a SQL database on the phone. This way, the data cannot be easily read by the public. Another improvements we should have done, is to encrypt the whole file system. Making the stored information protected from public exposion.

Native application If we want to archive higher performance on for example calculations, we should consider native application development. Native development provides access to native API features, rather than running it through a container.

Indicate finished surveys Wellness surveys are conducted daily, and injury surveys are done once a week. It would be nice if we can indicate which survey have been done for the week or the day. This way, the player no longer need to check whether he have conducted a survey for the day or not.

7.3.2 Web portal

General web portal The web portal now is created for coaches to monitor and analyze the players. We want to make this portal more for general use, where the players can log in and monitor their reported data.

Converting user id's All username's is encrypted with a randomized generated string. Only the coaches know which username that is connected to which player in real life. The coaches found it hard to read the encrypted username's in the web portal. Improvement can be done by converting the encrypted user id's, and show the real name only in the coach's browser. This needs be done on client side, without sending any information to the server.

Upload campaign Would be great if we could upload a campaign through the web portal, rather than through Ohmage's web portal.

Machine learning It is possible to extend the system with machine learning on collected data. The machine can then detect when a player is in danger to be injured, and tell the coach.

Artificial intelligent We can combine machine learning with AI to automatically detect injuries, and for example tell the player to relax for few days. And other AI related stuff.

Chat system We mentioned this feature in section above, we wanted to let the coach chat with a specific player through the web portal, and then, have player answer through their mobile application.

Campaign creator It would be nice for the coach and coordinators if they could create and edit campaign and surveys directly in the web portal.

Planned RPE The coaches must use the mobile application to add planned RPE, a better approach would be to add planned RPE values directly from the web portal.

Visualization We could let the coach specify which chart and how the data should be presented from the web portal. This way the coach can create their own graphical charts.

7.3.3 Server

Text style in campaign In the campaign, it would be great if we could style text in the XML file. For example <Question> Rate your <b>readiness<b> to play</Question>, and then, readiness will be presented as bold text in the application.

Third part devices pmSys would be a better monitoring system if we could integrate third party monitoring devices, and present the data. The coach can then compare subjective data versus objective data.

Other database technologies We can study other RBDMS such as PostgreSQL, or noSQL database such as MongoDB, and see if we can find any improvements for the server by replacing the MySQL database.

Security issues Study and improve security weakness of Ohmage server.

Update responses As for now, it does not exist an easy way to update values for all responses. For example, if the RPE scale was distributed with a number from 1 to 7. Then suddenly, the coach want to change the RPE scale from 1 to 5. If we want to do this with Ohmage, we have to first update RPE scale defined in the campaign, and then, we have to iterate through all saved responses, and update all RPE values and map the 1-7 scale to 1-5 scale.

Appendix A

Source code

All pmSys repositories are private. Access will be granted 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`

A.4 Improved Ohmage server

`https://github.com/knyyy/server`

A.5 Ohmage server

`https://github.com/ohmage/server`

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] Foster Carl, Florhaug JA, Franklin J, Gottschall L, Hrovatin LA, Parker S, Doleshal P, and Dodge C. A new approach to monitoring exercise training. *Journal of Strength and Conditioning Research*, 15(1):109–115, 2011.

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

[3] McLean BD1, Kelly V Coutts AJ, McGuigan MR, and Cormack SJ. Neuromuscular, endocrine, and perceptual fatigue responses during different length between-match microcycles in professional rugby league players. *International Journal of Sports physiology and Performance*, 5:367–383, 2010.

[4] 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. *Br J sports med*, 2012.

[5] Deborah Estrin, K. Mani Chandy, R. Michael Young, Larry Smarr, Andrew Odlyzko, David Clark, Viviane Reding, Toru Ishida, Sharad Sharma, Vinton G. Cerf, Urs Hölzle, Luiz André Barroso, Geoff Mulligan, Adrian Hooke, and Chip Elliott. Internet predictions. *IEEE Internet Computing*, 14(1):12–42, January 2010.

[6] Phyo Min Tun. Choosing a mobile application development approach. *ASEAN Journal of Management & Inniovation*, 1(1):69–74, 2014.

[7] Michael S. Mikowski and Josh C Powell. *Single page web applications: JavaScript end-to-end*. Manning, 2014.

[8] Prototype mobile applications built with ibm worklight for ibm watson. `http://www.ibm.com/developerworks/library/mo-prototype-watson/`. Accessed = 2015-05-13.

[9] Ajax: A new approach to web applications. `https://web.archive.org/web/20080702075113/http://www.adaptivepath.com/ideas/essays/archives/000385.php`.

[10] Expressjs - api. `http://expressjs.com/4x/api.html`. Accessed = 2015-04-07.

[11] John Hicks, Nithya Ramanathan, Donnie Kim, Mohamad Monibi, Joshua Selsky, Mark Hansen, and Deborah Estrin. Andwellness: An open mobile system for activity and experience sampling. In *Wireless Health 2010: Academic and Research Conference,*, 2010.

[12] Ohmage - campaign prompt types. `https://github.com/ohmage/server/wiki/Campaign-Definition#promptTypes`. Accessed = 2015-05-13.

[13] Belastningsovervåking i fotball. `http://www.nih.no/forskning/prosjektarkivet1/forskningsprosjekter-ved-nih/belastningsovervaking-i-fotball/`. Accessed: 2014-10-12.

[14] Norges fotball forbund - tippeligaen. http://www.fotball.no/Landslag_og_toppfotball/Toppfotball/tippeligaen/. Accessed = 2015-05-02.

[15] Alt om fotball - tippeliga 2015. http://www.altomfotball.no/element.do?cmd=tournament&tournamentId=1. Accessed = 2015-05-02.

[16] Statistikk breddefotball. http://www.fotball.no/nff/NFF-nyheter/2014/Statistikk-breddefotball/. Accessed: 2015-03-18.

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

[18] Lovdata. *Lov om behandling av personopplysninger*. Justis- og beredskapsdepartementet, 2000. USIKKER PÅ DENNE. https://lovdata.no/dokument/NL/lov/2000-04-14-31 Accessed = 2015-05-09.

[19] Benjamin Clarsen, Ola Rønsen, Grethe Myklebust, Tonje Wåle Flørenes, and Roald Bahr1. The oslo sports trauma research center questionnaire on health problems: a new approach to prospective monitoring of illness and injury in elite athletes. DENNE MÅ FIKSES.

[20] Blake D. McLean, Aaron J. Coutts, Vince Kelly, Michael R. McGuigan, and Stuart J. Cormack. Neuromuscular, endocrine, and perceptual fatigue responses during different length between-match microcycles in professional rugby league players. *International Journal of Sports Physiology and Performance*, 9:367–383, 2010.

[21] Rated perceived exertion(rpe) scale - cleveland clinic. http://my.clevelandclinic.org/services/heart/prevention/exercise/rpe-scale. Accessed = 2015-04-16.

[22] J. Burke, D. Estrin, M. Hansen, A. Parker, N. Ramanathan, S. Reddy, and m. B. Srivastava. Participatory sensing. In *World Sensor Web*, 2006.

[23] Smartphone's in norway. http://www.medienorge.uib.no/statistikk/medium/ikt/379/. Accessed = 2015-04-18.

[24] Deborah Estrin and Ida Sim. Open mhealth architecture: An engine for health care innovation. *Sciencemag*, 330:759–760, November 2010. Link = http://research.microsoft.com/en-us/um/beijing/events/ms_ubicomp11/mhealth_science.pdf Accessed = 2015-03-12.

[25] mhealth for developers. http://www.vitalwaveconsulting.com/pdf/2011/mHealth.pdf, 2011. Accessed 2015-04-18.

[26] mhealth - getting started. http://www.openmhealth.org/developers/getting-started/. Accessed = 2015-04-18.

[27] Dhis user manual. https://www.dhis2.org/doc/snapshot/en/user/html/dhis2_user_manual_en_full.html#d5e30. Accessed: 2015-03-01.

[28] Milan lab. http://www.acmilan.com/en/club/milan_lab. Accessed: 2014-10-12.

[29] Universitet i tromsø - bruk av teknologi for prestasjonsoptimalisering i tromsø il. http://www.olympiatoppen.no/om_olt/regioner/regionnordnorge/aktuelt/media20277.media. Accessed = 2015-05-04.

[30] 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 extensible end-toend participatory sensing platform. Technical report, UCLA Computer Science, Auguest 2014.

[31] Ohmage - front end wiki. https://wiki.mobilizingcs.org/app/web. Accessed = 2015-04-22.

[32] Ohmage. http://ohmage.org/. Accessed: 2015-03-07.

[33] Mwf mobile web framework - university of california. http://mwf.ucla.edu/. Accessed: 2015-03-07.

[34] Google play - ohmage mwf. https://play.google.com/store/apps/details?id=org.ohmage.mwoc&hl=no. Accessed: 2015-03-23.

[35] Cordova. https://cordova.apache.org/. Accesed = 2015-03-27.

[36] Cordova plugins registry. http://plugins.cordova.io/. Accessed = 2015-04-27.

[37] Glenn E. Krasner and Stephen T. Pope. A cookbook for using the model-view controller user interface paradigm in smalltalk-80. *J. Object Oriented Program.*, 1(3):26–49, August 1988.

[38] Ionic documentation overview. http://ionicframework.com/docs/overview/. Accessed: 2015-03-28.

[39] Sandeep Panda. *AngularJS: Novice to Ninja*. 2014.

[40] Angular - ngtouch documentation. https://docs.angularjs.org/api/ngTouch. Accessed = 2015-03-29.

[41] X2js - working with array. https://code.google.com/p/x2js/wiki/WorkingWithArrays. Accessed = 2015-03-31.

[42] Apple developer - app life cycle. https://developer.apple.com/library/ios/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/TheAppLifeCycle/TheAppLifeCycle.html. Accesed = 2015-03-02.

[43] Apple push notification service. https://developer.apple.com/library/ios/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/Chapters/ApplePushService.html#//apple_ref/doc/uid/TP40008194-CH100-SW9. Accessed = 2015-03-01.

[44] Google cloud messaging for android. Accessed = 2015-03-01.

[45] Nodejs - home. https://nodejs.org/. Accessed = 2015-04-06.

[46] Nodejs - about. https://nodejs.org/about/. Accessed: 2015-03-10.

[47] Npm home. https://npmjs.com. Accessed = 2015-04-07.

[48] Modulus - nodejs and expressjs sessions. http://blog.modulus.io/nodejs-and-express-sessions. Accessed = 2014-08-02.

[49] Mysql - what is mysql. http://dev.mysql.com/doc/refman/5.5/en/what-is-mysql.html. Accessed = 2015-05-01.

[50] Ohmage server - about the client server protocol and system entites. https://github.com/ohmage/server/wiki/About-the-Client-Server-Protocol-and-System-Entities. Accessed = 2015-02-03.

[51] Ohmage github repository. https://github.com/ohmage/server. Accessed = 2015-04-12.

[52] Ohmage api, top level entities. https://github.com/ohmage/server/wiki/APIs-for-2.x-Top-Level-Entities. Accessed = 2015-05-01.

[53] Nick Antonopoulos and Lee Gillam. *Cloud Computing: Principles, Systems and Applications*. Springer-Verlag London, 2010.

[54] Peter Mell and Timothy Grance. The nist definition of cloud computing. Technical report, Computer Security Division, National Institute of Standards and Technology, 2011.

[55] Oracle - java platform enterprise eidition notes. http://www.oracle.com/technetwork/java/javaee/documentation/javaee7sdk-readme-1957703.html#System_Requirements. Accessed = 2015-04-24.

[56] Amazon web services - products and services. http://aws.amazon.com/products/. Accessed 2015-04-25.

[57] Amazon web services - amazon ec2. http://aws.amazon.com/ec2/. Accessed = 2015-04-24.

[58] Amazon - global infrastructure. http://aws.amazon.com/about-aws/global-infrastructure/. Accessed = 2015-04-25.

[59] Microsoft azure - regions. http://azure.microsoft.com/nb-no/regions/. Accessed = 2015-04-25.

[60] Digital ocean - features. https://www.digitalocean.com/features/technology/. Accessed = 2015-04-25.

[61] Digital ocean - reliability. https://www.digitalocean.com/features/reliability/. Accessed = 2015-04-25.

[62] Amazon - pricing. http://aws.amazon.com/ec2/pricing/. Accessed = 2015-04-25.

[63] Microsoft azure - virtual machines pricing. http://azure.microsoft.com/en-us/pricing/details/virtual-machines/. Accessed = 2015-04-25.

[64] Digital ocean - reliability. https://www.digitalocean.com/pricing/. Accessed = 2015-05-17.