MINISTRY OF EDUCATION AND SCIENTIFIC RESEARCH

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

ATHLETES' HEALTH MONITORING SYSTEM

LICENSE THESIS

Graduate: **Stefania GLIGA**

Supervisor: **Assist. Eng. Cosmina IVAN**

2016

MINISTRY OF EDUCATION AND SCIENTIFIC RESEARCH

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

DEAN,
Prof. dr. eng. Liviu MICLEA

HEAD OF DEPARTMENT,
Prof. dr. eng. Rodica POTOLEA

Graduate: **Stefania GLIGA**

ATHLETES' HEALTH MONITORING SYSTEM

1. **Project proposal:** The purpose of the project is to define and construct a system for monitoring the health state of athletes. The system is designed for both athletes and coaches with the objective of replacing pen and paper analysis of data and offering the ability of visualizing the results in graphical form. The project's main objectives are creating a mobile application as easy to use as possible because of the nature of an athlete's schedule and creating a web portal with rich features in order for the coach and medical staff to monitor the athletes' health situation.

2. **Project contents:** Table of contents, Introduction, Project objectives, Bibliographic research, Analysis and Theoretical Documentation, Detailed Design and Implementation, Testing and Validation, Users' manual, Conclusions, Bibliography, Appendix.

3. **Place of documentation**: Technical University of Cluj-Napoca, Computer Science Department

4. **Consultants**: Assist. Eng. Cosmina IVAN

5. **Date of issue of the proposal:** November 1st , 2015

6. **Date of delivery:** June 30th, 2016

Graduate: _____

Supervisor: _____

MINISTRY OF EDUCATION AND SCIENTIFIC RESEARCH
TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

Declaraţie pe proprie răspundere privind autenticitatea lucrării de licenţă

Subsemnatul(a) **Gliga Stefania**, legitimat(ă) cu CI seria KX nr. 811028 CNP 2930215125772, autorul lucrării ATHLETES' HEALTH MONITORING SYSTEM elaborată în vederea susţinerii examenului de finalizare a studiilor de licenţă la Facultatea de Automatică şi Calculatoare, Specializarea Calculatoare Engleza din cadrul Universităţii Tehnice din Cluj-Napoca, sesiunea iulie 2016 a anului universitar 2015-2016, declar pe proprie răspundere, că această lucrare este rezultatul propriei activităţi intelectuale, pe baza cercetărilor mele şi pe baza informaţiilor obţinute din surse care au fost citate, în textul lucrării, şi în bibliografie.

Declar, că această lucrare nu conţine porţiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislaţiei române şi a convenţiilor internaţionale privind drepturile de autor.

Declar, de asemenea, că această lucrare nu a mai fost prezentată în faţa unei alte comisii de examen de licenţă.

In cazul constatării ulterioare a unor declaraţii false, voi suporta sancţiunile administrative, respectiv, *anularea examenului de licenţă*.

Data	Nume, Prenume
29.06.2016	Gliga Stefania
	Semnătura

Table of Contents

Chapter 1. Introduction 1

1.1. Project context 1

1.2. Problem domain 2

1.3. Project structure 3

Chapter 2. Project Objectives 4

2.1. Primary objective 4

2.2. Specific objectives 4

Chapter 3. Bibliographic Research 7

3.1. Subjective self-reporting vs objective measures 7

3.2. Medical background in sport 8

3.2.1. Training Load 8

3.2.2. Wellness 9

3.2.3. Monitoring illness and injury 10

3.3. mHealth 10

3.4. Similar Systems 11

3.4.1. Ohmage 11

3.4.2. Player Monitoring System 12

3.4.3. Athlete monitoring 13

3.4.4. Comparison between Ohmage and Player Monitoring System 13

Chapter 5. Analysis and Theoretical Foundation 15

5.1. Conceptual architecture 15

5.2. System requirements 16

5.2.1. Functional requirements 16

5.2.2. Nonfunctional requirements 17

5.3. Use case specification 18

5.3.1. Actors 18

5.3.2. Use cases 19

5.4. Technological perspective 23

5.4.1. Ruby programming language 23

5.4.2. Ruby on Rails (RoR) 23

5.4.3. PostgreSQL 25

5.4.4. Android 25

5.4.5. Bootstrap 26
5.4.6. HAML 27
5.4.7. Devise 27
5.4.8. Reform 28
5.4.9. Rpush 28
5.4.10. Chartkick 28
5.4.11. Volley 29
5.4.12. jQuery 29
5.4.13. GCM 30
Chapter 6. Detailed Design and Implementation 31
6.1. Implementation of the web application 31
6.1.1. Architecture 31
6.1.2. Authentication 35
6.1.3. Implementation details 35
6.2. Implementation of the Android application 38
6.2.1. Architecture 38
6.2.2. Authentication 39
6.2.3. Implementation details 40
6.3. Database Design 42
6.4. Deployment 44
6.5. Version control 45
6.6. Cloud deployment 45
6.6.1. Cloud service models 45
6.6.2. Cloud deployment models 46
6.6.3. Heroku 47
Chapter 7. Testing and Validation 48
Chapter 8. User's manual 51
8.1. Web application 51
8.1.1. Installation description 51
8.1.2. User manual 52
8.2. Mobile application 55
8.2.1. Installation description 55
Chapter 9. Conclusions 56
9.1. Main contributions 56

9.2. Future work 56
9.2.1. Mobile application 56
9.2.2. Web application 57
Chapter 10. Bibliography 58
Appendix 1 List of figures 61
Appendix 2 List of tables 62
Appendix 3 Glossary 63

Chapter 1. Introduction

1.1. Project context

In a world dominated by technology, subject of great interest and great importance in present days, everyone wants to exploit it at full capacity in the areas of interest, making desired information more accessible and manageable. Lately, the world has seen a number of advancements that revolutionized the way people work, live and play. Technology is not only used to increase performance, but also to improve safety, facilitate monitoring and saves humans from the trouble of doing by hand some demanding and exhausting tasks. One of the fields in which technology has played a significant part over the years, is the sports field.

From the beginning of civilization, sports has continuously played an important role in human life, being one of the most disputed aspects in society. The driving force behind sport is the athlete, without whom the sport would not exist. [1] The coach is an important factor in the success and development of athletes, leading them towards their goal. In modern sport era, training for performance has become a specialized science, termed sport science. The field of sport science has led to the development of specialized institutions throughout the world that use scientific methods to enhance sport performance. Enhancing performance through sport science is accomplished via the combined effort of a group of professionals, all with higher education in some aspect of sport. [1]

The world of sport has been changing unceasingly over the years and the use of technology is one of those areas that has made an impact on many sports in the modern days. Technology came to facilitate the ease of performing several tasks that maybe in the past were time-consuming and the results less reliable. These improvements brought by the continuous development of the technology are covering more and more areas of the sports field, making monitoring easier. Thus, information about trainings and athletes' state is more accessible and easier to process for performance improvement purposes. Sports technology, of course is not limited to enhancing performance; it potentially extends into rehabilitation and injury prevention. [2] One of the means by which we can obtain the necessary data for making this possible is monitoring athletes.

Athlete training is a repetitive process which imposes stress on an athlete, modifying his physical and psychological state. Monitoring this process is essential to making the training meaningful and keeping it on track. The most effective training programs rely on a good way of monitoring due to the fact that it increases training effectiveness. The more consistent the monitoring, the more meaningful the information will be. Monitoring training gives the opportunity to evaluate the impact that session had on every athlete present.

One of the simplest means of monitoring training used is keeping a detailed training log. The log is an athlete's personal monitoring tool. It represents the athlete's input about responses to training. Each log, regardless of the sport or person, contains certain basic information. The log should monitor factors outside the training: sleep, diet, and other factors that can have an effect on training. The coach's training log should be as detailed as possible and still practical in order to isolate variables to identify possible patterns as stated in [3].

Nowadays, coaches struggle to find ways to get their athletes to monitor key factors that influence their performance. Information about training volume, training intensity, sleep, stress and nutrition parameters is vital for modifying the training plan on a weekly or daily basis. It is therefore vital that athletes provide this information as it directly enhances the training environment and ultimately improves athlete performance.

1.2. Problem domain

Performance testing in sport science can be a time consuming and physically demanding process, and thus is generally assessed at the beginning and end of a performance enhancement training protocol. Questionnaires are primarily used in the social sciences but, when developed harmoniously with sport science and monitoring protocols, could allow sport scientists a fast and easy assessment of a performance enhancement training protocol and the recovery status of the athletes. The use of questionnaires in social sciences is a common mode of discerning information about a given topic. Specifically, questionnaires are a subjective method of discerning opinions, beliefs and behavior. Additionally, perceptions of an athlete's abilities can directly impact field performance. Because of this, the use of questionnaires in sport provides a method of evaluating the enhancement of sport performance as stated in [1].

The purpose of monitoring a person over time is to obtain information about behavior and habits, by capturing significant data from the person's activities. The data can then be processed and results analyzed and evaluated accordingly. A common approach is to perform these tasks by hand, on paper or on the computer, but the process still requires a lot of time and is extremely impractical in the long run. Collecting, processing and analyzing data implies a time-consuming effort.

By taking advantage of modern technologies and the level of availability of today's resources, monitoring becomes a simple and efficient task. In addition, the data collection process becomes faster and more effective, serving the purpose of monitoring with greater efficacy.

A system that effectively eases the process of data collection, analysis and visualization is proposed and aims to make the monitoring task more efficient. The purpose of this thesis is to create a digital approach to self-reporting and monitoring through questionnaires. Thus, the main focus will be on collecting subjective data through self-reporting that will then be analyzed by coaches and medical staff in order to obtain information about athletes' health state. A mobile application will be used to collect data and then send it to a server where it will be processed and stored securely. A web application will connect to the server and get the stored data in order to present it in charts and graphs. This will facilitate not only the monitoring process but also the coach's effort of having to analyze and evaluate the data. The system provides data in charts that will allow the coach to directly see the results in a form that makes interpretation easier and faster.

Therefore, the envisioned solution is addressed to both athletes and coaches in order to replace the time-consuming operations of monitoring athletes' health state in relation with training sessions. The ultimate purpose is to improve performance and prevent possible injuries.

1.3. Project structure

The following paragraphs will describe the thesis structure, the chapters and their content.

Chapter 1 – Introduction – This chapter contains a short presentation of the problem context and the solution envisioned to solve the problem.

Chapter 2 – Project Objectives – This chapter presents the description of the project main objective but also the secondary objectives proposed to accomplish by the athlete's health monitoring system.

Chapter 3 – Bibliographic Research – This chapter encapsulates the description of concepts approached and their utility within the project context. Algorithms and formulas used in calculating athlete health parameters will be described and detailed. In addition, this chapter provides similarities and differences from the system functionalities point of view, between the proposed system and similar systems available on the market.

Chapter 4 – Analysis and Theoretical Foundation – In this chapter the technologies used to develop the system are structured. Also, the concepts used to create the web application as well as those used to create the mobile application are presented and explained. This chapter displays the functional and non-functional requirements but also contains an identification of use cases for both applications proposed, and the ones considered more relevant are presented in a more detailed manner. The conceptual architecture of the system is represented, showing an overview of the entire system design.

Chapter 5 – Detailed Design and Implementation – The content of this chapter is represented by presenting the architectures of the two applications and detailed presentation of every component part. Deployment diagram, class diagram and database diagram are introduced here. The architectural pattern of each application is detailed and the communication between the modules is presented. Also the push notification functionality is described and explained.

Chapter 6 – Testing and Validation – Testing methods of the applications are present as well as the results of these tests.

Chapter 7 – User manual – This chapter encompasses the minimal physical hardware resources needed for installing the system. The web application and mobile application installation manuals are present as well as the manuals that help the user handle the system.

Chapter 8 – Conclusions – This chapter states the objectives that were accomplished and possible future work.

Chapter 2. Project Objectives

2.1. Primary objective

The purpose of this thesis is to design and implement a software system that optimizes the process of data collection, processing, analysis and visual presentation in order to accomplish the ultimate purpose of an efficient athlete health monitoring. Therefore, the primary objective is to develop a software solution for both athletes and their supporting staff to help increase performance by constant monitoring and keeping track of athletes' subjective feedback after trainings. Reporting subjective data about personal heath state and trainings will help the coach make an accurate assessment of the training sessions and their impact on the athlete.

Relying on the help of the technology, we envision a system that speeds up the monitoring process and thus, increases the accuracy of detecting possible injuries resulted after a training session. A possible injury or beginning of an injury identified early can prevent a health condition that maybe will require time off or will take a long time to heal, resulting in an unwanted outcome that neither the athlete nor the coach benefits from.

A mobile application is created in order to make the athletes have the most efficient user experience. Due to the athletes' lack of technical knowledge, the application has to be as easy to use as possible and because of the nature of their schedule, the response of the interactions have to take as little as possible so the focus can be channeled to obtain a maximum performance during training sessions. This component of the system aims to meet the athletes' needs and serve as a user-friendly tool that improves the self-reporting experience and acts as a motivational factor.

Because the coaches are of equal importance in this monitoring issue, due to the fact that they use and analyze the data in order to monitor the athletes, a web application will be created. The users of this application will be able to view the results and interpret them according to the wanted perspectives. The results will be presented in a way to facilitate the monitoring of the athletes by the coach and the medical staff, which means that they will not need to perform further operations on the data to receive the results they want. Results will be presented in graphical form using charts for better understanding and for faster discovery of any irregularities.

2.2. Specific objectives

General objectives of the envisioned system:

- The system aims to provide an expressive user interface, making the user experience more satisfactory and more appealing. Through this, an application more efficient to use is expected, giving the user the benefit of accomplishing a particular task in less time. An efficient utilization of an application begins in the moment the user gets familiarized with it and knows how to perform desired tasks. An expressive user interface makes this process faster due to the fact that learning how to interact with the system comes more naturally and is straightforward.
- The system will be used by various types of users: athletes, coaches, administrators, medical staff, so it will provide separate functionalities for

each of them. Role management helps the application manage authorization, which enables the users to have restricted access to the resources based on their type. So another objective of the system is to have the capacity of filtering the available resources based on the type of user that requests them.

- The system has to offer the users the possibility of authentication and reset password. The administrator will manage all accounts in the system, will create users, teams, and assign users to teams. After that the user will have the possibility of resetting and recovering the password automatically in case he forgot it.

An approach to self-reporting

The athlete will use a mobile application in order to register information about his health condition after trainings, through surveys. The user will be registered by the administrator and will be assigned to a team. The coach is able to create surveys and make them available to teams, but the athletes won't be able to access them if they are not enabled. The mobile application has to display and submit responses introduced by the user. It will provide a user friendly interface that will help the athlete easily register the survey data and submit it. After submitting responses, the athlete will have the ability to see own results and get a feedback from the system about his condition. The reporting operation is intended to take as little time as possible in order to not make the user experience time-consuming for the athlete.

Data processing

The user submits data through the mobile application to the server where the data is stored securely and is only available to the authorized users. The data processing part will involve the calculation of some basic parameters in order to analyze athletes' state. The system will calculate Rating of Perceived Exertion, wellness and overuse injury from the stored data and make them available to the users of the web application.

Push Notifications

The athletes will benefit from a push notification feature meant to make possible the unidirectional communication between the coach and the athletes. Most of the athletes have a busy schedule and tend to forget to take the surveys in time. Taking surveys at the right time is important in order to get an accurate view of the athletes' state after the training session because the perspective is still fresh. Having a notification system not only solves this problem but also enables the coach to send messages to the athletes, give feedback or make announcements. The coach can send notifications to all members of a team or to a specific athlete. Also notifications can act as reminders and be sent automatically by the system when it's time to take the survey.

Presentation and Visualization

The way data is presented and visualized is significant for serving best the user's requirements. The system provides a way of visualizing the data after it was submitted by the athlete, stored and processed. The web application designated to the coaching and medical staff will display the reported data in visual charts in order to make the coach's job easier when having to analyze and evaluate it. The system becomes more approachable

by processing data and turning it into something meaningful in terms of graphs and charts. The idea is to show the user relevant data in a manner that best aids him in his tasks, instead of overloading him with written information. Two types of graphical visualization are possible. The coach will be able to view data of a specific athlete or of the whole team. In addition, the mobile application will provide feedback data after the survey responses are submitted by the athlete. In this way users can monitor themselves and be aware of their own progress and are motivated to continue using the system.

Chapter 3. Bibliographic Research

3.1. Subjective self-reporting vs objective measures

Monitoring athlete condition is indispensable to guide the coach into creating training programs that best help the athletes evolve and to evaluate each training based on athlete perspective. Monitoring also facilitates the discovery of potential training overload and the detection of any advance towards negative health outcomes that usually result in bad performance. This effect does not favor neither the athlete, nor the coach. The available options to make this monitoring possible are obtained using objective and subjective means.

A study was performed in paper [4] that compares objective and subjective measures of athlete state. No consistent association between these measures that were analyzed, was found. It was discovered that objective measures were not as responsive as subjective measures in a training related context. This resulted from the subjective measures' capacity to detect increase and decrease in training, an undeniable indicator that the subjective measures are certainly responsive to training load. Although the objective measures response was limited in comparison with the subjective measures response, they have a great utility in capturing multiple parameters related with athlete health but the athlete's position in relation with the training session in not necessarily reflected. Objective measures usually consist of physiological and performance computed capacities which can give information about athlete state to increase health condition awareness. The result obtained by this study shows that the response of the subjective measures was consistent with the stress inflicted, fatigue and poor health intensifying proportional with increased training and reducing with decreased training load. The study encourages coaches and athletes to employ self-report measures due to the fact that subjective measures have proven to respond in the athlete health state according to changes induced in trainings.

Athletic training monitoring eases the assessment and adjustment of strategies to optimize performance results. Coaches are given a higher degree of confidence when having to adapt the training load of the sessions, aiming at the ultimate goal of improving performance and in the same time decreasing the risk of illness, injury and overtraining. Self-reporting methods such as surveys and diaries are cheap and simple means to monitor athlete's response to training sessions. Paper [5] states that there is more and more support for the fact that self-reporting measures may be more precise and solid than objective performance measures and focuses on the factors influencing the design of effective self-reporting, minimizing implicit limitations. The study performed recommends a self-reporting design through which to obtain quality and relevant data from the athlete with minimum effort. This means that the questions, number of questions and frequency of completion must be selected carefully along with an efficient employment of technology.

3.2. Medical background in sport

3.2.1. Training Load

As it is stated in paper [6], the ultimate goal of training is to prepare athletes to perform at their best in important competitions. By monitoring training, athletes will receive the desired training effect and will be prepared for competition, whilst minimizing injury and illness. Injury and illness can occur when physical demands outweigh the body's ability to fully recover between training sessions and competitions. Coaches and athletes both benefit from the accurate training load monitoring.

Foster [7] developed the session RPE method of quantifying training load, monotony and strain.

Rating of perceived exertion (RPE) as defined from Wikipedia "is a frequently used quantitative measure of perceived exertion during physical activity"[1]. Sport coaches use this method to measure the training intensity. The RPE value lies in a category of 1 to 10 and is called "the Borg Scale" (Figure 3.1). This value will be requested in a survey in order to get the athlete intensity perception of the training session. The coach can get a glimpse of how the training is perceived by the athlete, which provides a greater awareness of the athlete's physical state condition, but also measures the training degree to a certain extent. The athlete should answer the survey as soon as possible after the training session.

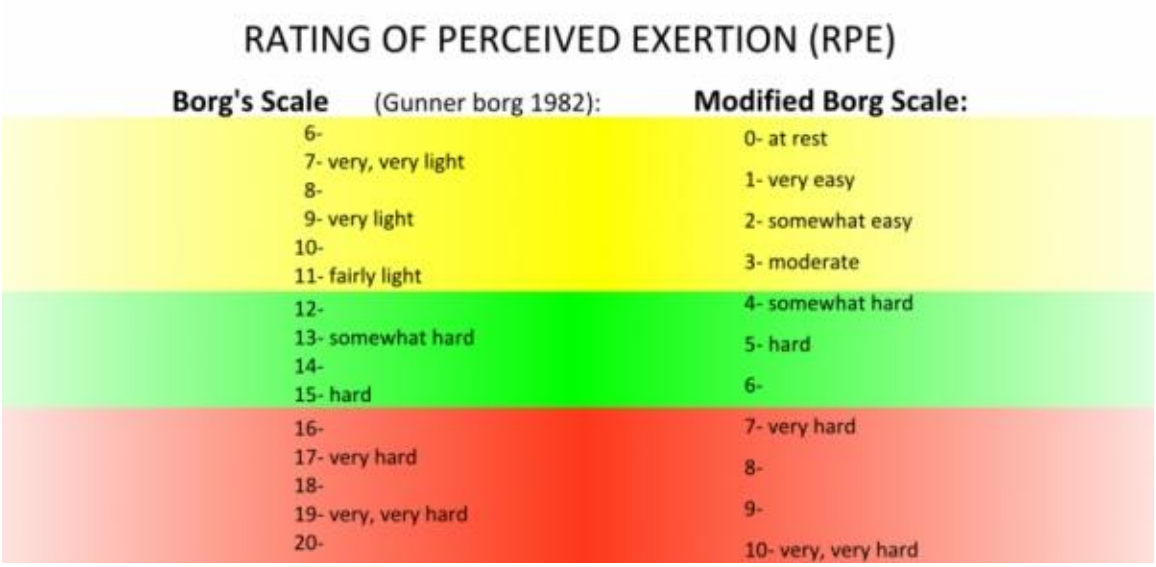

Figure 3.1 Borg RPE Scale[2]

[1] https://en.wikipedia.org/wiki/Rating_of_perceived_exertion
[2] http://www.cmsfitnesscourses.co.uk/blog/128/using-the-rpe-scale

3.2.2. Wellness

Wellness data can be captured through five topics[8]:

- Fatigue
- Sleep quality
- Muscle soreness
- Stress level
- Mood

These wellness aspects are evaluated on a scale of 1-5. An example of such a wellness questionnaire is shown in figure 3.2. The data should be captured once a day and preferably in the morning, in order to get accurate results. By using wellness results in combination with other monitored athlete health topics, the coach can get a glimpse of the athlete overall state and decide what is to be done in order to improve trainings for best influencing the athletes to improve performance.

The wellness data can be studied by the medical staff, and can indicate how the athlete is affected by the training sessions. The questionnaire is a self-evaluation of an athlete physical and mental health state, from which other aspects about the athlete can be deduced.

Wellness Questionnaire

	5	4	3	2	1
Fatigue & Energy Levels	Very Fresh	Fresh	Normal	Quite Tired	Very Tired
Sleep Quality	Very Good	Good	Difficulty Falling Asleep	Restless Sleep	Very Bad Sleep
Sleep Duration	>8 Hours	7-8 Hours	6-7 Hours	5-6 Hours	<5 Hours
General Muscle Soreness	Feeling Great	Feeling Good	Normal	Some Soreness or Tightness	Very Sore
Stress & Mood Level	Very Relaxed & Positive	Relaxed & In a Good Mood	Normal	Slightly Annoyed, Snappiness at Team-Mates	Very Annoyed, Irritable, Feeling Down

Figure 3.2 Example of Wellness questionnaire[3]

[3] http://www.scienceinsoccer.com/2013/12/creating-wellness-questionnaire-for-ipad.html

3.2.3. Monitoring illness and injury

Gathering data about the potential injuries of an athlete can show how severe the injury is and alert the coach and the medical stuff into taking some measures to treat, find the injury cause and monitor the athlete closely untill complete recovery. [8]

Figure 3.3 below shows the questionnaire used by a similar system presented in the next sections pmSys, in order to capture health problems such as illness and injury for a player.

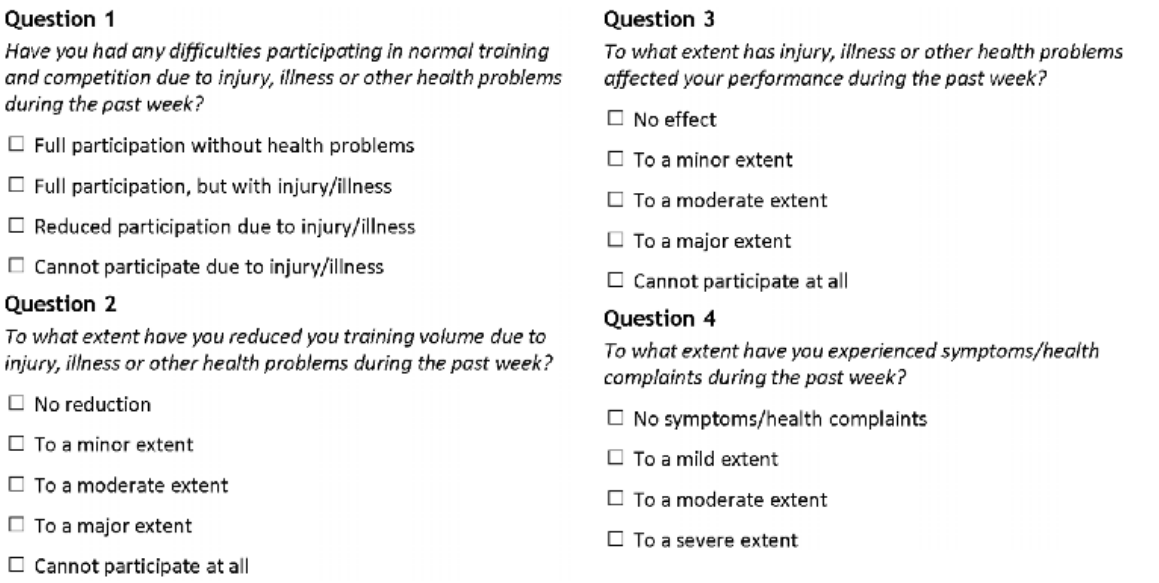

Figure 3.3 Example Injury questionnaire [8]

3.3. mHealth

Today's technology has enabled mobile services to widen the horizon of health services, resulting in a branch of eHealth, referred to as mHealth. Mobility and decreased location dependency are interconnected aspects that need to be considered for the efficiency of delivering correct health data. In the mHealth context, dependency on location inherently results in mobility limitation but the support of cloud computing and connectivity make this eHealth branch very promising and reliable due to the evolution of intelligent sensors. [9] With the ability of individuals to not only consume health services but also contribute to gathering data, mHealth extends the horizon of how to manage raw and processed data to obtain relevant results.

Users can utilize mobile devices for gathering health information and monitoring. Certain applications exist that provide guidance in health issues, diagnosis or simply monitor user's health condition. These applications gather data and then process it according to certain algorithms, being able to diagnose the user and provide him recommendations for further treatment. All mHealth applications are meant to improve health care quality and minimize error rate by using good algorithms. The access to health care is widened, people having the opportunity to get first-hand advice and feedback regarding their condition when patients would not contact their healthcare providers. This

could also decrease health care costs by lowering hospital bills as a consequence of reducing needless hospital visits. [10]

Smartphones are omnipresent and in continuous development and with them the number of mHealth users is in a continuous increase. The availability on Android and iOS platforms makes mHealth applications more accessible to any type of user, independent of the smartphone he is using. The motivation of the mHealth field came from the rapid evolution of mobile phones in developing countries. Even rural territories of a country gained greater access to mobile phones, thus the potential of making transaction and information costs smaller in order to offer qualitative health services increased.

3.4. Similar Systems

Given the fact that mobile technology has developed considerably and the eHealth field constitutes a great interest point nowadays, different systems have been created recently in order to solve the problem of health monitoring.

3.4.1. Ohmage

Ohmage is an open mobile data collection platform initiated by Deborah Estrin at Cornell University. It offers support for recording, storing, analyzing and visualizing data.[11] This data collection system enables users to register data themselves using a mobile application through self-reporting activities or the application can collect data automatically in continuous data streams. [12] Passive data, or data collected automatically and continuously without requiring user interaction is registered through the use of mobile sensors, while self-reported data consists of user observations or experiences in the form of questionnaires. Data is collected with mobile devices anytime and anywhere independent of the network connectivity, therefore the system can work offline.

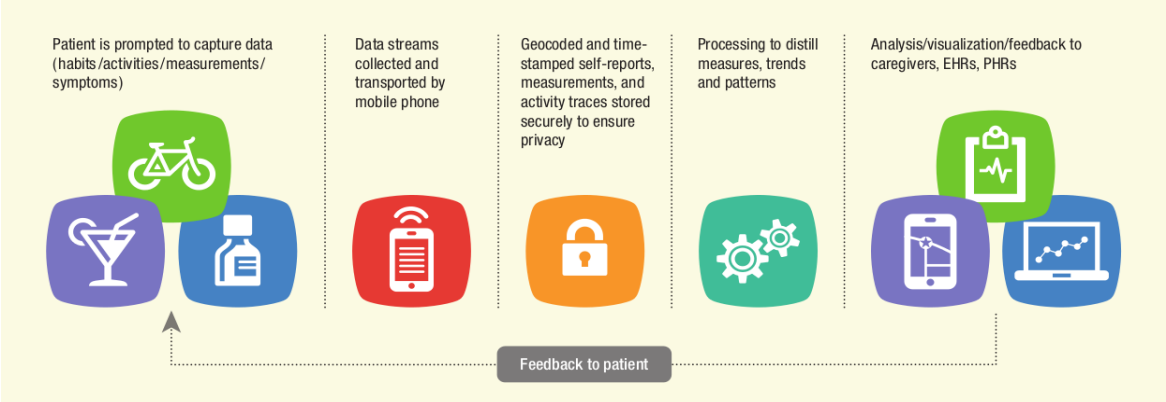

Figure 3.4 Ohmage software features[4]

Figure 3.4 shows the Ohmage main software features, displaying the flow of the data from collection to analysis and offering feedback to the user.

Four main components form the Ohmage architecture [11]:

[4] http://www.stat.ucla.edu/~jeroen/med2/#/step-3

- Ohmage backend – acts as a data store and offers interface for accessing data; also represents the central component of the platform
- Mobile applications – collect data from the users and give feedback
- Web applications – handle the data analysis and visualization

Ohmage provides an architecture rich in features and applications that can be visualized in Figure 3.5. Other custom systems and applications can be developed independently, using the Ohmage backend.

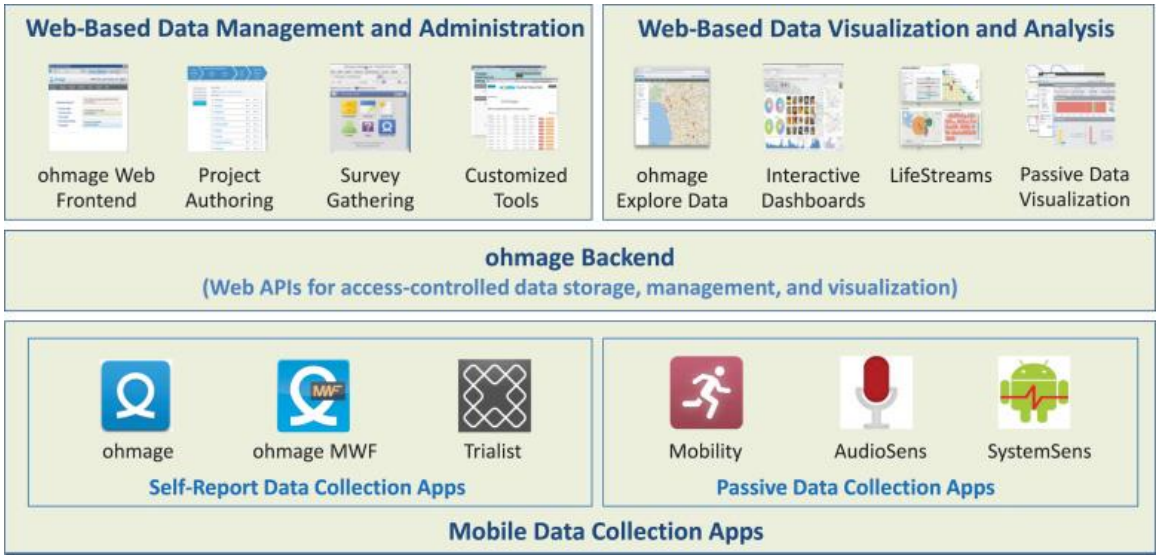

Figure 3.5 Ohmage system architecture [11]

3.4.2. Player Monitoring System

Player Monitoring System also known as **pmSys** is a software system especially designed for monitoring football players. The system was deployed for the Norwegian National Team and proved to represent an effective way of athlete monitoring, facilitating the need of reporting by paper. This system offers users data collection, processing and visualization in order to make players' and coach's lives easier when having to report on trainings. With the aid of a mobile application the users can self-report at any desired moment of time from anywhere and as many times as they need. The registered data is submitted to a server where it is stored and processed. The system organizes data automatically so the coach only needs to access the web portal in order to obtain the visual results of the player he wants to know about. The system makes it easier for a team to monitor wellness, injury and load.

The system consists of 3 main component applications [8]:

- Mobile application – enables self-reporting
- Web portal – designed for data monitoring and analysis
- Backend server – handles and processes reported data

The pmSys system uses the Ohmage backend server because of its functionality of handling reported surveys and presenting data. Disadvantages were found for using this backend server as it provides not only unnecessary data but also before data could be

presented and visualized, the application needed to do many unnecessary server requests to obtain enough data.[12]

3.4.3. Athlete monitoring

Athlete monitoring is another system that provides similar functionalities with those we want to accomplish through the system proposed in this thesis. Athlete monitoring is a "complete data management platform to monitor wellness, workloads, risk, fitness, performance, health & injuries and automatically interpret data using evidence-based methods and algorithms".[5] This system does not offer any open API thus it can hardly be extended.

3.4.4. Comparison between Ohmage and Player Monitoring System

PmSys (Player Monitoring System) is composed of 3 modules: a web platform, a mobile application and a notification system and is built using the Ohmage backend. So all users, campaigns (groups), classes (roles), surveys are stored there and cannot be manipulated by the pmSys application. A comparison the web applications' features is shown in Table 3.1 below.

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

Table 3.1 Comparison between Ohmage and pmSys we application features

So this management part can be accomplished only by Ohmage, but the Ohmage web portal was created for both coaches and system administrators, while pmSys focuses on the coaches to be able to perform analytical observations from the results display. This differentiates the systems that wanted different things to accomplish.

Referring to the visualization module, pmSys has a superior visualization of data, enabling the users to interact with it, providing content analysis of the survey answers, while Ohmage only provides statistical analysis. PmSys offers team data visualization while Ohmage doesn't have this feature at all. So definitely pmSys provides more useful feedback for monitoring and analyzing reported data and more specific charts than Ohmage.

PmSys mobile application features:

- Presentation of questionnaires
- Optimized and fast reporting
- Supports local notifications
- Feedback in form of visualizations
- Offline reporting
- Cross-platform support

[5] http://www.athletemonitoring.com/features/

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

Table 3.2 Comparison between Ohmage and pmSys mobile application features

The primary goal of the pmSys mobile application was to improve the existing Ohmage mobile application, emphasizing the optimization of the most important features such as visualization and self-reporting.

In conclusion, pmSys system provides a set of rich features that encompasses almost all features provided by Ohmage. PmSys not only offers more features but also includes the improved version of most of Ohmage features. It is to mention that this happens because of the fact that pmSys uses Ohmage backend to create an improved system that better meets the client's requirements.

Chapter 5. Analysis and Theoretical Foundation

5.1. Conceptual architecture

The project analysis phase identified three important components, as can be seen in Figure 4.1, which presents the conceptual architecture of the entire system. The system is composed of several subsystems that communicate between them and offer users data management consisting of data collection, processing, analysis, visualization, along with message notifications. Each of these three components is crucial to the system existence as each provides specific functionalities for the targeted users.

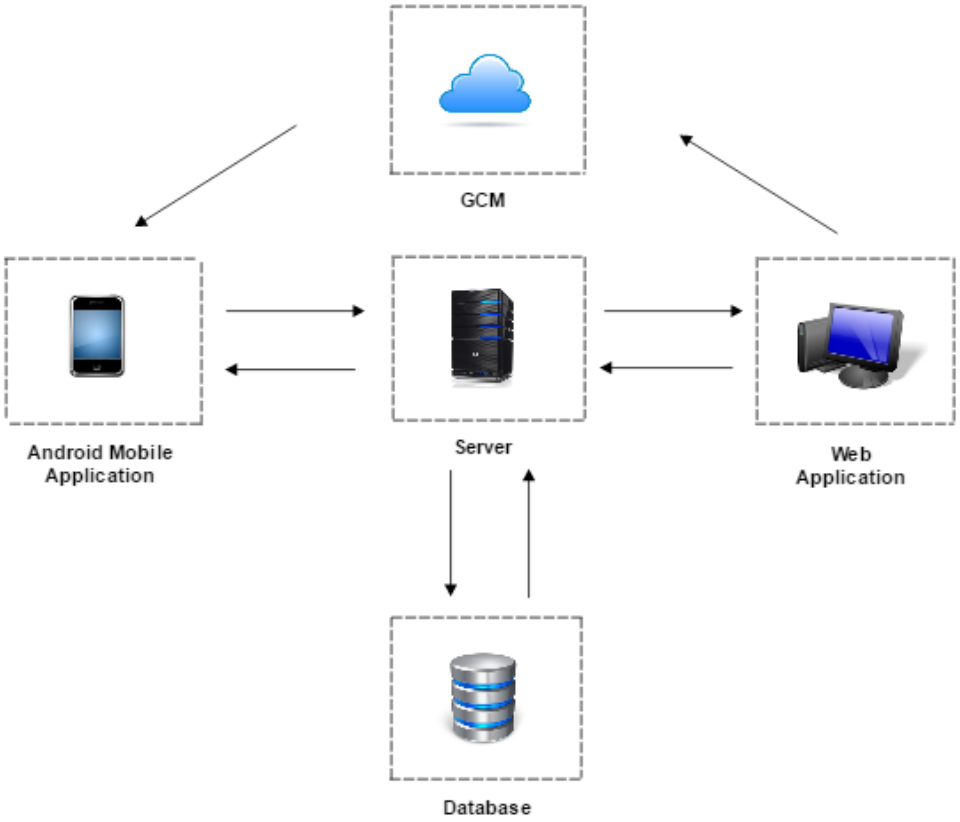

Figure 5.1 Conceptual architecture of the system

The first component is a web application, designed for the coaches and medical staff to analyze and interact with all the data the athlete has registered. In order for the coach to be able to monitor an athlete's progress, the web application provides team and single player visualization. In addition, this component also gives the user the ability of sending push notifications to players of his teams.

The second component is an Android mobile application. This application is used by the athletes in order to register survey responses. It is designed to be as easy to use as possible and enables push notifications sent from coaches by connecting to the web application. Application provides feedback after survey submission and athlete can visualize own subjective data.

The last component is the database where all the data needed by the system is stored securely.

5.2. System requirements

A software solution is required in order to easily monitor athletes' health state. The system should offer the coach all the necessary functionalities needed to successfully monitor the players from his team. The admin should be able to create users and teams and assign coaches to teams. Coaches should be able to create surveys and make them available to their team. Based on the type of user there are two applications provided: a web application for the coaches and the medical staff and a mobile application for athletes. The system should provide a way to capture and submit data through self-reporting, as well as process and analyze it. Then the system should present the collected data through visual and user-friendly graphical charts.

5.2.1. Functional requirements

The functional requirements of this system are as follows:

User authentication

The system can be used only by authenticated users. The authentication mechanism requires users to provide a username and password combination, which are stored in the database. A user that is not registered in the system has no access to any of the functionalities the system offers. For both the web application and the mobile application users are first asked for their credentials.

User management

Authenticated administrators should be able to create users by specifying some minimal details about them. They could also perform edit and delete operations on users.

Team management

Authenticated administrators should be able to create teams by specifying a team name. A team cannot be created if the admin doesn't assign a coach to it. Also admins can perform edit and delete operations on teams.

Survey management

Authenticated coaches should be able to create surveys. After a survey is created, it can be assigned to a team so that players from that team can have access to it. A survey is created with state `Disabled` by default. The coach can make the survey accessible at any time through the `Enable` operation.

Self-reporting

Self –reported data will be collected with surveys using the mobile phone. Thus, the mobile application will have to provide a way to present and submit finished responses.

Data processing

Upon survey completion and response submission through the mobile phone application, the system should be able to store, process and analyze the received data. In addition, the system must also be able to present data in order to be evaluated by the coach and the medical staff.

Visualization and presentation

The system should provide a way to present the data, after the collection and processing. Reported data will be presented in visual charts in order to be easier to analyze and evaluate. The mobile application will also provide feedback based on reported data in order to increase the motivation factor.

Push notifications

The system will offer the ability of unidirectional communication through notifications. The coach can send notification messages only to the players of his team in order to communicate important details, make announcements or send reminders to the players.

5.2.2. Nonfunctional requirements

The nonfunctional requirements of the system are as follows:

Availability

The system should be available all the time to the users connected to the internet or over the mobile phone. Server and users' location should not be considered an issue for providing user access and data presentation.

Security

The system must prevent unauthorized users from having access to the data stored in the system. Due to the fact that the system collects personal information from the users, the system needs to be secure enough to protect the data from being accessed by unauthorized users.

Scalability

The system must be able to scale with its growth due to the fact that it must be able to store large amount of data such as responses, surveys, user information. The system has to work fine as the number of users increases and thus multiple concurrent requests are performed.

Usability

Most of the users who are athletes might not have technical background, so the system must provide high usability through user-friendly design and simple user interactions. All features and user design must be simple to understand and must not require any guidance to use.

Performance

The reporting process must be fast and effective in order for the user to report and use the system more frequently. Data should be stored and retrieved instantly so the system in general should be fast and reliable.

5.3. Use case specification

Use cases have the purpose of providing a global perspective upon the functionalities and behavior of the system. They are used for the system requirements identification, clarification and organization.

5.3.1. Actors

After a thorough analysis of the system functionalities, several types of users can be identified and seen in Figure 4.2:

- Admin – handles the user and team management of the system
- Athlete – exclusive user of the mobile application
- Coach – user of the web application, monitors athletes, creates surveys
- Medical staff – user of the web application, monitors athletes

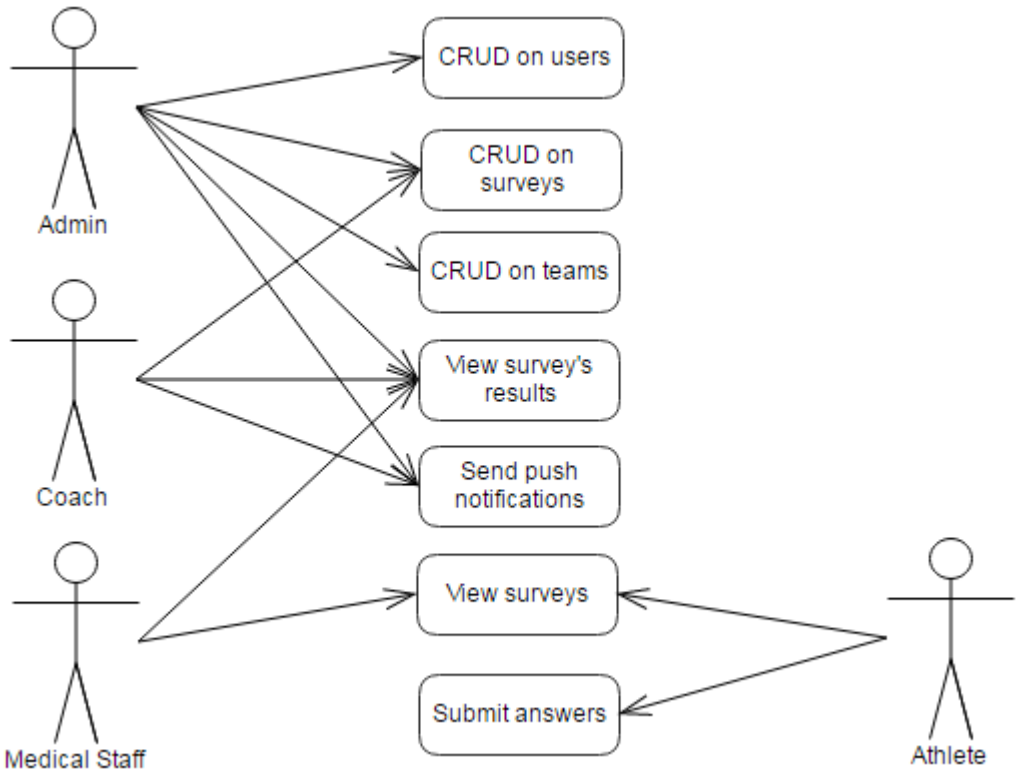

Figure 5.2 Use case diagram

5.3.2. Use cases

UC1: Login

Brief Description:

- The purpose of this use case is to capture the flow of events that an actor must follow in order to perform the login action in a custom scenario.

Primary actor: Coach

Stakeholders and Interests:

- The coach must be authenticated in order to access the system and perform operations.

Preconditions:

- The actor must have access to the login page of the system

Postconditions:

- The actor can perform the available actions which can modify the state of the system

Basic Flow:

Use-Case Start

This use case starts when the actor needs to login.

1. The actor accesses the login page.
2. The actor introduces the username and password.
3. The system verifies the validity of the credentials and allows the actor to enter the application.

Use-Case End

The athlete ends the login operation

Alternative Flows:

1. Wrong username/password combination

This flow can occur in one of the following steps: 2

1.1 The appropriate message is shown by the system.

2. The actor doesn't have an account

This flow can occur in one of the following steps: 2

2.1 The appropriate message is shown by the system.

UC2: Self-reporting

Brief description:

- The purpose of this use case is to capture the flow of events that an actor must follow in order to manage self-reporting in a custom scenario.

Primary actor: Athlete

Stakeholders and Interests:

- The athlete is interested in having a user-friendly application so he can manage to complete the required surveys at the required time and send the answers.
- The sports team manager is interested in having a user-friendly mobile application for the athlete to find it easy to locate surveys and register data. The user interface must be as simple as possible due to the fact that the athlete may not have much technical knowledge and experience with such applications. Also, response time should be short, so the user doesn't get bored in the process.

In addition, by showing some visual feedback to the athlete after submitting survey responses, increases the motivational factor to keep reporting.
- The coach is interested in having the most efficient means of monitoring so they will not have to do everything by hand.

Preconditions:
- The actor is authenticated and authorized for this use case.
- A self-reporting scenario is opened through selecting a Survey from the list.
- The survey's state must be „Active" in order for the athlete to access it. Athletes cannot access surveys in state „Disabled".

Postconditions:
- Survey should remain in a consistent state if the Send operation has ended with success.
- Survey responses are saved and submitted to server.
- Feedback is generated and displayed.
- Survey should not suffer any changes if the Send operation has not ended with success.

Basic Flow:

Use-Case Start

This use case starts when the actor needs to complete a survey.

1. The actor selects a survey and opens it for completion.
2. The system will display the survey questions and possible choices.
3. The actor selects one choice for each question.
4. The actor submits the answers

Use-Case End

The athlete ends the self-reporting operation.

Alternative Flows:

1. Abort survey completion

This flow can occur in one of the following steps: 3, 5.

1.1 The actor selects the Abort (cancel survey completion) operation and leaves the survey page.

1.2 The system remains unchanged.

2. Questions are not all completed

This flow can occur in one of the following steps: 4, 6.

2.1 Actor submits answers.

2.2 The system sends the actor back to step 3.

4. Operation failed

1.1 The operation failed from different reasons.

1.2 The system displays a user friendly message. No changes are made in the system

UC3: Create surveys

Brief description:
- The purpose of this use case is to capture the flow of events that an actor must follow in order to manage creating surveys in a custom scenario.

Primary actor: Coach

Stakeholders and Interests:

- The sports team manager is interested in enabling the coach to perform the actions that he considers are necessary in order to improve athlete performance and keep injuries far away.
- The coach is interested in having a means of creating own surveys in order to capture relevant data that interest him

Preconditions:

- The actor is authenticated and authorized for this use case.
- Access the survey creation page by clicking on "Add survey" on the survey listing page

Postconditions:

- A new survey instance is created.
- A message is displayed noticing the user that a new survey has been created.
- A message is displayed noticing the user that that the survey did not pass all validations and required fields are emphasized.

Basic Flow:

Use-Case Start

This use case starts when the actor wants to create a survey

1. The actor clicks on „Add survey" button to get to the survey creation page.
2. The system displays the survey creation form.
3. The actor completes the required fields.
4. The actor adds questions to the survey.
5. The actor submits the data.

Use-Case End

The coach ends the create survey operation.

1. Abort message send

This flow can occur in one of the following steps: 2.

1.1 The actor selects the Cancel operation.

1.2 The system remains unchanged.

2. Operation failed

2.1 The operation failed from different reasons.

2.2 The system displays a user friendly message. No changes are made in the system

The flow diagram of this use case can be seen in Figure 4.3 below.

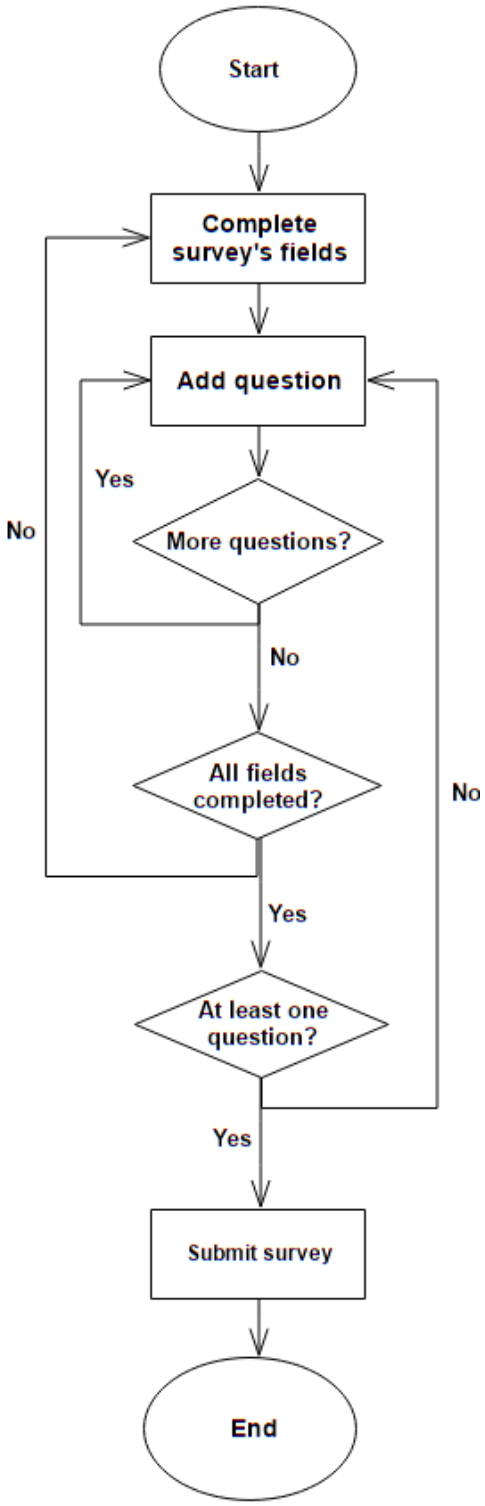

Figure 5.3 Flow diagram of creating a survey

5.4. Technological perspective

5.4.1. Ruby programming language

Ruby is a dynamic, open source programming language with a focus on simplicity and productivity[6]. It has an elegant syntax that is natural to read and easy to write.

- **Ruby is pure object-oriented:** Programming in Ruby does offer encapsulation of data, methods within objects, allows inheritance from one class to another and supports polymorphism of objects. Everything is represented as an object in Ruby, including primitive data types such as strings and integers. There is no need for wrapper classes such as Java has.
- **Ruby is multi-platform:** It runs on Linux and other UNIX variants and also the various Windows platforms.
- **Distributed Ruby:** DRb allows Ruby programs to communicate with each other on the same machine or over a network. It uses remote method invocation to pass commands and data between processes.
- **Ruby is multi-paradigm:** It allows procedural programming, with object orientation or functional programming (anonymous functions). It has support for introspection, reflection and meta-programming as well as support for interpreter-based threads.

Ruby is the language used for building the web application. One of the main reasons for using Ruby is because of the web framework built on it: **Rails** which is designed specifically for the development of web applications.

5.4.2. Ruby on Rails (RoR)

Ruby on Rails is a web application framework written in Ruby that facilitates faster building web applications. Rails was created in 2005 and has become a serious and popular alternative to traditional web development environments such as Java and .NET. Rails can be used to create professional applications, providing an open-source web framework with integrated support for integration testing, unit testing and functional testing. Rails offers support for using Ajax to make applications highly interactive, also for writing REST based interfaces so they can interact with other RESTful applications with little to no effort [13]. The Android application designed for the athletes will use exactly this benefit of the Rails framework because the application will have to communicate with the backend in order to get the required information for display.

As many other web frameworks, Rails uses the model-view-controller (MVC) architectural pattern, providing default structures for a database, a web service and web pages. It encourages and facilitates the use of web standards such as JSON or XML for data transfer and HTML, CSS and JavaScript for display and user interfacing. [14] The Rails architecture diagram can be seen in Figure 4.2 below.

[6] https://www.ruby-lang.org/en/

In addition to MVC, Rails emphasizes the use of other well-known software engineering patterns and paradigms, including convention over configuration (CoC), don't repeat yourself (DRY), and the active record pattern.

Convention over Configuration is a software design paradigm which seeks to decrease the number of decisions that developers need to make, gaining simplicity and not necessarily losing flexibility. The phrase essentially means that a developer only needs to specify the unconventional aspects of the application.[7]

Don't repeat yourself is a principle of software development, aimed at reducing repetition of information of all kinds, especially useful in multi-tier architectures. Every piece must have a single unambiguous, authoritative representation within a system. When the DRY principle is applied successfully, a modification of any single element of a system does not require a change in other logically unrelated elements.[8]

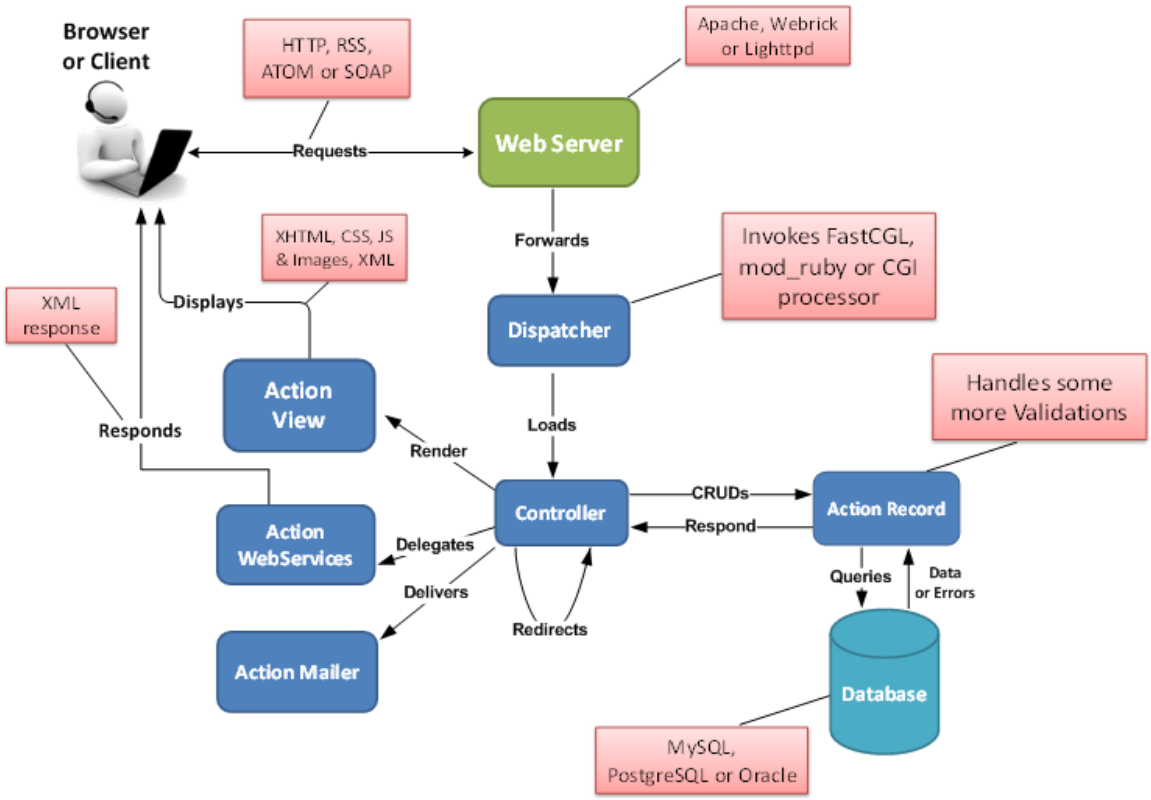

Figure 5.4 Generic Rails architecture diagram[9]

WEBrick is the default Ruby on Rails server. WEBrick is a HTTP server toolkit that can be configured as an HTTPS server, a proxy server and a virtual host server. [10]

Rails is separated into various packages, namely ActiveRecord, ActiveResource, ActionPack, ActiveSupport and ActionMailer. The **ActiveRecord** library implements

[7] https://en.wikipedia.org/wiki/Convention_over_configuration
[8] https://en.wikipedia.org/wiki/Don%27t_repeat_yourself
[9] http://vertisinfotech.com/ror-expertise
[10] http://ruby-doc.org/stdlib-2.0.0/libdoc/webrick/rdoc/WEBrick.html

ORM. It creates a persisting domain model from business objects and database tables, where logic and data are presented as a unified package. ActiveRecord adds inheritance and associations and integrates the Single Table Inheritance to the pattern so it increases the functionality of the active record pattern approach.

5.4.3. PostgreSQL

PostgreSQL is an object-relational database management system with an emphasis on extensibility and standards-compliance[11]. PostgreSQL is a database server with the main goal of secure data store, supporting best practices, allowing data retrieval at the request of other software applications. It can handle workloads ranging from small single-machine applications to large Internet-facing applications with many concurrent users.

In article [15] the following aspects about PostgreSQL are presented:

- Compared to other RDBMSs, PostgreSQL differs itself with its support for highly required and integral object-oriented and relational database functionality, such as the complete support for reliable transactions.
- Due to the powerful underlying technology, Postgres is extremely capable of handling many tasks very efficiently. Support for concurrency is achieved without read locks thanks to the implementation of Multi-version Concurrency Control (MVCC), which also ensures the ACID (atomicity, consistency, isolation, durability) compliance.
- PostgreSQL is highly programmable and therefore extendible, with custom procedures that are called 'stored procedures'. These functions can be created to simplify the execution of repeated, complex and often required database operations.

5.4.4. Android

Android is a mobile operating system currently developed by Google, based on the Linux kernel and designed primarily for touchscreen mobile devices such as smartphones and tablets. Android's user interface is mainly based on direct manipulation, using touch gestures that loosely correspond to real-world actions. [16]

Since Android devices are usually battery powered, Android is designed to manage processes to keep power consumption at a minimum. Android manages the applications stored in memory automatically: when memory is low, the system will begin invisibly and automatically closing inactive processes, starting with those that have been inactive for longest.

Each Android application run its own Linux process. A new process starts when any of the application's code needs to be executed. Each process has its own virtual machine (VM), so each application code runs independently of the other applications.

[11] https://en.wikipedia.org/wiki/PostgreSQL

Figure 4.3 shows major components of the Android OS. On top of the Linux kernel, there are the middleware, APIs written in C, libraries and application software running on an application framework which includes Java-compatible libraries.

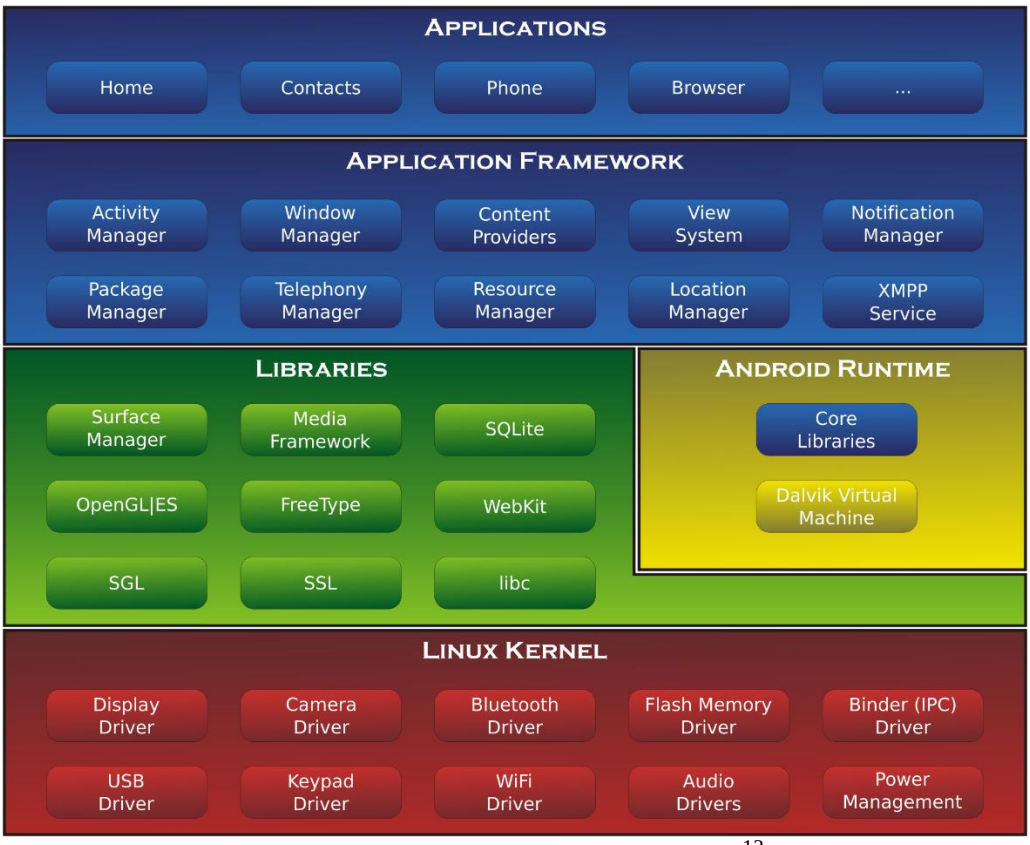

Figure 5.5 Android architecture[12]

The mobile application for the athletes is implemented in Android SDK. Android SDK is a software development kit that enables developers to create applications for the Android platform.

5.4.5. Bootstrap

Bootstrap is a very popular framework for creating web applications. It is a HTML, CSS, JavaScript framework used for web pages design, providing only front-end development support. The main reason for using bootstrap in the Ruby on Rails web application is because of the fact that it is very simple to use since the framework has consistent documentation and support. Bootstrap is compatible with the latest browser versions and supports responsive design, meaning that the pages adjust relative to the display size of the device used.

[12] https://en.wikipedia.org/wiki/Android_(operating_system)

Bootswatch is a collection of open source themes for Bootstrap[13]. For the web application we applied the Bootswatch Readable theme, which gives a plain and simple look to the application, keeping its elements well defined, visible and in harmony. [17]

5.4.6. HAML

Haml is a templating language with an attractive syntax that enables the user to write code faster and easier. Haml (HTML Abstraction Markup Language) finally gets transformed into HTML and makes writing it much easier. Haml is designed to strip as much of the repetitive HTML elements as possible, having no closing tags. Indentation stays at the core of its structure, providing a direct visualization of the DOM hierarchy. Haml parser throws explicit exceptions if indentation is not done appropriately so not closing a tag will never be a reason to worry. Haml is mostly used in Ruby on Rails applications acting as a replacement for inline page templating systems such as ERB. [18]

ERB (Embedded Ruby) is a templating system that embeds Ruby into a text document.[14] This feature combines plain text with Ruby code and the Ruby standard library contains it. The main reason for choosing HAML over ERB is because HAML encourages well-indented code and reflects the underlying structure of the document, making the code hierarchy visible and easier to read. Also, it provides a faster approach to writing HTML which constitutes a benefit for the web application development.

5.4.7. Devise

Devise is a flexible authentication solution for Rails applications. It is composed of 10 modules: [19]

- **Database Authenticable**: hashes and stores a password in the database to validate the authenticity of a user while signing in. The authentication can be done both through POST requests or HTTP Basic Authentication.
- **Confirmable**: sends emails with confirmation instructions and verifies whether an account is already confirmed during sign in
- **Recoverable**: resets the user password and sends reset instructions
- **Registerable**: handles signing up users through a registration process, also allowing them to edit and destroy their account.
- **Rememberable**: manages generating and clearing a token for remembering the user from a saved cookie
- **Trackable**: tracks sign in count, timestamps and IP address
- **Timeoutable**: expires sessions that have not been active in a specified period of time
- **Validatable**: provides validations of email and password. It's optional and can be customized to include own validations
- **Lockable**: locks an account after a specified number of failed sign-in attempts. Can unlock via email or after a specified time period

[13] https://github.com/thomaspark/bootswatch
[14] https://en.wikipedia.org/wiki/ERuby

- **OmniAuthable**: adds OmniAuth support

Devise is used in the web application in order to perform the authentication and the used modules are database authenticable and recoverable. This gem is used in order to make authentication easier and faster and because it has a wide variety of features available for usage, as can be seen above.

Many other gems that concern the same subject exist such as: Authlogic[15], Sorcery[16], Clearance[17], and others but they have a more limited approach.

5.4.8. Reform

Reform provides form objects that maintain validations for one or multiple models, where a model can be any kind of Ruby object [20]. Reform is used in the web application in order to act as an intermediate validation layer before writing data to the persistence layer. While form objects may be used to render graphical web forms, Reform is used in many pure-API application for deserialization and validation.

5.4.9. Rpush

Rpush is a gem for sending push notification in Ruby. The main benefits of using rpush are ease of use, reliability and a rich feature set. [21]

Rpush has a set of supported services:

- Apple Push Notification Services
- Google Cloud Messaging
- Amazon Device Messaging
- Windows Phone Push Notification Service

Rpush gem is used for sending push notifications to the Android device from the Rails web application, using GCM. Another alternative for using rpush is the **gcm[18]** gem. Gcm gem enables the ruby backend to send notifications to Android and iOS devices via Google Cloud Messaging. The reason of using Rpush over Gcm is that Rpush provides numerous advanced features not found in other gems, giving the developer greater control and insight as the developed project grows. Rpush is run as a daemon or inside an existing process and the user can choose the number of persistent connections for each application.

5.4.10. Chartkick

Chartkick is a charting library for Ruby on Rails that permits the creation of nice looking charts for web applications. [22] Chartkick is compatible with major browsers and can easily be used to give the application some visual effects that enhance the user's comprehension of the underlying information. Charts offer a way of presenting data in such a manner to make the user better visualize it, without the need for further analysis.

[15] https://github.com/binarylogic/authlogic
[16] https://github.com/NoamB/sorcery
[17] https://github.com/thoughtbot/clearance
[18] https://github.com/spacialdb/gcm

In the web application charts are used by the coaches and medical staff in order to visualize the athletes' responses to the carefully selected questions. Charts must be displayed as clear as possible in order for the coach to be able to interpret it as fast as possible. Chartkick provides beautiful and interactive charts with minimal effort in Rails.

An alternative to chartkick would be using lazy_high_charts[19] gem but this gem requires more effort to write the code and also support only Highcharts from ruby code while chartkick supports Chart.js[20] (HTML5 based Javascript charts), Google Charts[21] and Highcharts[22] (interactive javascript charts).

5.4.11. Volley

Volley is a HTTP library that makes networking for Android applications easier and faster [23]. Using Volley has a set of advantages:

- Automatic scheduling of network requests
- Multiple concurrent connections
- Support for request prioritization
- Ease of customization

Volley is used in the Android application in order to make possible sending requests to the Rails backend server in order to get the necessary data needed to be displayed. Volley library masters at RPC-type operations and it easily integrates with any protocol and provides support for raw strings, images and JSON. Volley is not suitable for large streaming operations since it holds all responses in memory during parsing.

One of the main reasons for choosing to use Volley is because it supports JSON objects, this being the format in which data is provided from the backend server. In addition it is simple to use and perfect for handling small requests like the ones we need for the application.

5.4.12. jQuery

jQuery is a fast, small and rich feature JavaScript library. It provides HTML document traversal and manipulation, event handling and animation.

Using jQuery has a series of benefits [24]:

- **Encourages separation of JavaScript and HTML**: jQuery offers a simple syntax for adding event handlers to the DOM using JavaScript, rather than adding HTML event attributes to call JavaScript functions
- **Eliminates cross-browser incompatibilities:** jQuery handles browser inconsistencies and offers an interface that functions across several browsers.
- **Is extensible:** new elements, functions can be added easily and then reused as a plugin

[19] https://github.com/michelson/lazy_high_charts
[20] http://www.chartjs.org/
[21] https://developers.google.com/chart/
[22] http://www.highcharts.com/

The main reason for using jQuery in the web application is for being able to create a dynamic behavior whenever needed like effect and animation or events. JQuery also enables DOM manipulation based on CSS selectors that uses element names and attributes as criteria to select nodes in the DOM and multi-browser support.

5.4.13. GCM

Google Cloud Messaging (GCM) is a Google mobile service that provides support for third-party systems to send messages to devices running on Android Operating System.

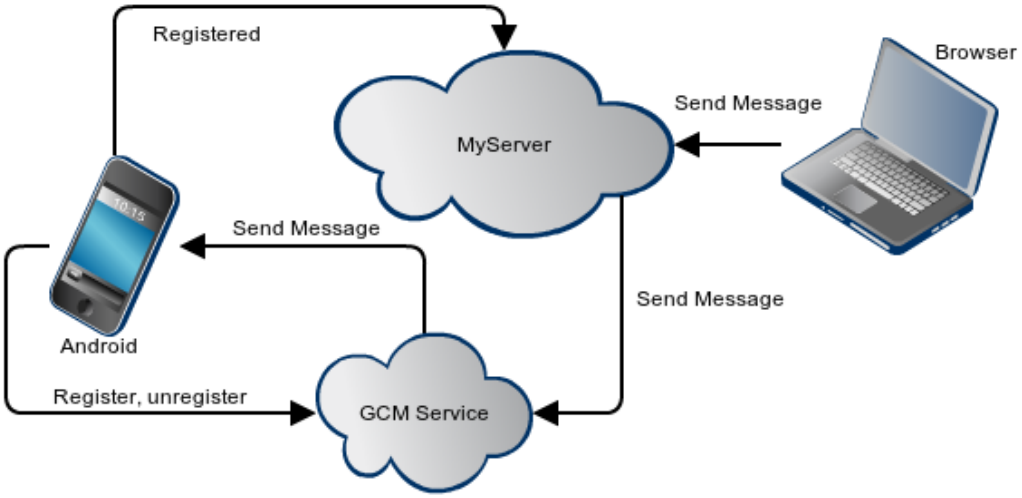

Figure 5.6 Google Cloud Messaging[23]

Figure 4.4, which can be seen above presents the necessary interactions that need to be done in order to send messages from a server to an Android device. First the device needs to register to GCM and give the server the registration id provided. After that the server can send messages to GCM that will be delivered to the device.

In our system we want to use GCM in order to enable the coach to send push notifications to an individual athlete or to a group of athletes. This is useful in order to remind the athletes to complete a survey at a given time and to notify them of training schedule changes or other messages the coach might want to send.

[23] http://phoneia.com/google-cloud-messaging-open-to-developers/

Chapter 6. Detailed Design and Implementation

This chapter will present the detailed design and implementation of the two components of the system: the mobile component and the web component. The diagrams of the two architectures will be described and each module will be detailed. In addition, other presented concepts are: database design, deployment, scrum methodology, continuous integration and cloud deployment.

6.1. Implementation of the web application

6.1.1. Architecture

The web component is a Ruby on Rails application, designed for the coaches and medical staff to monitor the athlete's responses to different issues considered of importance in order to improve performance and not threaten the athlete's health state. The architecture of the web application is based on the MVC[24] pattern due to the nature of the Rails framework. The application's MVC modules interaction can be seen in Figure 5.1. The client interacts through the views with the controller module that handles each user interaction. The controller applies the logic corresponding to the requested action and manipulates the model accordingly. Then it displays a message to notice or alert the user of the state of the request (success or failure), through a flash and shows the modified model in the corresponding view, giving the user the output of his actions.

The MVC architectural pattern divides the work that has to be done into three separate but highly cooperative subsystems: **Models**, **Views and Controllers**. The Model deals with the business logic, the View handles the display logic, while the controller centralizes the application flow. MVC permits a clean separation of the web application concerns, and keeps each part of the application in its corresponding module for easier maintenance.

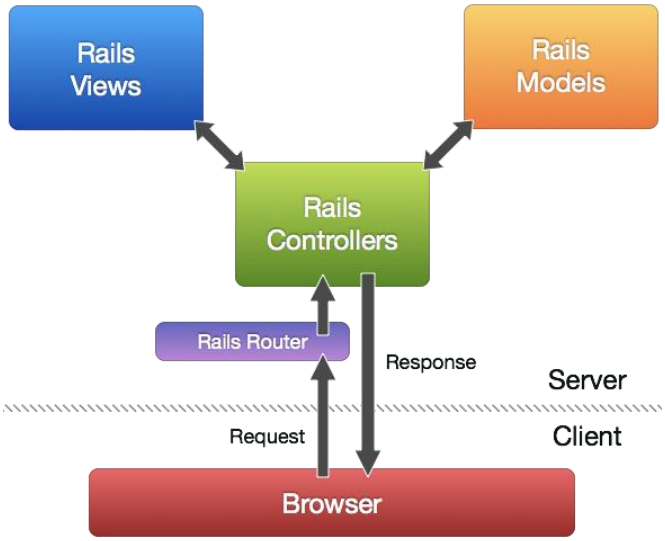

Figure 6.1 Rails MVC application structure[25]

[24] http://www.tutorialspoint.com/ruby-on-rails/rails-framework.htm
[25] http://blog.ifuturz.com/ruby-on-rails/ruby-on-rails-mvc-learn-with-fun.html

The web application architecture can be seen in Figure 5.2 and contains the following components: Models, Views, Controllers, Helpers, Forms, Services, Mailers and Assets.

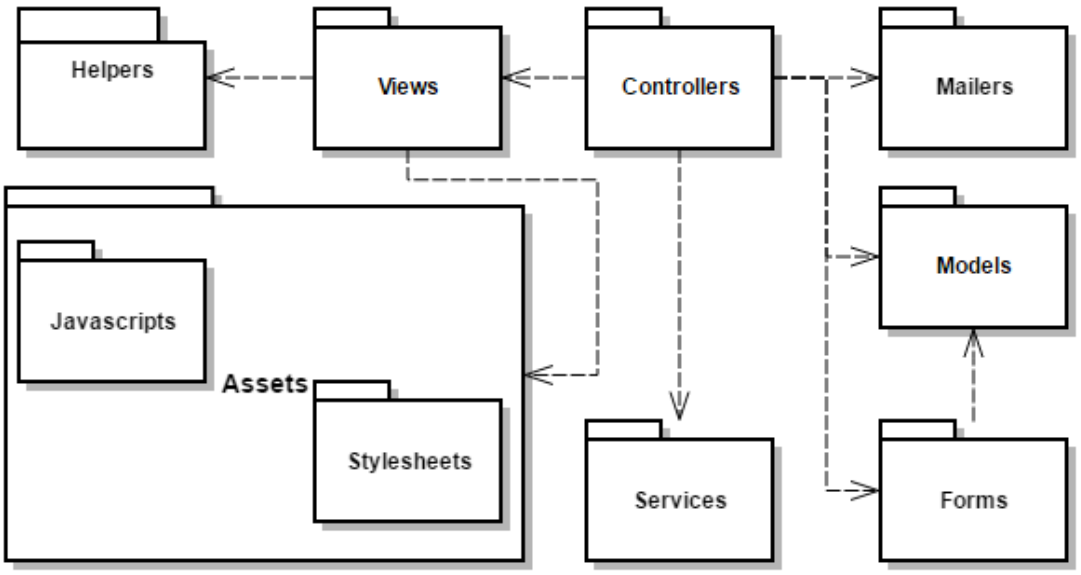

Figure 6.2 Web application block diagram

The **Models** module contains Ruby classes that communicate with the database, store and validate data, performing the business logic of the system. The structure of the models module can be seen in Figure 5.3.

The models contain associations between them. Associations are a set of class methods for tying objects together through foreign keys. They express relationships between entities and there are four ways of associating models: has_one, has_many, belongs_to and has_and_belongs_to_many. All models in the web application have associations between them. For example the survey and team models have a has_and_belongs_to_many association specifying a many-to-many relationship. This associates the classes via an intermediate join table that doesn't have an associated model.

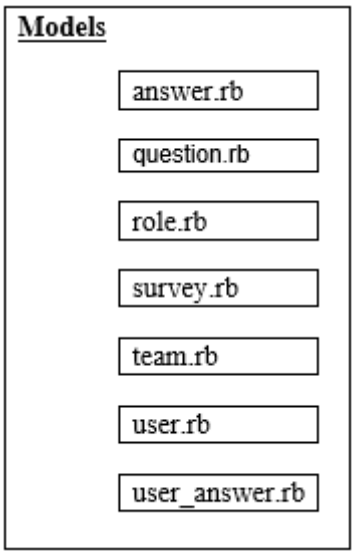

Figure 6.3 Models

Models also contain callbacks which are methods that get called at certain moments of an object's life cycle. In this way it is possible to write code that will run when an object gets created, saved, updated, validated, deleted, or loaded from the database. Such a method is implemented on the user model in order to generate an authentication token before creating a user that will be used for the authentication on the Android application.

The **Views** module represents the user interface of the application, consisting of HTML HAML files with Ruby code. The embedded Ruby code is fairly simple and only used to display the requested data to the user. To be able to present data in HAML format, the haml gem was added to the project. HAML provides a simplified syntax of the HTML code, which makes views faster and easier to write. The views are organized based on the entity that they display. Figure 5.4 shows the Views module.

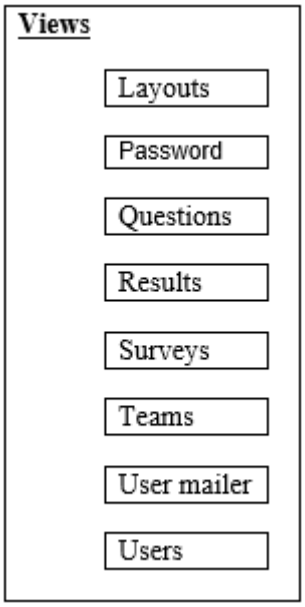

Figure 6.4 Views

The layouts folder contains the views of the application that are used on multiple pages. It includes the user layout consisting of the navigation and flash messages display, and the application view that contains the layout of the application when the user is not logged in.

Partial templates are used for breaking the rendering process into more manageable parts. Code for rendering a particular piece of the view can be put in its own file and may be used in multiple views. This simplifies views and makes them reusable. Views for questions, surveys, teams and users have a partial named *_form.html.haml* that is rendered in the new and edit pages because they both need the same form. However, the action on the form is distinct so it is passed as a parameter for each view.

The **Forms** module holds the form classes that act as an intermediate validation before data is written to the persistent layer. The Reform[26] gem is added to the project and is used to create these form objects that offer validations and nested setup of models. Forms are defined in separate classes and these classes partially map to a model. Figure 5.5 shows the Forms module content.

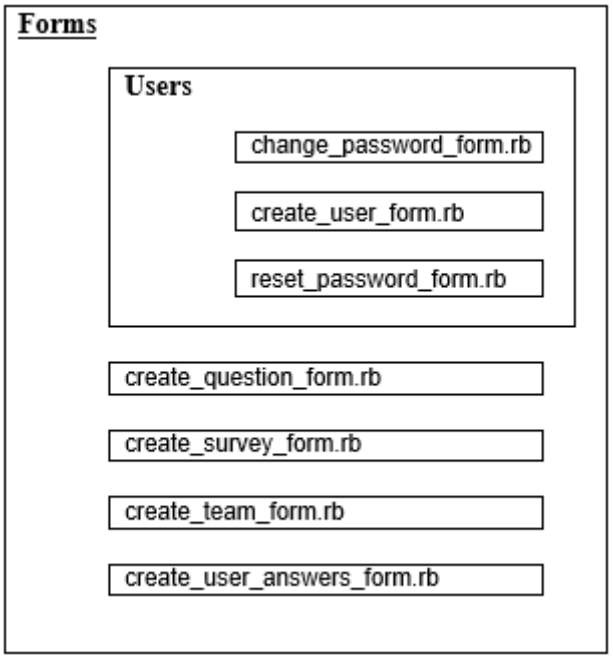

Figure 6.5 Forms

All these classes inherit from Reform::Form and contain the mapped model attributes declared and the desired validations. Form classes are instantiated in the controllers where the desired models are passed in. After verifying if all validations pass, the form object is saved and therefore so is the model object within it. In this manner, we do not use validations directly on the model, enabling any type of user manipulation from the rails console.

[26] https://github.com/apotonick/reform

The **Controllers** module is the logical center of the application, interacting with the views and the models and handling incoming requests from the browser. The content of the controllers' module can be seen in figure 5.6.

All controllers inherit from ApplicationController, which contains the user authentication filter and defines the layout of the application conditioned by the user being authenticated or not.

Each action in each controller has a route defined in the routes file that sets the type of the request and the path. Browsers request pages from Rails by making a request for a URL using a specific HTTP method such as GET, POST, PATCH, PUT and DELETE. Each method is a request to perform a specific action on a resource. A resource route maps a number of requests to actions in a single controller, so just by using ***resources :users*** in the routes file, seven different routes are created in the application, all mapping to the Users controller. These routes correspond to the index (listing), new, create, edit, update, show and destroy actions. The root route is mapped to the HomeController index method that represents the application homepage.

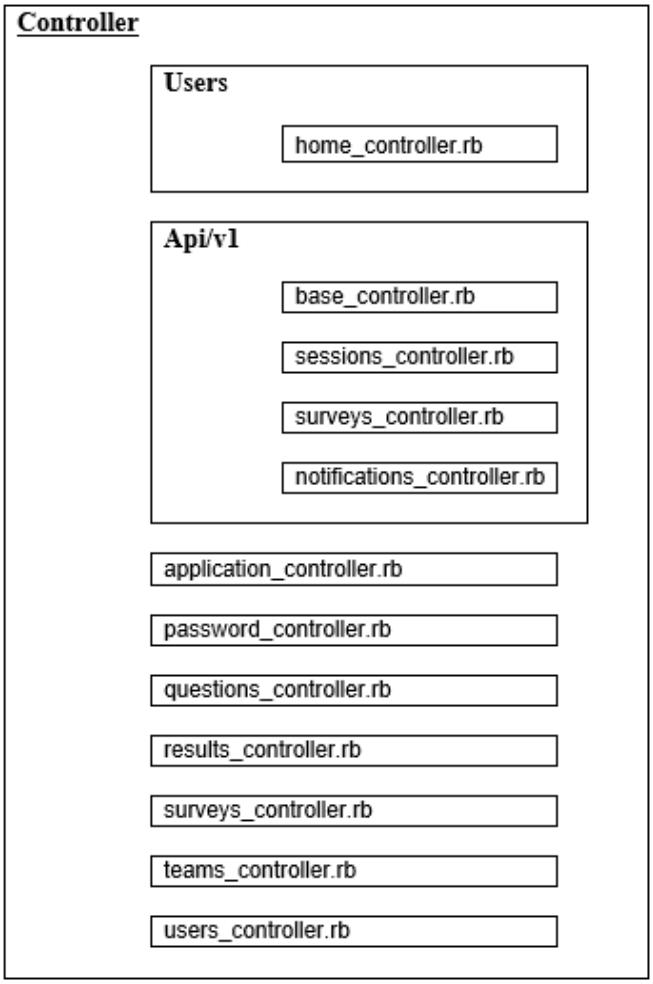

Figure 6.6 Controllers

Helpers are used in order to extract complicated logic from the view and in order to avoid code repetition. In the web application we defined a helper that displays a navigation item and is used for every item in the navigation bar.

The **Services** module contains three services: the user creation, the reset password and the change password. These functionalities were decoupled from the controllers because they were more complex and required various methods in order to accomplish their task.

The **Assets** module is called the Rails assets pipeline and is no longer a core feature of Rails 4 as it has been extracted into the sprockets-rails[27] gem, out of the framework. The pipeline concatenates assets which can diminish the number of requests that a browser makes to render a page. All JavaScript files are concatenated in a main .js file and all CSS files are concatenated in a main .css file. In the Assets module we have the stylesheets, javascripts and images.

[27] http://guides.rubyonrails.org/asset_pipeline.html

The stylesheets of the project are the Bootwatch Readable theme combined with elements of the Bootstrap Cardio theme. The javascripts folder contains the javascript code for rendering the dynamic addition of fields and table columns needed in the application. When a user wants to assign multiple answers to a survey question, the view displays a table with two columns containing the already existent answers to that question. On the page there is a button that dynamically adds a new row to the table with two text fields if the user wants to add a new answer. Similarly when the user wants to add a new member in a team, a select box needs to appear dynamically so that the user can select the member to be added from the already existent users in the system. The buttons that make the additions have attached an onClick() javascript function that gets called in order to insert the fields. The functions use jQuery to append a new element to the table having the specified id or to the div having a given class.

The **Mailers** module contains a user_mailer class that is used to send the email with the generated password to the user, after it has been created or after the user forgot his password and chose to reset it. Mailers allow sending emails from the application using the mailer classes and views. Each of the user_mailer methods has an associated view in which the html of the mail and content are defined.

6.1.2. Authentication

The web application uses the Devise[28] gem, which is a popular authentication solution for Rails applications that takes care of many different aspects for the developer. In order to get Devise up and running, a series of simple commands must be issued and it can be customized easily. In the web application we use Devise to handle login and logout, but also the reset password and change password mechanisms. This is implemented by adding in the User model the devise database_authenticable and recoverable modules.

Devise also generates routes for the login and logout pages, so in the routes file a single line is added: *devise_for :users*. By adding `before_action :authenticate_user!` in the ApplicationController we force the user to redirect to the login page if the user was not authenticated. Devise also provides a series of useful helpers that we use throughout the application. The ***user_signed_in?*** method verifies if a user is signed in and the ***current_user*** method gets the current signed in user. Both can be used in every controller and view in the system.

6.1.3. Implementation details

One of the important components of the web application is the Gemfile[29]. The Gemfile is a file used for describing gem dependencies for Ruby programs. A gem is a collection of Ruby code that we can extract into a collection which we can call later. The Gemfile is located in the root of the project directory where the Bundler[30] expects to find it. The bundler provides a consistent environment for Ruby projects by tracking and installing the exact gems and versions that are declared in the Gemfile.

[28] http://www.rubydoc.info/github/plataformatec/devise/#Warden
[29] http://tosbourn.com/what-is-the-gemfile/
[30] http://bundler.io/

The web application was created having in mind that it would be used by the coaches of teams, in order to monitor their athletes' health state by analyzing the responses to a series of surveys. They can create any survey and make it available to the athletes to complete. The results can be seen in visual charts per team or per athlete and in order to make this possible the chartkick[31] gem was used. Chartkick is a javascript library that works with Google Charts and that creates beautiful and interactive charts. It has many options for customization and also has many supporting libraries like groupdate[32]. Groupdate gem was also used because it provides the simplest way to group models by day, week, hour of day, month, year and works hand in hand with chartkick.

When accessing the view results page as coach, every survey available to the team is shown and each one has a page that displays a column chart for every question in the survey in order to point out how many users chose a particular answer. In addition at the top of this page a line chart is added in order to display the number of users that answered the survey in a particular date. This was done with the help of the above mentioned libraries directly in the view, as shown below:

```
%h4.margin
  = 'How many answered this survey?'
- @users = User.where(team_id: @team.id)
= line_chart UserAnswer.where(user: @users,
answer:@answers).group_by_day(:created_at).count

- @questions.each do |question|
  %h4.margin
    = question.text
  - @user_answers = UserAnswer.where(answer_id: @answers, user: @users)
  = column_chart @user_answers.joins(:answer).group('answers.text').count
```

Form objects are used using the Reform gem in order to create an intermediate validation layer of the model before saving it in the database. The form gets instantiated in the controller where a model is assigned to it. Beside an underlying model, a form contains other attributes. When creating a survey, we create a form for it and pass it a new survey instance. After it has been instantiated, we validate the form, passing it the parameters introduced by the user in the view form. This validation does the mapping of the form model with the parameters hash received. If the validations pass, the form is saved and the survey is saved along with it. We can see the create method from the SurveysController below.

```
def create
    @form = CreateSurveyForm.new(Survey.new)
    if @form.validate(survey_params)
      @form.save
      redirect_to surveys_url, flash: { notice: 'Survey successfully added' }
    else
      render 'new'
    end
  end
```

[31] http://chartkick.com/

[32] https://github.com/ankane/groupdate

But when we create the survey, we want to be able to also create questions for it and save them along with the survey association. For that we used nested models in the form. So besides declaring the survey attributes, we also declared a collection of questions associated to the survey along with the attributes that we needed to access from them. In order to do the mapping of the questions parameters sent by the user and the nested questions model we created a populator. Populating is invoked in the validate method and will add nested forms depending on the incoming hash. The populator shown in the class below creates as many questions as you want to introduce and also handles destroying them if you want to delete a question from a survey.

```
class CreateSurveyForm < Reform::Form
  property :name
  property :description
  property :status
  collection :questions, populator: :populate_questions do
    property :text
    property :_destroy, virtual: true
  end

  validates :name, :description, :status, presence: true

  def populate_questions(fragment:, **)
    item = questions.find { |question| question.id.to_s == fragment['id'].to_s &&
!question.id.nil? }
    if fragment['_destroy'] == '1'
      questions.delete(item)
      return skip!
    end
    item ? item : questions.append(Question.new)
  end
end
```

In order to add questions to a survey, the user interface was designed to provide a button that dynamically adds fields to a div, along with a delete field button for the case we decide to delete a question. In order to achieve this functionality, a jQuery function was created so that each time a button with the *add-field* class is clicked, it inserts a new text field and appends it to the div. This function is included in the `$(document).ready` so it will run only when the page DOM (Document Object Model) is ready for JavaScript code to execute. The function is shown below:

```
$(document).ready(function(){
    $('.multi-field-wrapper').each(function() {
        var $wrapper = $('.multi-fields', this);

        $(".add-field", $(this)).click(function(e) {
            var inputs = $(this).closest('div').find($("input[type='text']") );
            var counter = inputs.length;
            var field = $(this).closest('div').attr('id');
            var multifield = $(document.createElement('div')).attr('class', 'multi-
field', 'id', counter);
            if ($(this).attr("id") == 'questions')
                multifield.after().html('<input type="text" class="form-control"
id="create_survey_' + field + '_attributes_' + counter +
```

```
                    '_'+ 'text' + '" name="create_survey['+ field + '_attributes][' +
counter + '][text]" value="" style= "width: 450px"><button class="btn btn-danger btn-sm
glyphicon glyphicon-trash" type= "button" onclick="deleteField(this)"></button>');
            multifield.appendTo($wrapper);

        });
    });
})
```

6.2. Implementation of the Android application

6.2.1. Architecture

The mobile application was designed to run on Android devices and was implemented in Java programming language, using the Android SDK tools and the Android emulator. Android SDK contains all the necessary documentation, libraries and tools to build and debug an application. Android Studio is the official IDE for Android platform development and was used for writing the application, being very user-friendly. It uses gradle to automate the application build and to facilitate the addition of libraries to the project. Android Studio also provides quick fixes and Android specific refactoring which makes it a powerful tool in the development of the application.

The mobile application was developed just for the athletes, so they can manage to submit their subjective perspective on the training and their current heath state independent of the access they have to a computer. So the Android application is meant to facilitate the accessibility to the application in order to motivate the user to register responses to the available surveys at the required time.

Figure 5.7 shows the structure of the mobile application. The application is composed of Activities, Adapters, Layouts and classes that make possible the Push Notification feature. An **Activity** is an application component similar to a controller that renders a view with the same name from the Layouts folder and has methods that implement the view's functionality. The mobile application consists of multiple activities that are connected to each other. MainActivity is the activity presented to the user when first launching the application and represents the login page.

An **Adapter** object is an application component that acts as a bridge between an AdapterView and the underlying data for that view. The adapter is responsible for making a view for each data item in the array given. We have two adapters, one for creating the survey list, and one for creating the questions list along with the answers.

The **Layouts** folder contains all the xml views from the application. A layout represents the visual structure for a user interface. Layouts are usually associated with activities and when a new activity is created, a new layout file with the same name is generated and displayed on create. The mobile application has a layout for each activity, layouts for two different list items and a footer layout where a button is placed in order to be displayed at the bottom of the survey question list.

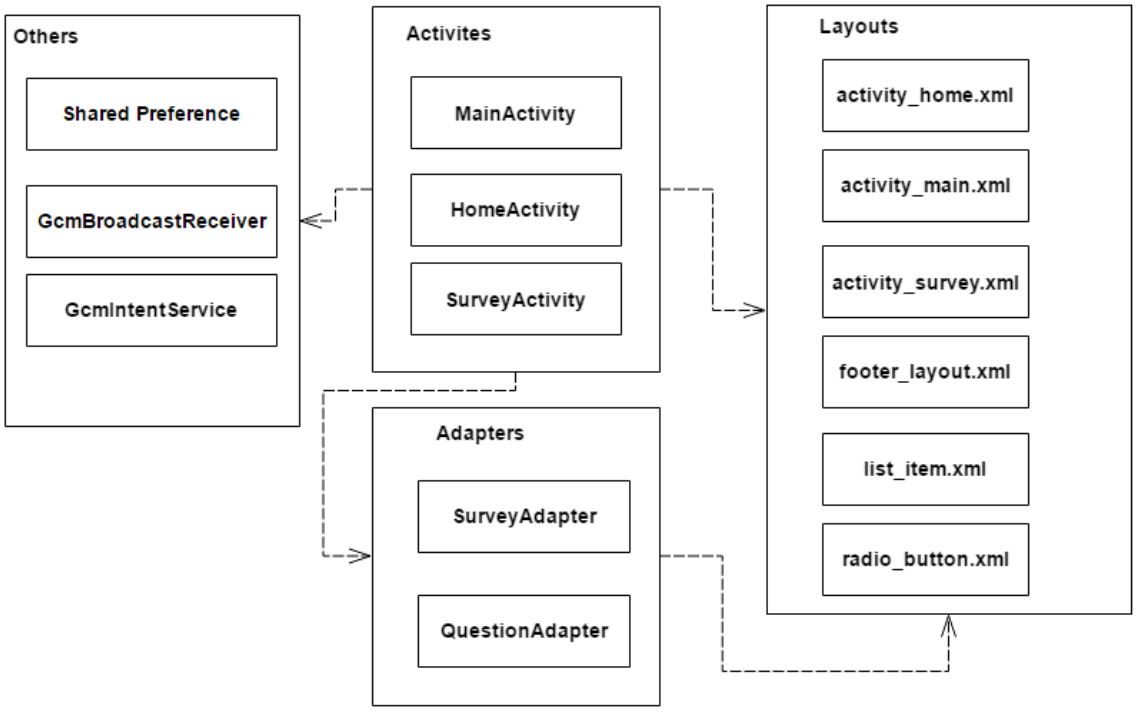

Figure 6.7 Mobile application architecture

The application contains three pages on which the user can navigate. The MainActivity class represents the login page and is the first page accessed by the user when running the application, containing the email and password fields that the user will have to complete in order to authenticate. The introduced fields will be checked on the backend server and after successful login, the athlete will be redirected to the HomeActivity consisting of the survey listing page.

The survey list page along with the survey page consists of ListView items.
A ListView is a group that displays a list of scrollable items. Items are inserted in the list using an Adapter. So the list and the list item are two separate files in the layouts folder. For example we take activity_home and list_item from the above diagram. Activity_home.xml contains only a ListView, while the list_item.xml contains two TextViews: one for displaying the survey title and one for displaying the survey description. The SurveyAdapter gets all surveys from the request, converts each item into a view and places the view in the list.

6.2.2. Authentication

The mobile application uses stateless authentication which means that no session is persisted on the server side. When the user introduces his credentials correctly, he receives a token stored in the database and will attach that token in a header in each subsequent request he makes to the server in order to be identified. This token based authentication method is commonly used nowadays for those using an API, being a good way of handling authentication for multiple users.

6.2.3. Implementation details

The authentication token received at login time will get stored in the application. For that a class named SharedPreference was created in order to keep the token and retrieve it whenever a request must be done.

For obtaining the necessary data for the mobile application, requests are sent to an API that runs on the server. This API is consumed by the client and the requested data is sent in JSON form. A POST request is sent in order to make the authentication and submit the survey data and GET requests are done in order to get the surveys with questions and answers. Transmitting network data was implemented using a library named Volley[33], that handles networking for Android using HTTP requests. Volley has support for JSON, which was the form in which the data was intended to be transmitted from the server, so the library met the application's needs.

For example the following code shows the getSurveys() function, which retrieves the available surveys, attaching the necessary headers (Authorization, Accept) to the request:

```
public void getSurveys() {

        final RequestQueue queue = Volley.newRequestQueue(this);
        String url = "http://192.168.1.5:3000/api/v1/surveys";

        final JsonObjectRequest jsObjRequest = new JsonObjectRequest
                (Request.Method.GET, url, null,getSurveysListener(),
getSurveysErrorListener()) {

            @Override
            public Map<String, String> getHeaders() throws AuthFailureError {
                Map<String, String> params = new HashMap<String, String>();
                params.put("Authorization",
SharedPreference.getValue(HomeActivity.this));
                params.put("Accept", "application/json");
                System.out.println(params);
                return params;
            }

        };

        queue.add(jsObjRequest);.

}
```

If the requested data can be accessed and we receive a response with success the following function gets the JSON Array from the request response and displays data in list format on the screen:

```
private Response.Listener<JSONObject> getSurveysListener(){
       return new Response.Listener<JSONObject>() {
            @Override
            public void onResponse(JSONObject response) {
```

[33] https://developer.android.com/training/volley/index.html

```
                try {
                    JSONArray surveys = response.getJSONArray("surveys");
                    adapter = new SurveyAdapter(HomeActivity.this, surveys);
                    listView.setAdapter(adapter);
                } catch (JSONException e) {
                    e.printStackTrace();
                }
            }

        };
    }
```

In order to display the selected survey and make it available to the athlete, the application has to retrieve the questions and answers for that specific survey, so it has to pass the survey between activities. This action was accomplished by attaching extra information to the specific Intent:

```
         v.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                try {
                    Toast.makeText(context, "You Clicked "+survey.getString("name"),
Toast.LENGTH_LONG).show();
                    Intent intent = new Intent(context, SurveyActivity.class);
                    intent.putExtra("survey_id", survey.getString("id"));
                    context.startActivity(intent);

                } catch (JSONException e) {
                    e.printStackTrace();
                }
            }
        });
```

In order to display the selected survey, questions must be shown in form of a list and answers must be displayed with radio buttons below each question. For this we created a ListView containing the text of the question and a radio button group. For displaying each question with corresponding radio button group an Adapter is used. Below can be seen the method that does the actual insertion of each item in the ListView of questions.

```
public View getView(int i, View convertView, ViewGroup viewGroup) {
        final JSONObject question  = (JSONObject) getItem(i);
        ViewHolder holder;

        LayoutInflater layoutInflater = LayoutInflater.from(context);

        if (convertView == null) {
            holder = new ViewHolder();
            convertView = layoutInflater.inflate(R.layout.radio_button, viewGroup,
false);
            holder.question = (TextView) convertView.findViewById(R.id.question);
            holder.relativeLayout = (RelativeLayout)
convertView.findViewById(R.id.relative_layout);
            convertView.setTag(holder);
        } else {
            holder = (ViewHolder) convertView.getTag();
        }
```

```
        try {
            holder.question.setText(question.getString("text"));
            createRadioButton(holder, convertView, question.getString("id"));
        } catch (JSONException e) {
            e.printStackTrace();
        }
        return convertView;
    }
```

This function calls the createRadioButton function for each question in the survey list and inserts the result in the view. The createRadioButton function retrieves the answers for the question passed as a parameter and creates a radio button for each answer with the answer text from the JSON. The function can be seen below.

```
private void createRadioButton(ViewHolder holder, final View view, final String
question_id) throws JSONException {
        final RadioButton[] radioButtons = new RadioButton[20];
        final RadioGroup radioGroup = new RadioGroup(context);
        radioGroup.setOrientation(RadioGroup.VERTICAL);
        int count = 0;
        for (int i = 0; i < answers.length(); i++) {

            if (answers.getJSONObject(i).getString("question_id").equals(question_id))
{
                radioButtons[count] = new RadioButton(context);

radioButtons[count].setId(Integer.parseInt(answers.getJSONObject(i).getString("id")));
                radioGroup.addView(radioButtons[count]);

radioButtons[count].setText(answers.getJSONObject(i).getString("text"));
                count++;
            }
        }

        holder.relativeLayout.addView(radioGroup);

        radioGroup.setOnCheckedChangeListener(new RadioGroup.OnCheckedChangeListener()
{
            @Override
            public void onCheckedChanged(RadioGroup radioGroup, int i) {
                RadioButton rb = (RadioButton) view.findViewById(i);
                listener.handleSelect(question_id, String.valueOf(rb.getId()));

            }
        });
    }
```

6.3. Database Design

The system uses a PostgreSQL database that contains 8 tables. Each table corresponds to a model in the application. In order to add tables and fields to the database, or to perform any type of database changes, rails can generate migrations[34]. Migrations are a convenient way to change the database schema over time in a consistent and easy way. Each migration can be thought of as representing a new version of the database. A schema

[34] http://edgeguides.rubyonrails.org/active_record_migrations.html

is initially empty and each migration modifies it to add or remove tables, columns or entries. When a migration is executed the *db/schema.rb* file will be also updated to match the new structure of the database. A migration that adds a surveys table to the database with the necessary attributes look like this:

```
class CreateSurveys < ActiveRecord::Migration
  def change
    create_table :surveys do |t|
      t.string :name
      t.string :description
      t.string :status
    end
  end
end
```

In the next diagram we can see all tables from the database and the relations between them. The primary key are specifically identified and so are the foreign keys in each table.

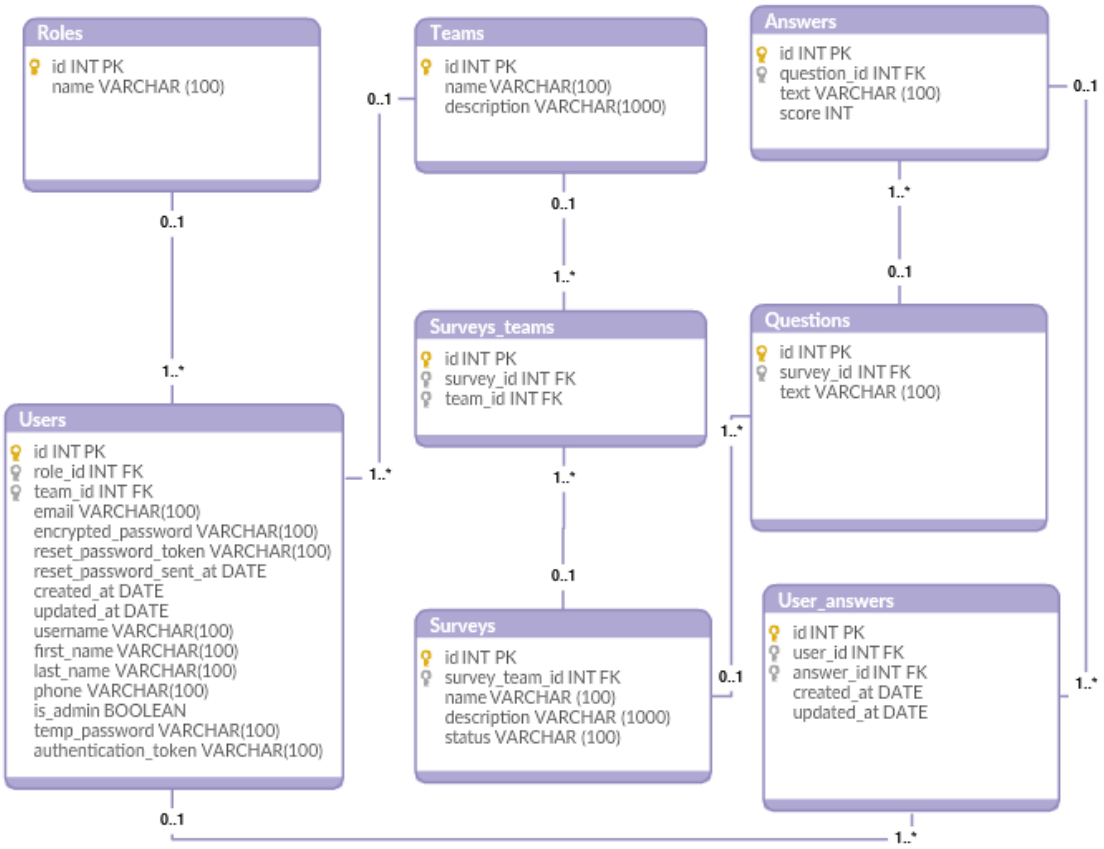

Figure 6.8 Database diagram

The database tables do not contain duplicate data so there is no redundancy at data level. Whenever a modification is performed, data is updated in a single table. The tables contain attributes that concern only a specific entity. Below is an analysis performed in order to determine the level of normalization of the system database.

First normal form (1NF) states that each table must contain a primary key, not have repetitive groups and each attribute can receive an atomic value, not a set of values. In

addition each attribute must be dependent on the primary key. The system database meets these criteria.

Second normal form (2NF) conversion is done by eliminating partial dependencies (dependencies of an attribute to only a part of the composite primary key). The system database does not have composite primary keys, the primary key being composed of a single attribute. This means that the database is already in 2NF so this conversion is not required.

Third normal form (3NF) conversion is done by eliminating the transitive dependencies (the dependencies of an attribute that is not part of the primary key to another attribute that is not part of the primary key either). Generally this dependencies only happen if one table holds data from different fields, which is not the case of our database. In the system database all tables' data concern only a specific model.

Boyce Codd normal form (BCNF) conversion is done by eliminating the functional dependencies given by the fact that an attribute that is not part of the primary key is a determinant of another attribute from the primary key. In the system database, each created table has a primary key that contains a single attribute which is the determinant of all the other attributes from the table. Therefore our database qualifies for the Boyce Codd normal form.

6.4. Deployment

The deployment diagram in Figure 5.9 shows what hardware components can be used and which software components run on each of them. The connections between components can be also depicted and the communication is done through HTTP requests.

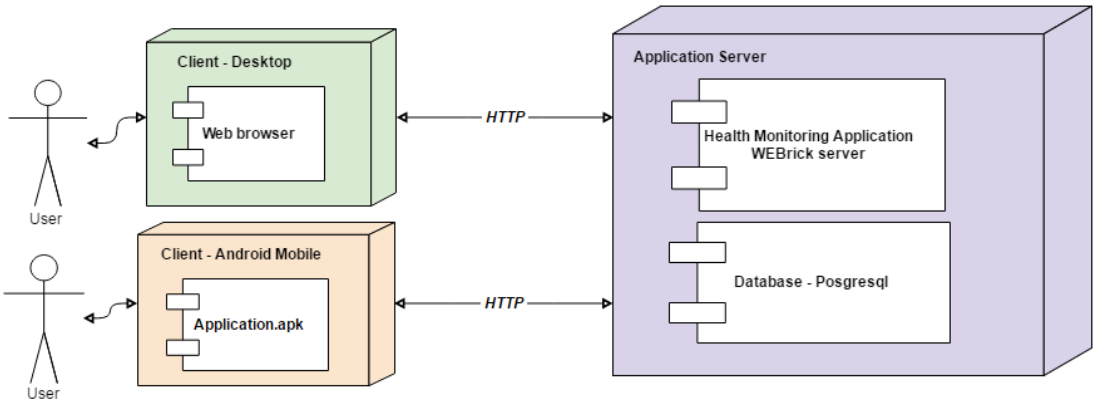

Figure 6.9 Deployment Diagram

The purpose of this diagram is to show the hardware resources necessary in order to run the system. The server hosts the PostgreSQL database and will be accessed by both client applications in order to perform the necessary operations.

The mobile application can be accessed from any device running on Android with version at least 9 and requires access to the internet in order to function properly. In development environment, for using the Android phone or tablet to test the application, the localhost on which Rails server starts must allow public connections, so the application

connected to the same network can access it. In order to do this the server was started with the ***- -binding 0.0.0.0*** option and the requests were made using the computer IP.

The communication between the Android application and the server is done through HTTP requests to the Rails API that gives back responses in JSON form. The mobile application then displays the received data to the athlete, in order to enable him to perform the required operations.

The web application can be accessed from any browser, either on a mobile device or desktop environment but it is primary conceived for use on a PC.

6.5. Version control

When developing the proposed system, version control was employed by using Git[35]. Git is a version control system used for software development in order to back up your source code. Every time a significant change was done, it was committed, each commit being a backup of the files at that point in time. Each commit points back to the one that came before, so version history can be seen from the BitBucket repository. The main benefits of using version control systems is that you can see anytime how your code looked like in the past.

Each repository has a ***master*** branch and other branches can be created in order to keep independent versions of the files when you work on separate tasks. This is useful to avoid the interference of multiple streams of work and then in order to combine everything back to a single unitary project, a merge request is performed. After reviewing the changes and solving possible conflicts, merge is applied. However, this is usually used when multiple people work on the same project, so in the case of the proposed software solution only the master branch was used, pushing all commits there.

6.6. Cloud deployment

Cloud computing, also known as the cloud, is a synonym of the Internet. Cloud can serve a wide range of functions over the Internet like storage and virtual servers, therefore defining itself as a broad collection of services that provide shared processing resources on demand.

6.6.1. Cloud service models

Cloud computing contains three cloud service models that ca be seen in Figure 5.10 and are explained below.

- **Infrastructure as a Service (IaaS)** – the most basic cloud-service model that provides computing infrastructure: virtual machines and other resources, as a service to subscribers. The consumers can manage the operating system and their software application and they cannot manage the cloud underlying infrastructure.[25]
- **Platform as a Service (PaaS)** – offers a development environment to application developers including operating system, programming language

[35] https://en.wikipedia.org/wiki/Git_(software)

execution environment, database and web server. Developers can run their software solutions on a cloud platform without the trouble of managing the underlying hardware and software. So the consumers only have control over the deployed applications and they also have the ability to create own configuration settings for the application environment [25].

- **Software as a Service (SaaS)** – provides access to application software and databases but consumers cannot manage the infrastructure and platforms where the application runs [25].

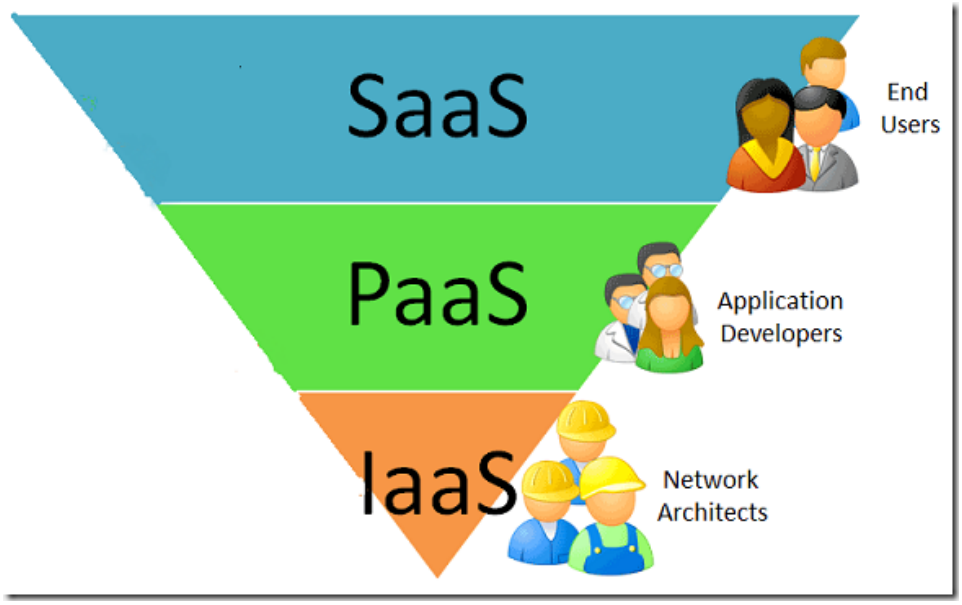

Figure 6.10 Cloud Computing Models[36]

6.6.2. Cloud deployment models

Cloud computing contains various deployment models [26]:

- **Private cloud –** the platform for cloud computing is implemented on a secure environment guarded by a firewall. It only gives access to the authorized users
- **Public cloud –** when the services are delivered over a network that is open for public use. The difference between public and private clouds lies in the level of security offered for various services given to the public cloud consumers by the cloud hosting providers.
- **Hybrid cloud** – a composition of two or more cloud types that stay distinct entities but are bound together, providing the advantages of multiple deployment models. Non-critical resources like development and test can reside in the public cloud while the sensitive data must be stored internally.

[36] http://blog.samisa.org/2011/07/cloud-computing-explained.html

- **Community cloud –** cloud setup is shared between community participant organizations. Community member share similar privacy, performance and security aspects.

6.6.3. Heroku

One of the advantages of choosing cloud providers to host the Rails backend server is that most cloud services are able to offer a close to 100% uptime. In addition, they provide scalability, meaning that we can scale our system and increase hardware specifications as much as we need. Most popular cloud platforms that can host Rails applications are: Amazon AWS[37], Digital Ocean[38], Engine Yard[39] and Heroku.

Heroku was chosen for the deployment from the above list because it provides support for Ruby applications, it's free at the beginning and deploys are automated using git push. So given the fact that git was used for version control, it is really simple and easy to deploy, once the repository is updated.

Heroku is a cloud platform service for deploying and running modern applications including Rails applications. It makes the deployment experience very short and easy. The applications run in smart containers named dynos with curated, automatically patched operating system images. Heroku Postgres and Heroku Redis are fully managed data services operated by Heroku. Heroku runtime provides a set of services that orchestrate and manage the execution and scale of the application. Users can access the application from any device using HTTP [27].

In order to run the application on Heroku, no changes needed to be done and as Rails is an established framework, Heroku can figure out which parts of the application are runnable. The Heroku platform uses git as the primary means of deploying the application, so when an application on Heroku was created, the web application's local repository was associated to it and deploying the code was basically pushing the master branch into the new git remote created.

When Heroku receives the application code, it begins a build of the application, fetching all dependencies included in the Gemfile, as well as generating files based on the asset pipeline. When deploying an application for the first time, Heroku automatically boots a dyno, loads it with your slug (generated output of the system build executed in the last step that is ready for execution) and execute the command associated with the web process type in the Procfile. Each time a new version of the application is deployed, a new slug is created and a release is generated.

[37] https://aws.amazon.com/
[38] https://www.digitalocean.com/
[39] https://www.engineyard.com/

Chapter 7. Testing and Validation

This chapter presents the main testing processes that were applied for the whole system. Testing in general doesn't guarantee that a system functions correctly and completely in all situations, but can help identify possible problems that maybe weren't expected by the developer and drive him on the correct path of finding solutions for the problems found. Testing can be defined as a validation process of the fact that the system was implemented meeting the imposed requirements. System testing is done based on the functionalities offered by the system, through the user interface, by analyzing use cases' correctness. The behavior in different situations is analyzed by introducing valid and invalid data and observe the provided feedback and the update of the stored data.

System testing is very important, sometimes a crucial step of a software product delivery. In order for a system to behave as expected all modules and the communication between them must operate correctly. The system's development process was an iterative one and as a result each feature implemented was tested after finishing implementing it. Therefore, each component of the system was manually tested, starting from some simple success scenarios to scenarios where invalid data was introduced in order to test the implemented validations and the display of the appropriate messages. This manual testing model applied at the end of a component's implementation was used for both the web application and the mobile application.

Testing the Rails API for the Android application was done before the Android application was implemented, so this was possible using Postman. Requests were sent to the specific URLs with the appropriate Headers and methods and the JSON response was shown on the screen. So even before the mobile application was developed in order to test if the backend sent the appropriate response, these functionalities were tested to make sure that the Rails application was fully functional. Of course they were again tested when the Android application was ready to make requests and display responses.

Besides manual testing, the evaluation of the proposed system was done by comparing it with the pmSys system (Player Monitoring System) developed as a master thesis project by a team in Norway. They developed a software solution for football players in order to monitor their wellness, injury risks and RPE. The pmSys application does this by connecting to an external server named Ohmage.

PmSys system is composed of 3 modules:

- A mobile component
- A web component
- A push notifications module

The athletes' health monitoring system is composed of 2 modules but it integrates the sending of the push notifications by using GCM:

- A mobile component
- A web component

PmSys mobile application features:

- Cross platform(iOS, Android)
- Notifications

- Reminders
- Change password
- Survey response registration

Function	Survey list	Survey response registration	Notifications	Reminders	Change password	Cross platform
pmSys	Yes	Yes	Yes	Yes	Yes	Yes
Athletes' Health Monitoring System	Yes	Yes	Yes	No	No	No

Table 7.1 pmSys vs Athletes' Health Monitoring System mobile application features

Even if the mobile application doesn't have all the features that pmSys offers, the main focus of our system was to be able to answer surveys as quick as possible. That's why the questions are displayed on the same page in a scrollable list, having a submit button at the end of the list, instead of having questions appear each on a separate view with a next button, like pmSys application displays the surveys.

PmSys web portal features:

- Detailed visualization of players' results
- Sending push notifications

Systems	User management	Team management	Survey management	Visualization	Notifications
pmSys	No	No	No	Yes	Yes
Athletes' Health Monitoring System	Yes	Yes	Yes	Yes	Yes

Table 7.2 pmSys vs Athletes' Health Monitoring System web application features

PmSys uses Ohmage as backend so all the users, the teams, the roles, the surveys are handled from there with the proper rights. That means that you cannot add or remove users or manage teams from the pmSys application. It is also not possible to create and assign surveys to teams. All this data comes from making requests to the Ohmage backend but it cannot be manipulated by the pmSys administrator user. Our application provides an accessible way of handling all this management operations by the admin user directly from the web application. In addition, the application offers the coaches the ability of

creating own surveys with the purpose of gathering subjective data about different fields that the coach considers of importance, in order to monitor athlete performance and detect possible injuries.

In conclusion, the proposed system compared to a similar system that is actually used today by multiple Norwegian football teams, has advantages and disadvantages, providing some features that the pmSys system does not support but also not providing others that pmSys offers. However, our system reached the desired requirements and the results of the testing are the expected ones, stating that the proposed system is valid and works properly.

Chapter 8. User's manual

8.1. Web application

The users of the web application don't need to have specific hardware resources, only a functional browser in order to be able to access the application. The deployment of the application can be done in any cloud computing system that support ruby.

The dependencies of the web application:

- Ruby 2.3.0
- Rails 4.2.6
- PostgreSQL 9.5

8.1.1. Installation description

The next steps are conceived to offer the possibility of installing the web application on a personal computer.

The recommended operating system for installing and running the web application is Linux because of the fact that many binary gems are difficult or impossible to install on Windows. Anyhow, installing and running the application on Linux is done very easily following the next steps:

1. Installing Ruby

We install Ruby using RVM (Ruby Version Manager) so that we can use various versions of ruby on the same computer if we need or if we want to use a new ruby version we only need to run a command to change the default ruby version used. We install RVM by opening a terminal and running:

```
\curl -sSL https://get.rvm.io | bash.
```

To display a list of all known rubies we run:

```
rvm list known.
```

We want to install ruby 2.3.0 and use it so we run the following commands:

```
rvm install 2.3.0
rvm use 2.3.0
```

To check if this worked correctly:

```
ruby -v.
```

2. Installing Rails

Since Rails has many dependencies, we will need to install a JavaScript runtime like NodeJS. This allows us to use the Asset Pipeline in Rails which combines and minifies javascript files to provide a faster production environment. To install NodeJS:

```
curl -sL https://deb.nodesource.com/setup_4.x | sudo -
E bash -
sudo apt-get install -y nodejs
```

And now, to install Rails 4.2.6:

```
gem install rails -v 4.2.6
```

Now that Rails is installed, we check that everything installed correctly:

```
rails -v
```

3. Installing PostgreSQL

In order to install PostgreSQL we need to add the PostgreSQL repository:

```
sudo sh -c 'echo "deb http://apt.postgresql.org/pub/repos/apt/ `lsb_release -cs`-pgdg main" >> /etc/apt/sources.list.d/pgdg.list'
```

After this we need to import the GPG key of the repository:

```
wget -q https://www.postgresql.org/media/keys/ACCC4CF8.asc -O - | sudo apt-key add -
```

Update the package list:

```
sudo apt-get update
```

The next command installs the latest version of PostgreSQL:

```
sudo apt-get install postgresql postgresql-contrib
```

For a better view of the database pgAdmin is also installed:

```
sudo apt-get install postgresql-9.5 pgadmin3
```

4. Running the project

The last step before being able to use the application is to start the server and access the server location from the browser. We start the rails server using the following command in a terminal:

```
rails s
```

If the server starts successfully, then our project will be available at http://localhost:3000, the server administration address.

8.1.2. User manual

The first page the user sees is the login page because the system can be accessed only by authenticated users. In order to authenticate successfully, the users have to introduce a username and a password in the corresponding fields on the login page.

The login page is shown below:

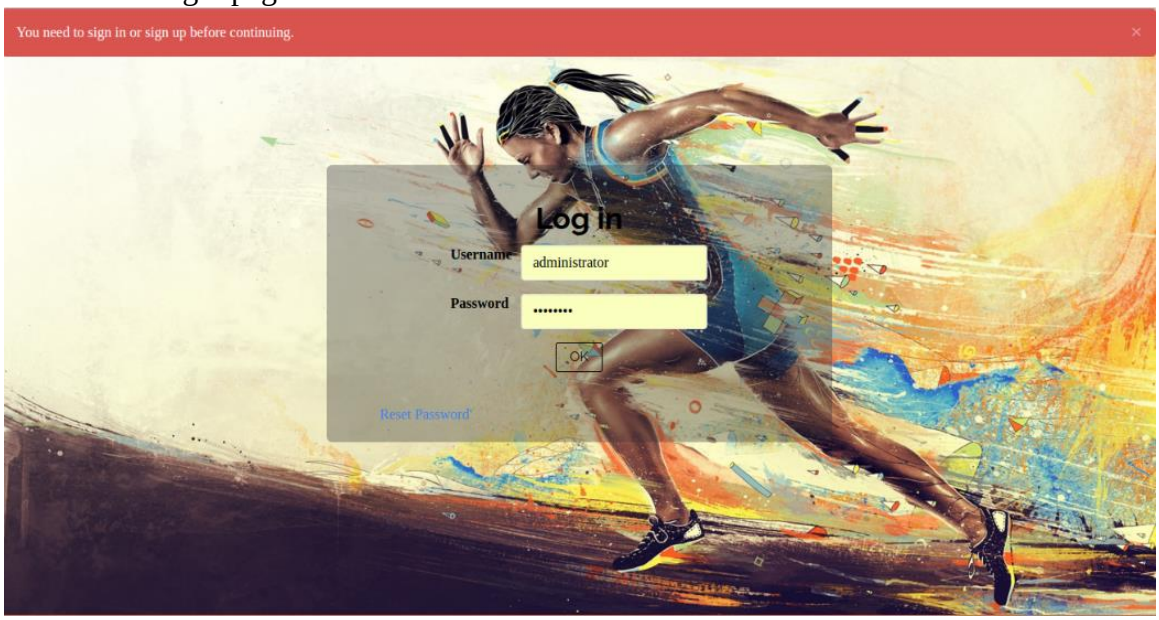

Figure 8.1 System login page

Once the authentication finalizes successfully, the users will be able to use the application. The Home page is the first page they see after logging in.

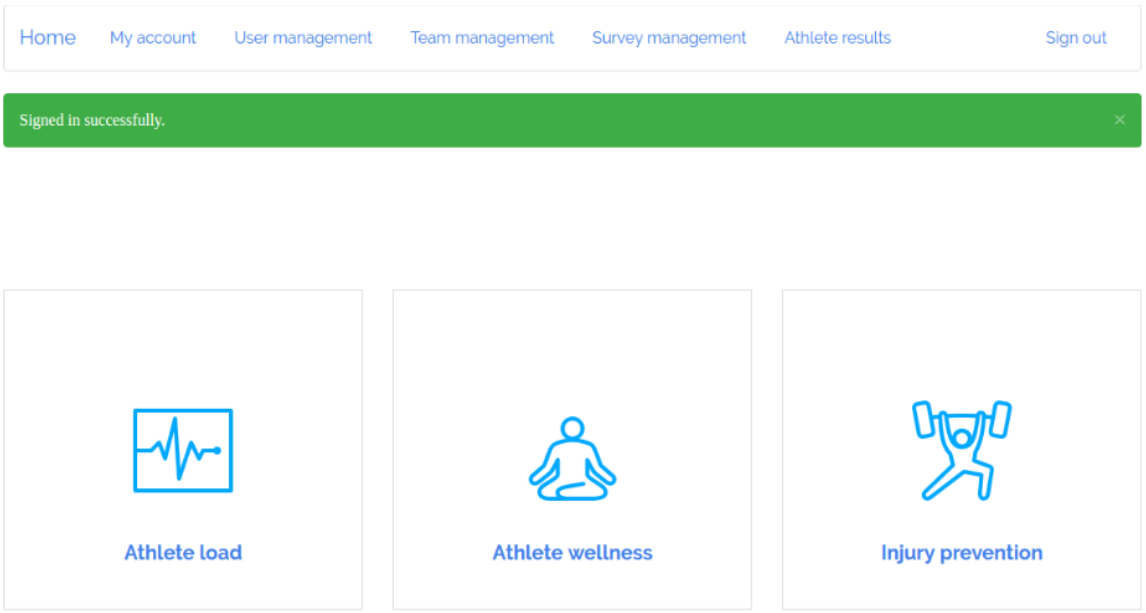

Figure 8.2 System Home page

As it can be seen in the above figure, we have a navigation at the top of the screen from where we can access other features offered by the application. For example if we click on Survey management we will see the following page:

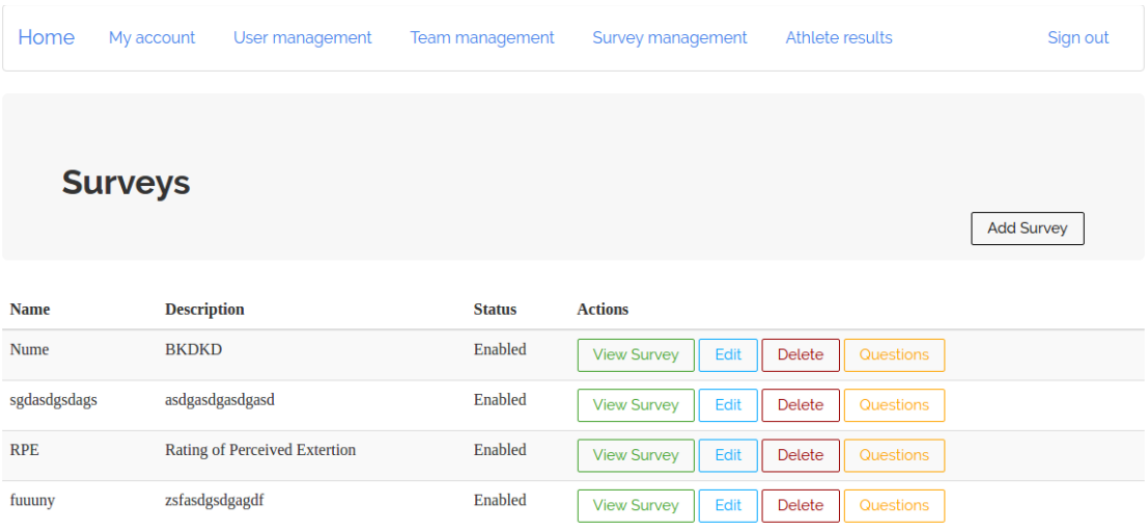

Figure 8.3 Survey view

On this page if we want to visualize a survey, we only have to push View Survey in order for the survey page to appear.

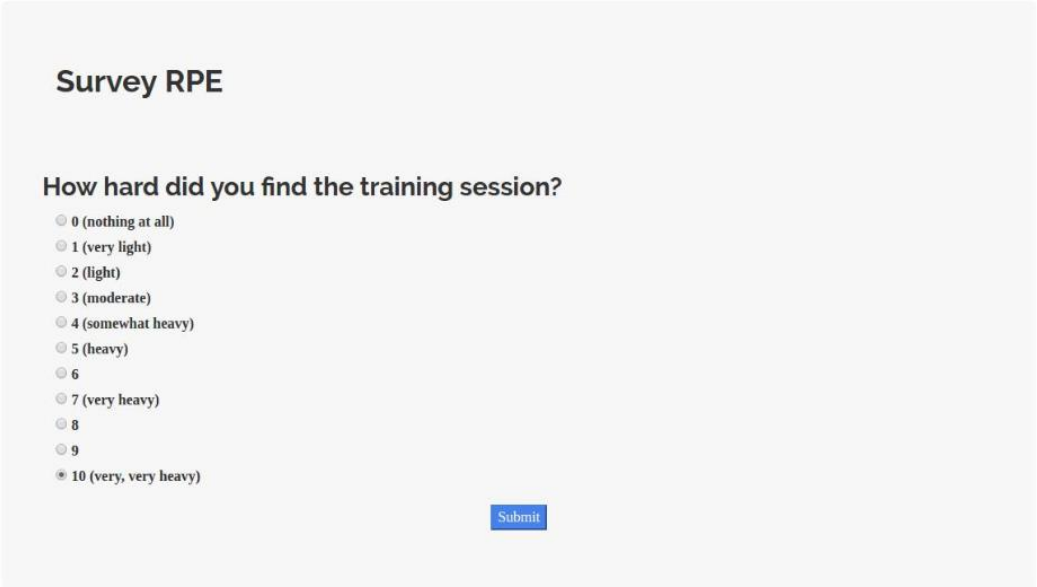

Figure 8.4 RPE survey

For viewing results we can access the View results page from the top navigation. The page displays all the teams from the system and we can choose any team to view its results by survey for all team or by individual member in team.

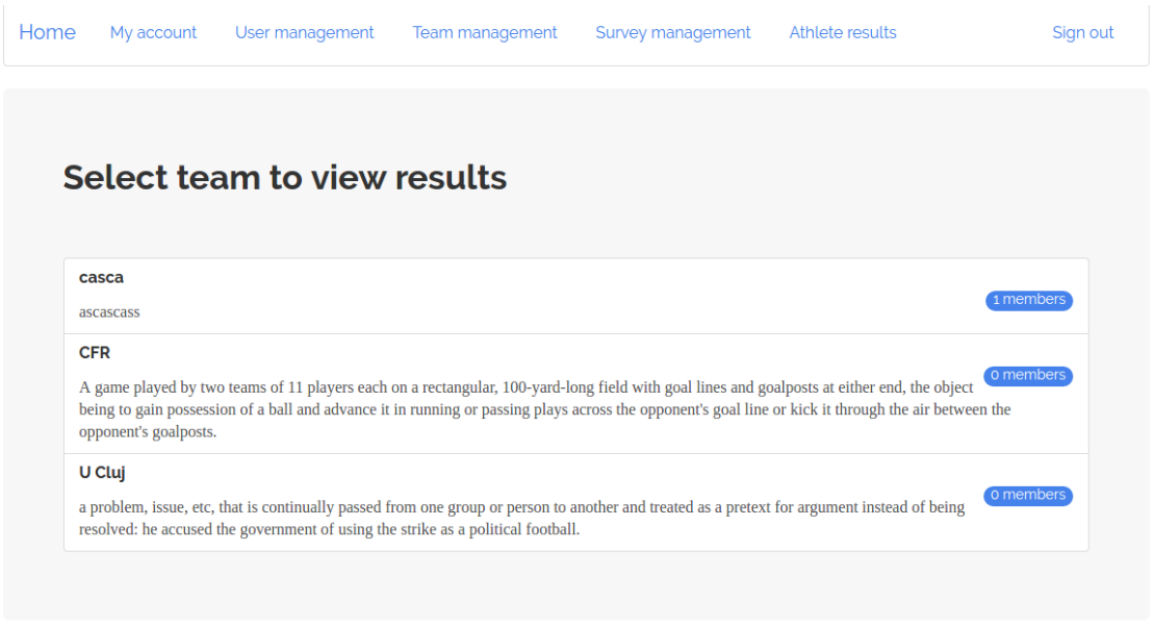

Figure 8.5 View results page

8.2. Mobile application

The mobile application needs Android OS to run. The application can be used as long as it is in the same local network with the backend Rails server. This can happen if the Android devices and the device that hosts the server are connected to the same wireless network.

8.2.1. Installation description

1. **Hardware Requirements**
   - Mobile device: phone or tablet
   - Internet connection

2. **Software requirements**
   - Android OS
   - Minimal version for the operating system is 9.0

3. **Installing the application**

- Android supports sideloading, allowing you to install applications outside of Google Play. This is disabled by default so we need to go to *Settings-> Security* and enable the Unknown sources check box.
- Download the .apk file on a computer
- In case the user does not have a file management application, he can download one from Google Play
- When the USB connection with the device is done, the .apk file is copied to the device file system.
- We navigate with the use of the file manager application to the place the application has be uploaded and we click on it, this action triggering the application to start installing on the device. We need to give the application access to the Internet in order to function correctly.

Chapter 9. Conclusions

In this thesis we have proposed and implemented a monitoring system for athletes, that performs data collection, processing and presentation. The system is an easy approach toward monitoring athletes by collecting subjective data and presenting it to the coaches for easier analysis of athlete performance and detection of possible injury factors. The system realized is a combination of a mobile application used to report data and a web application to monitor and analyze the data.

9.1. Main contributions

The proposed system managed to achieve its ultimate purpose of being a system that can be used to monitor athletes through gathering subjective data.

We implemented a method of self-reporting by using surveys to collect data using a mobile application. The submitted data is sent and stored on the backend server. The web application retrieves the data from the server and presents it to the coach in different charts. Therefore, the coach is able to analyze it better and faster in order to detect any potential injury causing factors and monitor the athlete performance by reviewing training related answers. Basically, the coach can introduce in a survey, any question that he considers of importance for the analysis of the athlete state that could impact his further performance.

Secondary objectives have also been realized: the users are identified by their role, they can authenticate using username and password, team management, user management, and survey management are implemented facilitating the addition of any survey from the user interface by introducing questions and corresponding answers. Also the coach can visualize from the results section the results of the athletes corresponding to his team in charts.

The system compared with similar systems has advantages and disadvantages. While the proposed system offers direct survey creation from the web application user interface, pmSys which uses Ohmage as the backend server, needs to upload surveys to the third party software in a specific format. However, pmSys mobile application supports offline mode which is an aspect that our mobile application does not treat. In conclusion, the proposed system can compete with other similar software systems but is not necessarily better, having pluses as well as minuses. The system was deployed in cloud and is accessible from any computer with internet connection.

9.2. Future work

Generally, each software solution developed can be extended and improved in order to better satisfy client needs. Further development of a system can include anything from implementing new functionalities to improving the ones implemented, in order to achieve better performance. In the next section possible future work and improvements for the proposed system are presented.

9.2.1. Mobile application

- **Mobile application for general use**: The mobile application could be extended in order to also offer benefits to other types of users such as coaches and medical staff.

Due to mobile applications' accessibility, they represent the preferred type of applications nowadays.

- **iOS and Windows Phone**: Currently, the proposed solution only works on Android devices, so offering similar mobile applications for iOS and Windows Phone could be a great future work. Nowadays, Android users are approximately as many as the iOS users so athletes surely won't all be using Android, therefore the necessity of offering an application also for the other platforms is a concern.
- **Offline support**: Currently the application requires Internet connection in order to function properly. An offline mode could be developed for the athletes so they could answer the surveys, save them and submission would be done when Internet connection will become available.

9.2.2. Web application

- **Web application for general use:** The web application could be extended to be used by all actors of the system. This means enabling athletes to also use the web application in order to respond to the surveys.
- **Artificial intelligence:** Using artificial intelligence, the application could perform machine learning on collected data in order to automatically detect injuries and pass a message to the user recommending him what should do. The system could also send a message to the coach when an athlete is in danger to be injured.
- **Objective monitoring:** the system would be more complex if third party monitoring devices could be integrated. These would collect objective data and present it, in order to enable the coach to make a comparison between subjective and objective data.
- **Multiple language support:** Given the fact that the application could be used in different countries, maybe English is not spoken by every athlete or some of them do not understand some words. In order to exclude these scenarios in which the athletes may give deceptive answers, language translations could be used and each user could set the language in which the application will be presented. However, this will also mean a greater trouble for the coach, given the fact that he will have to insert translations for the introduced surveys.

Chapter 10. Bibliography

[1] J. Reed, "Coach and Athlete Perceptions of an Athlete Monitoring and Strength and Conditioning Program", East Tennessee State University, 2014.

[2] E. Chi, G. Borriello and N. Davies, "Guest Editors' Introduction: Pervasive Computing in Sports Technologies", *IEEE Pervasive Computing*, vol. 4, no. 3, pp. 22-25, 2005 [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1495386&tag=1. [Accessed: 02- Feb- 2016]

[3] V. Gambetta, *Athletic Development: The Art & Science of Functional Sports Conditioning*. Human Kinetics Publishers, 2006.

[4] A. Saw, P. Gastin and L. Main, "Monitoring the athlete training response: subjective self-reported measures trump commonly used objective measures: a systematic review", *British Journal of Sports Medecin*, 2015 [Online]. Available: http://bjsm.bmj.com/content/early/2015/09/09/bjsports-2015-094758. [Accessed: 02- Feb- 2016]

[5] A. Saw, P. Gastin and L. Main, "Monitoring athletes through self-report: Factors influencing implementation", *Journal of sports science & medicine*, pp. 137-146, 2015 [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/25729301. [Accessed: 02- Feb- 2016]

[6] B. Piggott, "The relationship between training load and incidence of injury and illness over a pre-season at an Australian Football League Club", Edith Cowan University, 2008.

[7] C. Foster, "Monitoring training in athletes with reference to overtraining syndrome.", Medicine *and Science in sports and exercise*, pp. 1164-1168, 1998 [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/9662690. [Accessed: 03- Feb- 2016]

[8] K. Vuong, "PmSys: a monitoring system for sports athlete load, wellness & injury monitoring", University of Oslo, 2015.

[9] S. Eriksen, M. Georgsson, M. Hofflander and J. Lundberg, "Health in Hand: Putting mHealth Design in Context", *Usability and Accessibility Focused Requirements Engineering (UsARE)*, pp. 36 - 39, 2014 [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6890999. [Accessed: 10- Mar- 2016]

[10] A. Singh Gagneja and K. Gagneja, "Mobile Health (mHealth) Technologies", *2015 17th International Conference on E-health Networking, Application & Services (HealthCom)*, pp. 37 - 43, 2015 [Online]. Available:

http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7454470. [Accessed: 10- Mar- 2016]

[11] H. Tangmunarunkit, C. Hsieh, F. Alquaddoomi and N. Ramanathan, "Ohmage: A General and Extensible End-to-End Participatory Sensing Platform", *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 6, no. 3, 2015 [Online]. Available: http://dl.acm.org/citation.cfm?id=2717318. [Accessed: 18- Mar- 2016]

[12] T. Hoang, "pmSys Implementation of a digital Player Monitoring System Thuc Tuan", University of Oslo, 2015.

[13] S. Ruby, D. Thomas and D. Heinemeier Hansson, *Agile Web Development with Rails 4*. Pragmatic Bookshelf, 2013.

[14] *Ruby on Rails*. [Online]. Available: https://en.wikipedia.org/wiki/Ruby_on_Rails. [Accessed: 16- May- 2016]

[15] "SQLite vs MySQL vs PostgreSQL: A Comparison Of Relational Database Management Systems". [Online]. Available: https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems. [Accessed: 13- Jun- 2016]

[16] "Android (operating system)". [Online]. Available: https://en.wikipedia.org/wiki/Android_(operating_system). [Accessed: 10- Jun- 2016]

[17] *Bootstrap* [Online]. Available: https://en.wikipedia.org/wiki/Bootstrap_(front-end_framework). [Accessed: 14- May- 2016]

[18] *About HAML*. [Online]. Available: http://haml.info/about.html. [Accessed: 06- May- 2016]

[19] *Devise*. [Online]. Available: https://github.com/plataformatec/devise. [Accessed: 08- Jun- 2016]

[20] *Trailblazer Reform*. [Online]. Available: http://trailblazer.to/gems/reform/. [Accessed: 12- Jun- 2016]

[21] *Rpush*. [Online]. Available: http://www.rubydoc.info/gems/rpush/1.0.0. [Accessed: 08- Jun- 2016]

[22] *Chartkick*, 2016. [Online]. Available: http://chartkick.com/. [Accessed: 12- Jun- 2016]

[23] *Transmitting Network Data Using Volley*. [Online]. Available: https://developer.android.com/training/volley/index.html. [Accessed: 13- Jun- 2016]

[24] *jQuery*. [Online]. Available: https://ro.wikipedia.org/wiki/JQuery. [Accessed: 10- Jun- 2016]

[25] *Cloud computing*. [Online]. Available: https://en.wikipedia.org/wiki/Cloud_computing#Service_models. [Accessed: 15- Jun- 2016]

[26] *Cloud computing deployment models*. [Online]. Available: https://www.ibm.com/developerworks/community/blogs/722f6200-f4ca-4eb3-9d64-8d2b58b2d4e8/entry/4_Types_of_Cloud_Computing_Deployment_Model_You_Need_to_Know1?lang=en. [Accessed: 12- Jun- 2016]

[27] *Heroku*. [Online]. Available: https://www.heroku.com/platform#platform-diagram-detail. [Accessed: 14- Jun- 2016]

Appendix 1 List of figures

Figure 3.1 Borg RPE Scale 8
Figure 3.2 Example of Wellness questionnaire 9
Figure 3.3 Example Injury questionnaire [8] 10
Figure 3.4 Ohmage software features 11
Figure 3.5 Ohmage system architecture [11] 12
Figure 4.1 Conceptual architecture of the system 15
Figure 4.2 Use case diagram 18
Figure 4.3 Flow diagram of creating a survey 22
Figure 4.4 Generic Rails architecture diagram 24
Figure 4.5 Android architecture 26
Figure 4.6 Google Cloud Messaging 30
Figure 5.1 Rails MVC application structure 31
Figure 5.2 Web application block diagram 32
Figure 5.3 Models 32
Figure 5.4 Views 33
Figure 5.5 Forms 33
Figure 5.6 Controllers 34
Figure 5.7 Mobile application architecture 39
Figure 5.8 Database diagram 43
Figure 5.9 Deployment Diagram 44
Figure 5.10 Cloud Computing Models 46
Figure 7.1 System login page 52
Figure 7.2 System Home page 53
Figure 7.3 Survey view 53
Figure 7.4 RPE survey 54
Figure 7.5 View results page 54

Appendix 2 List of tables

Table 3.1 Comparison between Ohmage and pmSys we application features.....13
Table 3.2 Comparison between Ohmage and pmSys mobile application features14
Table 6.1 pmSys vs Athletes' Health Monitoring System mobile application features.....49
Table 6.2 pmSys vs Athletes' Health Monitoring System web application features 49

Appendix 3 Glossary

RPE	Rating of perceived exertion
mHealth	Mobile Health
eHealth	Electronic Health
pmSys	Player Monitoring System
API	Application Programming Interface
UC	Use Case
DRb	Distributed Ruby
RoR	Ruby on Rails
REST	Representational State Transfer
MVC	Model View Controller
JSON	JavaScript Object Notation
XML	Extensible Markup Language
HTML	Hyper Text Markup Language
CSS	Cascading Style Sheets
CoC	Convention over Configuration
DRY	Don’t Repeat Yourself
HTTP	Hypertext Transfer Protocol
HTTPS	Secure Hypertext Transfer Protocol
ORM	Object Relational Mapping
SQL	Structured Query Language
RDBMS	Relational Database Management System
MVCC	Multi-version Concurrency Control
ACID	Atomicity, Consistency, Isolation, Durability
VM	Virtual Machine
OS	Operating System
SDK	Software Development Kit
HAML	HTML Abstraction Markup Language
DOM	Document Object Model
ERB	Embedded Ruby
IP	Internet Protocol
GCM	Google Cloud Messaging
iOS	iPhone Operating System
RPC	Remote Procedure Calls
URL	Uniform Resource Locator
IDE	Integrated Development Environment
NF	Normal Form
BCNF	Boyce Codd Normal Form
IaaS	Infrastructure as a Service
PaaS	Platform as a Service
SaaS	Software as a Service
AWS	Amazon Web Services
RVM	Ruby Version Manager