ClubNet
Software Requirements Document
Version 1.0.1

Project team
J.G.C. Brouns | 0856180
S. Chen | 0842556
K. van Eenige | 0862649
S.S. Iyer | 0866094
T.L. Komar | 0870470
D. van der Laan | 0868405
T. Sostak | 0842556
K.W. Verhaegh | 0860736
J. Verhagen | 0816613

Project managers
C.N.I.W. Schappin
N.W. Wielinga

Project supervisor
N. Zannone

Customer
G. Budziak
Abstract

This document contains the software requirements for the ClubNet software system, which is developed by The Brofessionals development team. The requirements in this Software Requirements Document (SRD) satisfy the requirements in the User Requirements Document [1]. This document complies with the Software Engineering Standard, as specified by the European Space Agency [4].
# Contents

1 Introduction 9  
  1.1 Purpose ................................................. 9  
  1.2 Scope .................................................. 9  
  1.3 Definitions and Abbreviations ....................... 10  
    1.3.1 Definitions ........................................ 10  
    1.3.2 Abbreviations ...................................... 10  
  1.4 List of references ..................................... 11  
  1.5 Overview ............................................. 11  

2 General Description 12  
  2.1 Relation to current projects .......................... 12  
  2.2 Relation to Predecessor and Successor Projects ........ 12  
  2.3 Function and Purpose .................................. 12  
  2.4 Environment ......................................... 13  
  2.5 Relation to Other Systems ............................ 14  
    2.5.1 Coach Assist ....................................... 14  
  2.6 General Constraints .................................. 14  
    2.6.1 Security .......................................... 14  
    2.6.2 Extensibility and Adaptability .................... 15  
    2.6.3 Usability ......................................... 15  
  2.7 Model Description .................................... 16  
    2.7.1 Environment Model ................................. 16  
    2.7.2 Class Diagrams .................................... 18  
    2.7.3 Model-View-Controller Model ....................... 32  
    2.7.4 Data Model ........................................ 33  
    2.7.5 Message Sequence Diagrams ........................ 39  

3 Specific Requirements 57  
  3.1 Functional Requirements ............................... 58  
    3.1.1 User ................................................ 59  
    3.1.2 Coach .............................................. 63  
    3.1.3 Player ............................................. 65  
    3.1.4 PR user ........................................... 65  
    3.1.5 Feed ............................................... 69  
    3.1.6 FeedItem .......................................... 70  
    3.1.7 Note ................................................ 71  
    3.1.8 Exercise .......................................... 71
3.1.9 Training Schedule .................................................. 71
3.1.10 Chatable .......................................................... 72
3.1.11 Suggestion Feed Item ............................................. 72
3.1.12 Heroes ............................................................ 73
3.1.13 Betting Round ...................................................... 73
3.1.14 Sponsoring ........................................................ 74
3.1.15 Practicality Form .................................................... 75
3.1.16 Exercise Poll ....................................................... 77
3.1.17 Betting Pool ........................................................ 78

3.2 Non-functional Requirements ......................................... 80
3.2.1 ClubNet Environment .............................................. 80
3.2.2 Web Environment ................................................... 81
3.2.3 Back-end ............................................................ 81
3.2.4 Front-end ............................................................. 81
3.2.5 Supportability ........................................................ 82
3.2.6 Reliability ............................................................ 82
3.2.7 Performance ........................................................ 82
3.2.8 Usability .............................................................. 83

4 Requirements Traceability Matrix ....................................... 84
4.1 URD to SRD ............................................................. 84
4.2 SRD to URD ............................................................. 88

A User Interface - App ..................................................... 94
A.1 Log In Screen .......................................................... 94
A.2 Forgot Password Screen and Reset Password Screen ............ 95
A.3 Feed ................................................................. 96
A.3.1 Creating a New Item ............................................... 97
A.3.2 Creating an Exercise Poll ......................................... 98
A.3.3 Manipulating a Feed Item ......................................... 99
A.3.4 Filter ............................................................... 100
A.4 Sidemenu ............................................................... 101
A.5 Chat ................................................................. 102
A.5.1 Chat Screen ........................................................ 102
A.5.2 New Chat Screen .................................................. 103
A.5.3 Chatting With Someone .......................................... 104
A.6 Profile Screen ........................................................ 105
B User Interface - Web

B.1 Log In Page ................................................................. 106
B.2 Forgot Password Page and Reset Password ......................... 107
B.3 Sidemenu ................................................................. 108
B.4 Feed Page ................................................................... 109
B.5 Betting Page .................................................................. 110
B.6 Club Settings Page ......................................................... 111
B.7 Profile Page .................................................................. 112
B.8 Members Page .............................................................. 113
  B.8.1 Add Member ............................................................. 114
  B.8.2 Edit Member ............................................................. 115
  B.8.3 Delete Member ......................................................... 116

C Transitions .................................................................. 117

C.1 Logging In .................................................................... 118
C.2 App Transitions ............................................................ 119
C.3 Web Transitions ............................................................ 121
# DOCUMENT STATUS SHEET

## GENERAL

<table>
<thead>
<tr>
<th>Document title</th>
<th>Software Requirements Document v1.0.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identification</td>
<td>SRD/1.0.1</td>
</tr>
<tr>
<td>Authors</td>
<td>K. van Eenige, K. Verhaegh, S. Chen, D. van der Laan</td>
</tr>
<tr>
<td>Document status</td>
<td>Finished document</td>
</tr>
<tr>
<td>Version</td>
<td>Date</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
</tr>
<tr>
<td>0.0.1</td>
<td>02-05-2016</td>
</tr>
<tr>
<td>0.0.2</td>
<td>09-05-2016</td>
</tr>
<tr>
<td>0.1.0</td>
<td>12-05-2016</td>
</tr>
<tr>
<td>0.3.0</td>
<td>18-5-2016</td>
</tr>
<tr>
<td>0.5.0</td>
<td>23-5-2016</td>
</tr>
<tr>
<td>0.8.0</td>
<td>07-6-2016</td>
</tr>
<tr>
<td>1.0.0</td>
<td>09-6-2016</td>
</tr>
<tr>
<td>1.0.1</td>
<td>15-6-2016</td>
</tr>
</tbody>
</table>
# DOCUMENT CHANGE RECORDS

## GENERAL

**Date:** 15 June 2016  
**Document Title:** Software Requirements Document  
**Document Identifier:** SRD/1.0.1

## CHANGES

<table>
<thead>
<tr>
<th>Section</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Set priorities, they were unset before</td>
</tr>
<tr>
<td>2.7.2.2</td>
<td>Removed class &quot;Matches&quot; from web class diagram</td>
</tr>
<tr>
<td>A.1</td>
<td>Updated the figure and description</td>
</tr>
</tbody>
</table>
1 INTRODUCTION

1.1 PURPOSE

This Software Requirements Document (SRD) contains the software requirements for the ClubNet system. More specifically, it provides a translation from the user requirements in Section 3 of the User Requirements Document (URD) [1] to these software requirements. The requirements in the URD[1] are specified by the client (IntuitiveTechnologies B.V.), while these software requirements are placed upon the software by The Brofessionals. In a similar sense as in the URD[1], the requirements in this document will state what the system will do and not how it will do so. In order to achieve a lower level overview than the URD[1], this document will also provide models which provide a simplified view of the software.

1.2 SCOPE

ClubNet is a software system containing the ClubNet mobile application and a web interface. The ClubNet mobile application is designed for smartphones and tablets, and the web interface is designed for all modern web browsers. The entire ClubNet software system is conceived by IntuitiveTechnologies B.V. and developed by The Brofessionals.

The purpose of the ClubNet system is to assist coaches and PR managers of amateur football clubs in organizing team and club related activities in an efficient manner. The ClubNet application provides a controlled communication mechanism to help coaches arrange activities while the web interface will be used only by PR managers to manage the activities happening in the club. Even though the current scope is only about football, the ClubNet system can be extended to be used for other sports in the future.
### 1.3 DEFINITIONS AND ABBREVIATIONS

#### 1.3.1 DEFINITIONS

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Android</td>
<td>A mobile operating system mainly developed by Google</td>
</tr>
<tr>
<td>AngularJS</td>
<td>An open-source web application framework mainly maintained by Google [6]</td>
</tr>
<tr>
<td>Bet</td>
<td>A prediction of the outcome of a match</td>
</tr>
<tr>
<td>Betting pool</td>
<td>A betting competition on matches during a football season</td>
</tr>
<tr>
<td>Brofessionals</td>
<td>The development team of ClubNet</td>
</tr>
<tr>
<td>CoachAssist</td>
<td>An independent software system developed by Intuitive Technologies B.V. [5]</td>
</tr>
<tr>
<td>Color scheme</td>
<td>A set of three colors that represent the club and are used for branding the app.</td>
</tr>
<tr>
<td>Cordova</td>
<td>A framework for creating cross-platform mobile apps</td>
</tr>
<tr>
<td>Exercise poll</td>
<td>A type of feed item in which users give their preference out of some given choices</td>
</tr>
<tr>
<td>External Sponsor</td>
<td>A sponsor who does not have a user account of the club</td>
</tr>
<tr>
<td>Feed</td>
<td>An overview that is visible to a user, containing all the feed items subscribed by that user</td>
</tr>
<tr>
<td>Feed item</td>
<td>An item containing all information about a specific activity</td>
</tr>
<tr>
<td>Form</td>
<td>A type of feed item in which users can indicate whether they satisfy some target</td>
</tr>
<tr>
<td>Intuitive Technologies B.V.</td>
<td>A software engineering company situated in the Netherlands serving the role of client</td>
</tr>
<tr>
<td>Ionic</td>
<td>A complete open-source SDK for hybrid mobile app development [7]</td>
</tr>
<tr>
<td>iOS</td>
<td>A mobile operating system developed by Apple Inc.</td>
</tr>
<tr>
<td>Meteor</td>
<td>A full-stack JavaScript solution for web apps written using Node.js [8]</td>
</tr>
<tr>
<td>MongoDB</td>
<td>A cross-platform NoSQL database [9]</td>
</tr>
<tr>
<td>Season</td>
<td>A period of the year in which matches are played</td>
</tr>
<tr>
<td>Sticky</td>
<td>Marking a club feed item so it stays at the top of the start club feed</td>
</tr>
</tbody>
</table>

#### 1.3.2 ABBREVIATIONS

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PR</td>
<td>Public Relations</td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kit: a set of development tools for the creation of applications for a certain framework</td>
</tr>
<tr>
<td>URD</td>
<td>User Requirements Document</td>
</tr>
</tbody>
</table>
1.4 LIST OF REFERENCES

References


1.5 OVERVIEW

The remainder of this document consists of three chapters. Chapter 2 gives a general description of the ClubNet system including its relation to other projects, its function and purpose, the environment, some general constraints, and the model description.

Section 2.1 and 2.2 discuss the relation to current, predecessor and successor projects respectively. Section 2.3 discusses the general functionality and purpose of the ClubNet application.

Section 2.4 describes the environment in which ClubNet will operate. Section 2.5 discusses the relation of ClubNet to other existing system, CoachAssist [2]. Section 2.6 consists of the general constraints that The Brofessionals must adhere to. The last section of chapter 2, section 2.7, describes all logical models of the ClubNet systems.

Chapter 3 lists all the specific functional and non-functional requirements, with their respective priorities, divided into logical categories. Chapter 4 supplies a traceability matrix that maps the requirements in the URD[1] to the requirements stated in chapter 3 of the SRD, and vice versa.
2 GENERAL DESCRIPTION

2.1 RELATION TO CURRENT PROJECTS

The ClubNet interface is inspired by the standard set for a (news)feed by current major social applications, such as Facebook or Twitter. There are also competitors that have comparable applications, however no inspiration was taken from these applications. There might be other projects related or similar to ClubNet that we have no knowledge of.

2.2 RELATION TO PREDECESSOR AND SUCCESSOR PROJECTS

ClubNet has no predecessors and thus does not build upon any predecessor projects. The customer has implied to us that he does want to go on and build further upon ClubNet for future user requirements. Therefore, the customer may continue to change, add or remove functionality from the ClubNet application and back end server and adapt the interface as he sees fit, after the initial development by The Brofessionals.

2.3 FUNCTION AND PURPOSE

ClubNet will provide clubs with a way to improve communication within a team and within the club as a whole. Coaches can regulate all kinds of important activities that have to do with the team in the club feed, such as voting exercises, suggesting exercises or practicalities. The players of the team will then be able to see these items in the club feed. Furthermore, other club members that are not part of the team can see club feed items relevant to the whole club, such as sponsoring events, club betting and heroes.

ClubNet will mainly provide the following functionality modules:

- **Exercise voting**: Players in a team will be able to vote on one of three exercises, of which the winning exercise will be practiced in the coming training session.

- **Practicality**: Players need to be brought to far away matches or the outfits of the team need to be washed. These, and more, all fall under practicalities, which can be organized in ClubNet.

- **Sponsoring event**: A team can ask for the club to create a sponsoring event for them for a certain goal. This asks the members of the club for some contribution. Of course, the team will do something in return when the goal has been reached.

- **Club betting**: Here all users of ClubNet can vote on matches of teams of their club (except for players and coaches voting on their own match) and accumulate points by betting the right outcome. At the end of the season the user that has accumulated the most points will be awarded a (small) prize by the club. This betting will make all persons involved
with the club even more involved and enthusiastic for all the matches, even those they are not directly involved with.

- **Heroes:** There might be a hero in the club that nobody knows about, but someone wants to spread the story. ClubNet facilitates this, spreading the story throughout the whole club.

- **Exercise suggestion:** With ClubNet a coach can ask one of his players to suggest an exercise for the next training.

The purpose of ClubNet is to help the club and coaches in taking away the effort that comes with organizing and communicating within teams and clubs. Another purpose is to be a profitable venture for Intuitive Technologies B.V..

### 2.4 ENVIRONMENT

The ClubNet system consists of a mobile application and a web user interface. Each of these two components supports a different environment to run on:

- **Web user interface:** All modern web browsers (Chrome v48, Internet Explorer v11, and Safari v8 and newer).

- **Mobile application:** Android 4.1 (Jelly Bean) and higher, iOS 7.0 and higher.

For the mobile applications, it should work in portrait mode on both phones and tablets.

The system will be used by the following users:

- **End Users:** These people will receive account credentials to the app or web interface through their club. There are different types of users:
  - Coach: People who will make use of all the features of the mobile app.
  - Player: People who will use the mobile app to check what the coach and club provides for them on their feed and react on it, and respond to direct messages from the coach.
  - General club member: People who will look at what the club provides for them on their feed and react on it.
  - PR manager: Club administrators of the club feed. They oversee what is visible to the whole club in the feed through the web interface and provide all the players and general club members with accounts.

- **Service Provider:** The people (Intuitive Technologies B.V.) responsible for the maintenance and operation of the back end server. They will provide the ClubNet service to clubs which includes the app and the web interface.
Developers: These people are responsible for maintaining the application and developing it further, after the end of this project.

The details of the end users are specified in section 2.4.2 of the URD [1].

2.5 RELATION TO OTHER SYSTEMS

ClubNet is associated with CoachAssist [2]. It does not necessarily depend on CoachAssist, but the exercise voting functionality and retrieving the coach accounts will not function without the functioning APIs of CoachAssist that will be used to retrieve the coach accounts, training schedules and training exercises from the CoachAssist database. Currently no other system depends on ClubNet, but this might change in the future when functionality is added to ClubNet.

2.5.1 COACH ASSIST

CoachAssist is, as the name already suggests, an app that assists coaches. It is an intelligent app that automatically generates training sessions that are tailor-made to fit the profile of the coach, the team and the facilities available. To be able to connect ClubNet to CoachAssist [2], CoachAssist will provide coach accounts information, schedules of training sessions and their associated exercises. Each exercise consists of a description and an image, both stored in the database of CoachAssist. Each coach account consists of first name, last name, date of birth, club ID, team ID and account creation date. In ClubNet, these exercises will be voted on by players in an exercise poll as described in URD[1]section 2.2.1. The coach accounts information will be used for user logging in and user accounts management.

2.6 GENERAL CONSTRAINTS

The main constraint for ClubNet is that it should be easy and intuitive to use for all end users, especially coaches. Furthermore, the design of the system should focus on maintainability, reliability, modularity and scalability because the customer has the intention of further developing ClubNet after initial development by the Brofessionals.

2.6.1 SECURITY

ClubNet will store personal passwords for each user profile on the database. To protect this sensitive information, we hash all the passwords. This is done by MongoDB, which uses the SHA-256 hash function. Furthermore, there might be other sensitive information personally identifying the users too, like their name and email. We protect this data by limiting user profile access through secure API calls, and by limiting user access to accounts. Each user can only view his own account, and change settings for their own account. Only PR users and database administrators from Intuitive Technologies B.V. can see the list of all accounts together with
the attached personal information. Passwords are never shown to any user or administrators. They will be solely be visible to the user as stars or dots, and the administrators can only see the hash in the database.

2.6.2 EXTENSIBILITY AND ADAPTABILITY

The client prefers to use a single code base to be able to easily extend and adapt ClubNet, similarly to what has been done for the development of the CoachAssist app. To this end, we will develop the app in a modularized style. We will use Meteor with AngularJS together with Ionic to build a Web app. Ionic builds on top of Apache Cordova which takes care of packaging the web app into native apps for iOS and Android and potentially more platforms. This results in a single code base that works on all devices we are developing for as specified (in the URD [1], and in the specific requirements in Chapter 3). Furthermore, AngularJS itself places a lot of emphasis on modularization. It does so by using controllers, services, and directives. This will increase testability, extensibility, and adaptability as it allows for easier reuse of the written code.

2.6.3 USABILITY

A main focus while developing ClubNet is that ClubNet has to be user-friendly. As the app will be used by children from 6 years onward, and possibly other users that are not familiar with systems like this, the app has to be easy to understand and easy to use. Any user should be able to know how to use the system in less than one hour. To achieve this level of user-friendliness the app will have simple user interfaces with obvious functionality. On top of that the app must not have more than three levels of nested screens.
2.7 MODEL DESCRIPTION

This section contains several important models that are needed for a complete description of the high-level design of the system. These include domain model, class diagrams, model-view-controller model, ER model and sequence diagrams.

2.7.1 ENVIRONMENT MODEL

The ClubNet system consists of a back-end server and a front-end. It also interacts with the external CoachAssist database to support certain functionality.

![Environment Model Diagram]

**FIGURE 1: ENVIRONMENT MODEL**

The front-end has two separate components, namely the ClubNet mobile app and the web user interface. As described in section 2.4 in the URD[1], the web user interface will be used exclusively by a PR user and the mobile app will be used by all other users. The Ionic framework is used for the mobile app front-end in order to create an interface that supports different mo-
bile environments and functionality. To meet the security constraint, the front-end does not have any direct access to the back-end database. Instead, it sends requests to the back-end server to access and modify necessary information stored in the database. The front-end only implements functionality that is necessary for the visual interactions with users of the ClubNet application.

The back-end server has two components, namely the back-end logic and the back-end database. The Meteor framework is used to achieve responsiveness of the back-end server. The back-end database stores all necessary information that is needed for the full functionality of the ClubNet system. It is a stand-alone database without any dependency on an external database. The back-end logic implements all server side functionality that will be used by the front-end. It serves as an extra control layer between the front-end and the back-end database. In details, the back-end server will listen to front-end requests, process the requests and provide corresponding responses. Furthermore, the back-end server can notify the front-end for new database updates if needed.

The back-end logic interacts with CoachAssist through the back-end server of CoachAssist. Similar to ClubNet, the CoachAssist back-end server consists of a back-end logic and back-end database. The back-end logic of ClubNet sends requests to the back-end logic of CoachAssist to retrieve the required data. The data that needs to be retrieved from the CoachAssist is related to the functionality of the voting exercises. The CoachAssist back-end server is the only external element in the environment model.
2.7.2 CLASS DIAGRAMS

The ClubNet system consists of two parts: the ClubNet app and web interface. These two parts share the same back-end server but they still have exclusive functionality. Thus we model these two parts separately.

2.7.2.1 CLASS DIAGRAM OF THE APP

Each class will be described with its attributes and functions. Along with the attributes and functions, a description of the relations between the classes will also be provided.

- **FeedItem**
  FeedItem is an abstract class that is implemented by the different types of feed item. Each different item brings some functionalities, and different attributes. However there is some functionality and a few attributes that all feed items have.
  This class has the following attributes:
- **id**: The unique identifier of a feed item.
- **title**: A string that holds the title of a feed item.
- **description**: A string that holds the description of a feed item.
- **creationDate**: A date object that holds the date on which the feed item was created.
- **sticky**: A boolean that holds whether an item has been stickied.
- **type**: A string that holds the type of feed item, this will be one of 6: Heroes, Exercise poll, Suggestion, Betting round, Sponsoring, or Form.

This class provides the following functionality:

- **showInfo()**: A function that displays the information of a feed item.
- **updateInfo(String[] info)**: A function that updates the information of the feed item. It is called after the information of a feed item is edited by a user.
  * **info**: A string array that holds the information to update.

- **Heroes**
  Heroes is a class that represents a heroes feed item. It is a specification of class FeedItem.

  This class has the following attribute:

  - **image**: A string that holds the URL to the image that can be added to a heroes feed item.

- **ExercisePoll**
  ExercisePoll is a class that represents an exercise voting feed item. It is a specification of the FeedItem.

  This class has the following attributes:

  - **deadline**: A date object that holds the deadline of a Exercise poll.
  - **intPublic**: A boolean that holds whether the intermediate results are visible to the public.
  - **finalPublic**: A boolean that holds whether the final results are visible to the public.
  - **nrVotes**: An integer that holds the total number of votes accumulated on an Exercise poll so far.
  - **ended**: A boolean representing whether the voting has been closed.
  - **nrVoters**: An integer that holds the total number of voters who can vote on the poll.
  - **results**: An associative array that stores the accumulated votes on each exercise.

This class provides the following functionality:
- **newVote(int id)**: A function that registers a new vote on an exercise.
  * **id**: The unique identifier of the exercise which is voted on.
- **getWinner()**: A function that returns the id of the winning exercise of the vote.
- **getExercises()**: A function that returns the ids of all candidate exercises.
- **getResults()**: A function that returns the (intermediate and final) results.

- **Exercise**
  Exercise is a class that represents an exercise. It contains information of a certain exercise.

  **This class has the following attributes:**
  
  - **name**: A string that holds the name of an exercise.
  - **icon**: A string that holds the URL to the icon of an exercise.

- **TrainingSchedule**
  TrainingSchedule is a class that represents a scheduled training. It has a set of three candidate exercises that need to be voted on.

  **This class has the following attribute:**
  
  - **time**: A date object that holds the time on which the training schedule is.

  **This class provides the following functionality:**
  
  - **addExercise(Exercise[] exs)**: A function that adds a set of exercises to the training schedule.
    * **exs**: An array of Exercise objects to be added.

- **Suggestion**
  Suggestion is a class that represents an exercise suggestion feed item. It is a specification of the FeedItem.

  **This class has the following attributes:**
  
  - **deadline**: A date object that holds the deadline of a suggestion feed item.
  - **suggestion**: An object that holds the suggestion of a player. This can include URL's to videos or pictures of the suggested exercise.
  - **playerID**: The unique identifier of the player that can respond on this suggestion item.

  **This class provides the following functionality:**
  
  - **setSuggestion(String suggestion)**: A function that sets the suggestion provided by the player.
• **Sponsoring**

Sponsoring is a class that represents a sponsoring feed item. It is a specification of the FeedItem. **This class has the following attributes:**

  - `targetAmount`: A double that represents the amount of money needed for the sponsoring event.
  - `raisedAmount`: A double that represents the amount of money accumulated so far for the sponsoring event.
  - `deadline`: A date object that holds the date/time at which the sponsoring event will end. After this date and time users can no longer respond to the sponsoring feed item.
  - `sponsorships`: An associative array that stores the ids of the sponsors and the amount they sponsored.

**This class provides the following functionality:**

  - `addSponsorship(int amount, int userID)`: A function that registers a new sponsorship from a user. It updates the attributes of raisedAmount and sponsorships.
    * `amount`: The amount to be sponsored by the user.
    * `userID`: The unique identifier of the user that is sponsoring the event.
  - `checkEnd()`: A function to check if the sponsoring has ended.

• **PracticalityForm**

PracticalityForm is a class that represents the practicality feed item. It is a specification of the FeedItem. This form can be for laundry, driving, absence, or a customized target. **This class has the following attributes:**

  - `repeatInterval`: A string that holds how often a form should be repeated. It can take the values of either daily, weekly, or monthly.
  - `target`: A string that holds the target to be reached by the form. It can take values of either driving, laundry, absence, or a customized target.
  - `targetValue`: An integer that holds the value of the target to be reached. For instance the number of seats for a driving form, or simply 1 player does the laundry for a laundry form. It takes the value of 0 for the absence form.
  - `contributions`: An associative array that stores the contributions from users. It stores values of 0 for the absence form.
- **achievedValue**: The value reached so far. For instance it can stores how many seats have been allocated so far.
- **locked**: A boolean that holds whether the form is locked, a form will be locked when the targetValue has been reached.

This class provides the following functionality:

- **addContribution(int userID, int amount)**: A function that registers a new contribution from an user.
  
  * **userID**: The unique identifier of the user that contributes.
  * **amount**: The amount the user will contribute, e.g. the amount of seats. It takes the value of 0 for absence form.

- **withdraw(int userID)**: A function that withdraws a contribution from an user.
  
  * **userID**: The unique identifier of the user that is withdrawing his/her response.

- **BettingRound**
  
  BettingRound is a class that represents the betting round feed items. This is a specification of the FeedItem. A betting round exists in a season, and consists of matches. Each round has 3 to 5 matches and happens in 1 season.

  This class has the following attributes:

  - **deadline**: A date object that holds the deadline of a betting round.
  - **matches**: A two-dimensional array that holds the matches in the betting round.
  - **results**: An associative array that holds the results of the matches.

  This class provides the following function:

  - **getResults()**: A function that returns the results of all matches in the betting round.

- **BettingPool**
  
  BettingPool is a class that represents the betting pool of a season. A betting pool consists of weekly betting rounds and has a leaderboard that keeps track of the points earned by each user.

  This class has the following attribute:

  - **season**: A string holds the season that the betting round is in.

  This class provides the following functionality:

  - **getLeaderBoard(int limit)**: A function that returns the leader board of the betting pool with the specified limit.
**limit**: A number that specifies how many users of the leaderboard are returned in order of most points.

- **User**
  
  User is an abstract class that represents all the basic functionality and attributes of any user account. It gets specified by the GeneralMember, Coach, and Player class.

  **This class has the following attributes:**
  
  - **type**: A string that holds whether this account is a general member, coach, or player
  - **userID**: The unique identifier of a user account
  - **userName**: A string that holds the name of the user
  - **email**: A string that holds the email of the user
  - **password**: An encrypted string that holds the password of the user
  - **clubID**: The unique identifier of the club this account belongs to
  - **bettingScores**: An associative array that stores scores of all betting pools.

  **This class provides the following functionality:**
  
  - **changePassword(String pw)**: A function that changes the password of a user.
    
    * **pw**: A string that holds the new password to be used by the user.
  - **changeUsername(String name)**: A function that changes the user name of an user.
    
    * **name**: A string that holds the new name to be used by the user.
  - **resetPassword()**: A function that resets the password for a user, an email will be sent with the steps to follow.
  - **respondTo(FeedItem item, String value)**: A function that registers a response from the user to a feed item.
    
    * **item**: The feed item to which the user respond.
    * **value**: The value of the response.
  - **login(String email, String password)**: A function that logs a user in, if their email and password match the database.
    
    * **email**: The email filled in by the user.
    * **password**: The password filled in by the user.
  - **logout()**: A function that logs a user out
  - **subscribe(String[] types)**: A function that subscribes a UserAccount to a set of particular item types. When users are subscribed they get a notification when a feed item of that type is created.
• types: An array that holds the types of the feed items the user subscribes to.

- `changeLanguage(String lang)`: A function that changes the language preference of a user.
  
  * `lang`: The new language to use inside of the application.

- `receiveNotification(String[] nots)`: A function that receives a push notification.
  
  * `nots`: A string array that holds the information in a notification.

- `updateInfo(String[] info)`: A function that updates the information of the user. It is called after the information of a user is edited.
  
  * `info`: A string array that holds the information to update.

- `withdrawResponse(FeedItem item)`: A function that withdraws a response from a feed item which the user responded to.
  
  * `item`: The feed item from which the response will be withdrawed.

• Coach

Coach is a class that represents the coach account. This is a specification of the UserAccount class.

This class has the following attribute:

- `teamID`: The unique identifier of the team this coach coaches.

This class provides the following functionality:

- `addNote(int itemID, String note)`: A function that allows the coach to add a new note on a feed item.
  
  * `note`: A string that holds the content of the note.

- `createItem(String[] info)`: A function that creates a new feed item with specified the information.
  
  * `info`: A string array that holds the information of a feed item.

- `editNote(Note note, String text)`: A function that changes the content of a note.
  
  * `note`: The note whose content needs to be changed.
  
  * `text`: The content of the new note.

- `retract(FeedItem item)`: A function that retrieves a new feed item. The retrieved feed item will no longer be stored in the database.
  
  * `item`: The item that needs to be retrieved.

- `sticky(FeedItem item)`: A function that stickies a feed item. The stickied feed item will appear at the top of the user feed.
• **GeneralMember**
  GeneralMember is a class that represents the general member account. This is a specification of the UserAccount class. As this account cannot do more than the basic functionality that all accounts can, it has no own attributes or functions.

• **Player**
  Player is a class that represents the player account. This is a specification of the User class.
  This class has the following attribute:
  - `teamID`: The unique identifier of the team this player (account) belongs to.

• **Note**
  Note is a class that represents a note that can be put on a feed item. A note can only be created, seen, edited, and deleted by coaches. A note is associated with one coach (the one that created it) and one FeedItem instance. Each FeedItem can only have one note from one coach, however multiple coaches can have notes on the same feed item.
  This class has the following attribute:
  - `text`: A string that holds the text of the note.

• **Chatable**
  Chatable is an interface that has chat related functionality.
  This interface has the following functionality:
  - `chatWith(int id)`: A function that enables a user to chat with another user.
    * `id`: The unique identifier of the user with which a chat session is initiated.
2.7.2.2 CLASS DIAGRAM OF THE WEB INTERFACE

For the web interface the class diagram is considerably smaller. Since a PR user is the only user that will use this part of the ClubNet system this is the only user that we modeled (without a super class). As the PR user only sees the heroes, sponsoring, and betting round items we left out the other three feed items since they are irrelevant for the web interface.

![Class Diagram](image)

**FIGURE 3: WEB CLASS DIAGRAM**

Each class will be described with its attributes and functions. Along with the attributes and functions, a description of the relations between the classes will also be provided, if they are not trivial.

- **PRUser**
  PRUser is a class that represents the PR user and handles the functionality of what a PR user should be able to do. It has a small amount of attributes, only the personal information of the PR user. The functions are representations of the requirements in the URD[1].

  **This class has the following attributes:**

  - **clubID:** An integer that holds the unique identifier of the club of a PR user.
  - **email:** A string that holds the email of the PR user.
  - **password:** A (encrypted) string that holds the password of the PR user.
  - **userID:** An integer that holds the unique identifier of the PR users’ account.
userName: A string that holds the name of the PR user, to be shown in the web interface.

This class provides the following functionality:

- **addSponsor**(*Sponsoring item, String name, double amount*): A function that lets the PR user add an external sponsor to a sponsorship event.
  * *item*: The sponsoring instance to which the sponsor should be added.
  * *name*: The name of the sponsor to be added.
  * *amount*: The amount of money the sponsor wishes to sponsor.

- **addUser**(*String[] info*): A function that lets the PR user add a new user to the database.
  * *info*: An array of strings holding all the required information to add a user to the database.

- **approveHeroes**(*FeedItem item*): A function that lets the PR user approve a heroes feed item.
  * *item*: The feed item to be approved.

- **createBettingPool**(*String[] info*): A function that lets the PR user create a betting pool object.
  * *info*: An array of strings containing all the information for the betting pool.

- **createHeroes**(*String[] info*): A function that lets the PR user create a heroes feed item.
  * *info*: An array of strings containing all the information for a heroes feed item.

- **changePassword**(*String pw*): A function that lets the PR user change his password.
  * *pw*: The new password to be used by the PR user.

- **changeEmail**(*String email*): A function that lets the PR user change his email.
  * *email*: The new email to be used by the PR user.

- **denyHeroes**(*FeedItem item*): A function that lets the PR user deny a heroes feed item.
  * *item*: The heroes item to be denied.

- **deleteUser**(*int id*): A function that lets the PR user delete a user account.
  * *id*: The unique identifier of the user account to be deleted.

- **login**(*String username, String password*): A function that logs in the PR user, given the username and password are correct.
  * *username*: The username of the PR user (email).
  * *password*: The password of the PR user.

- **logout**(): A function that logs out the PR user.
- `retractFeedItem(FeedItem item)`: A function that lets the PR user retract a specific feed item.
  
  * `item`: The to be retracted feed item.

- `resetPassword()`: A function that lets the PR user reset his password.

- `setClubLogo(String pic)`: A function that lets the PR user add a club logo to the database.
  
  * `pic`: A URL of the image (club logo) to be added to the database.

- `setClubName(String name)`: A function that lets the PR user set the club name.
  
  * `name`: The clubname to be set.

- `setColorScheme(String[] colors)`: A function that lets the PR user set three club specific colors to customize the web interface and app.
  
  * `colors`: An array of strings containing three strings, of which each is a colorcode.

- `setHeroesLimit(int limit)`: A function that lets the PR user set a limit to the number of heroes items a club can post per month.
  
  * `limit`: The limit to be set.

- `sticky(FeedItem item)`: A function that lets the PR user sticky a feed item, so that it appears at the top of the feed.
  
  * `item`: The feed item to be stickied.

- `viewMatches(String season)`: A function that lets the PR user view all the matches of a specific season.
  
  * `season`: The season of which all matches should be shown.

- `viewSponsorships(Sponsoring item)`: A function that lets the PR user view all the sponsorships of a sponsoring event.
  
  * `item`: The sponsoring item of which all the sponsorships should be shown.

**Feed**

Feed is a class that represents the feed consisting of feed items. The feed can be filtered and updated and is specific per PR user as each user sees items from their associated club or uses different filters.

This class has the following attributes:

- `visibleTypes`: A boolean array that holds true or false for each item type the PR user can view.

This class provides the following functionality:

- `filter()`: A function that filters the feed based on the array `visibleTypes`.

- `updateFeed()`: A function that updates the feed by requesting new data from the database.
• **FeedItem**

FeedItem is an abstract class that represents the items that are connected and are in the feed. All the feed items that are modelled here can be manipulated by the PR User through certain functions elaborated on earlier in this chapter. This FeedItem inherits the same constraints from the Feed, thus specific per PR user as each user sees items from its associated club.

This class has the following attributes:

- *creationDate*: A Date object that holds the date on which the item was created.
- *description*: A String that holds the description of the FeedItem.
- *id*: An integer that holds the id of the FeedItem, used to identify between different FeedItem’s.
- *sticky*: A Boolean that holds true or false, true means that the FeedItem is a sticky, false means it is not sticky.
- *title*: A String that holds the title of the FeedItem.
- *type*: A String that holds the type of the FeedItem, used to determine the design of the FeedItem.

This class provides the following functionality:

- *showInfo()*: A function that shows all the information that is available for the FeedItem.
- *updateInfo(String[] info)*: A function that updates the FeedItem with the information given inside the String array.
  
  * info: A string array that holds the information to be updated.

• **Heroes**

Heroes is one of the FeedItem types that the PR user has control over. The feed items of this type need to be approved or dissapproved by the PR user.

This class has the following attributes:

- *image*: A String that holds the url of the image used for the Heroes feed item.

• **Sponsoring**

Sponsoring is one of the FeedItem types that the PR user has control over. The PR user can add a sponsor to this feed item outside of the web app.

This class has the following attributes:

- *deadline*: A Date object that holds the deadline for the sponsoring event. The sponsoring event ends when the deadline is reached.
- **raisedAmount**: A double that holds the current total raised amount of money amongst all sponsors.
- **sponsorships**: A double associative named array that holds all individual contributions to the sponsoring, the index is the userID, with the amount in the array.
- **targetAmount**: A double that holds the amount of money that the maker of the sponsoring wants to reach before the deadline.

This class provides the following functionality:

- **checkEnd()**: A function that returns a boolean true or false. It returns true when the deadline has been reached, false otherwise.
- **raise(int userID, double amount)**: A function that raises the current raisedAmount with the amount and adds the amount to the sponsorships array.
  * **userID**: The userID of the user raising the sponsoring.
  * **amount**: The amount of money that the user wants to contribute.

• **BettingPool**

BettingPool is a class that represents the collection of BettingRounds of a season. The BettingPool is set up by the PR user for one whole season. A BettingPool consists of weekly betting rounds and has a leaderboard that keeps track of the points earned by each user.

This class has the following attributes:

- **season**: A string that holds the season that the betting round is in.

This class provides the following functionality:

- **getLeaderBoard(int limit)**: A function that returns the leader board of the betting pool with the specified limit.
  * **limit**: A number that specifies how many users of the leaderboard are returned in order of most points.

• **BettingRound**

BettingRound is a class that represents the betting round feed items. This is a specification of the FeedItem class. A betting round exists in a season, and consists of matches. Each round has 3 to 5 matches and happens in 1 season.

This class has the following attributes:

- **deadline**: A Date object that holds the deadline of a betting round.
- **matches**: A two-dimensionsal array that holds the matches in the betting round.
- **results**: An associative array that holds the results of the matches.
This class provides the following functionality:

- `getResults()`: A function that returns the results of all matches in the betting round.
2.7.3 **MODEL-VIEW-CONTROLLER MODEL**

Because ClubNet is build with a combination of the following JavaScript platforms; Meteor, AngularJS and Ionic, we must follow a model-view-controller model architecture, as seen in figure 4. From top to bottom, these are modelled as follows:

- **User**: The user; making use of the controller to influence the app and to whom the view is displayed.

- **View**: Implemented in HTML and CSS, which will be displayed to the user with the information given by the model.

- **Model**: Holds the information that can change and needs to be displayed to the user.

- **Controller**: The AngularJS JavaScript files, which will make use of directives, services and controllers to manipulate the data in the model.

- **Meteor Client**: Used by the whole front end, to exchange data with the back end.

- **Meteor Server**: For back end side calculations and updates.

- **MongoDB**: Stores data that needs to be shared between all users.

---

**FIGURE 4: MODEL-VIEW-CONTROLLER**
2.7.4 DATA MODEL

The data model is the Entity Relationship diagram of our database. We use MongoDB as it operates well with Meteor. The client utilizes the same database for the CoachAssist application and has expressly indicated its usage for ClubNet as well. Given the client has sufficient knowledge to operate a MongoDB database, it means maintaining it will be easy and hassle free.

The ER diagram of the database is structured as seen below in Figure 5. It uses the Crow's Foot notation, i.e. the entities are shown as boxes and the relations as lines. The cardinality is shown as different types of feet at the end of the lines, see also the legend.

![Figure 5: The ER-Diagram](image)

Legend:
- **Exactly One**
- **Zero or One**
- **Zero, One, or More**
- **One or More**
• Feed Item

Each feed item corresponds to a specific activity and contains all needed information for that activity. The data we will store is as follows:

- **itemID**: Used to identify each feed item.
- **sticky**: Used to determine whether or not a feed item has been "stickied".
- **lastModifiedDate**: Date of last modification of the feed item, used for sorting.

• FeedItemType

This stores all possible feed item types together with their icon that will be shown on the feed.

- **type**: A specific type of feed item.
- **label**: The label to be shown in the front-end to represent a specific item type.
- **icon**: An icon belonging to a specific type of feed item.

• Exercise poll

This is a specific feed item that represents the Exercise polls for exercises. It is associated with 3 exercises on which the players can vote. The attributes are as follows:

- **title**: This the title of the feed item.
- **deadline**: The deadline of the poll, when this has been reached voting is no longer possible.
- **results**: Results of the poll, so how many votes each exercise has accumulated, stored as an integer array containing three integers.
- **intermediatePublic**: This represents whether or not the intermediate results are visible to the public.
- **finalPublic**: This represents whether or not the final results are visible to the public.
- **nrVotes**: The amount of votes (so far) on a particular poll.
- **nrVoters**: The amount of unique voters on a particular poll.
- **ended**: Whether or not the poll has ended, if this is true the poll should be closed.

• Exercise

This is an exercise, which is used in the exercise Exercise poll described above. There are two attributes:

- **exerciseID**: A unique identifier for each exercise.
- **image**: A URL to an image of the exercise.
- name: The name of the exercise.

- **Suggestion**
  
  This represents a suggestion feed item that is shown to specific user and coach it is connected to. It only has one attribute except for the player and coach it is connected to:
  
  - **deadline**: The date and time at which the suggestion will close if the player did not respond yet.

- **Form**
  
  This represents a form for any practicality, like driving, laundry, and absence lists. There are quite some attributes we save, as the form has to be kept for administrative purposes.
  
  - **title**: The title of the form, can be chosen by the coach creating the form.
  
  - **repeatInterval**: A string that holds with which interval a form should be repeated. It can take the values of either daily, weekly, or monthly.
  
  - **target**: Any one of laundry, driving, and absence. Can also be customized to have any other form, so this just takes a text value.
  
  - **targetValue**: The amount of seats needed to drive somewhere, or any other target value that needs to be reached.
  
  - **achievedValue**: The target value reached so far.
  
  - **locked**: Whether or not the form is already locked; a form is locked when its target has been reached.

- **Sponsor Event**
  
  This entity represents a sponsoring event for a team. It has some attributes, and is associated with ExternalSponsor, which will be described after this. Each of the attributes are specified by the coach when he creates a sponsoring feed item.
  
  - **title**: The title of the sponsor event.
  
  - **description**: The description of the sponsor event, including what the club wants to do and what they will do in return.
  
  - **targetAmount**: The target amount for the sponsoring.
  
  - **raisedAmount**: The raised amount for the sponsoring so far.
  
  - **deadline**: The deadline for the sponsoring, after this date no more money can be collected or sponsored.
• **External Sponsor**

This entity is solely here so that we can represent external sponsors that do not have a ClubNet account but do want to sponsor the club. These sponsors will likely be local companies or wealthy people. To possibly make a feed item about this sponsor (as a thank you) we will store two attributes:

- **sponsorName**: The name of the person or company that wants to sponsor the club with this sponsoring event.
- **amount**: The amount the person or company wants to sponsor.

• **Heroes**

The Heroes entity saves a heroes story, written by a coach or by a PR manager. For this reason we simply store the following attributes:

- **title**: The title of the heroes story.
- **description**: The story itself.
- **image**: An URL to the image that can be added to the story.

• **Betting Round**

This represents a betting round in which players can bet on matches during a season. Therefore we have separate match and season entities too. Thanks to these entities the betting round only stores one attribute:

- **deadline**: The deadline of this specific betting round, consisting out of 3 to 5 matches, it will be specified by the PR user.

• **Match**

A club betting round consists of 3 to 5 matches. A match entity holds a few attributes:

- **team1ID**: The home-playing team.
- **team2ID**: The away-playing team.
- **result**: The final result of the match.

• **Season**

A season is the time span in which the teams will play matches. Normally this is a year, or two separate half years (which are then two seasons).

- **year**: The year of this season.
- **quarter**: The quarter of this season.
• **Betting Result**

The betting result holds the result for each user that made a bet on the betting round described above.

- *points*: The points a user has accumulated so far.

• **User**

A user is the account of a user of the application. It is split up in Player, Coach, PR Manager, and General Club Member. Player and coach have special roles within the application. For more information on the users see the URD[1]. A user has the following properties:

- *userID*: A unique ID to identify the user.
- *username*: The users username (which will be their email address).
- *password*: The users password, which will be protected.

• **Access Control**

Access control stores the access matrix. For each user type it specifies what they can and cannot do in the application.

- *userType*: The type of the user, these are the roles used in the matrix.
- *accessValue*: Whether or not a role can access certain things.

• **Club**

The club entity represents the club as a whole, and stores some attributes that are used for personalizing the application for each club.

- *clubID*: A unique ID to identify each club.
- *clubName*: The club name of the club that uses the application.
- *colorScheme*: A colorscheme chosen by the club, used for personalizing the application.
- *logo*: The logo of the club, also used for personalizing the application.

• **Team**

The team in which players play and which a coach coaches. This entity is thus connected with the player and coach entities.

- *teamID*: A unique ID to identify each teams, used to check if players or coaches are betting on their own team in a betting round.
- *teamName*: The name of the team, which will be displayed in the betting rounds.
• **Note**

  This entity represents the notes a coach can make on any feed item. It’s unique identifier will be the itemID of the feed item it is attached to.

  - **text**: This attribute stores the text written on the note.

• **Response**

  This entity models all responses from users on feed items, for instances the Exercise polls or the betting rounds. Therefore, we store a value that is parsed in different ways depending on which feed item it was a response to.

  - **value**: The response value.

• **Chat**

  The chat entity represents a chat session between two users and stores the chat history.
2.7.5 MESSAGE SEQUENCE DIAGRAMS

Signing in to ClubNet:
When a user first opens ClubNet, the user is prompted to sign in to ClubNet. This is done by using the username and password that are given to the user by the PR manager. This username is the e-mail address of the user and the password is automatically generated. The user fills in the username and password in the corresponding fields of the ClubNet interface. Then the user presses the Log In button to send this data to the back end, which will propagate it to the database. When it has arrived at the database, a validation will be done on the combination of the username and password. If this combination is incorrect, the database will send an error back. This error will then be displayed to the user. If this combination is correct, the database will confirm it and send back the newest items that can then be displayed to the user.

To do anything in ClubNet a user needs to be signed in, so this is a precondition to all other sequence diagrams.

FIGURE 6: SIGNING IN
**Adding a user account:**

To add a user account to ClubNet, a PR user must first follow the sign in sequence to sign in to the ClubNet web interface. When the PR user is signed in to the web interface, he clicks the account management option. This will prompt the interface to get a list of the current users from the database. When this list is returned, it is shown to the PR user in a separate interface. In this new interface, the PR user can press the add user button, which will show an add user dialog. In this new dialog, the PR user must enter the email address and the full name of the user account that needs to be added. When this is correct, the PR user confirms it. Then this personal information will be sent to the database to be stored. When this is done, it is confirmed and an e-mail is sent to the newly created user with its personal information and generated password.

---

**FIGURE 7: ADDING A USER ACCOUNT**
Changing password:
For a user to change the password, the user must first follow the sign in sequence to sign in to
the ClubNet app. When the user is signed in, he can go to his profile page. This prompts the app
to get the profile of the user from the database. When the database gets the request, it sends
back the user profile and finally it will be displayed to the user. The user can then press the
change password button to open a dialog. In this dialog, the user fills in his new preferred pass-
word, and when this adheres to the standards we set, can click save. When save is pressed, the
new password is sent to the database to be stored. Finally, when the new password is stored,
it is confirmed and the window closes.

FIGURE 8: CHANGING PASSWORD
Changing email address:
The procedure for changing of the e-mail address is practically the same as changing the password. The only difference is, instead of filling in his new preferred password, the user fills in his new preferred email address.

FIGURE 9: CHANGING E-MAIL
Creating a feed item:
For a coach user to create a feed item, the coach user must first follow the sign in sequence to sign in to the ClubNet app. When the coach user is signed in, he can press the create feed item button. This will display a list of item types that the user can create. The coach user can press on one of these items to pop up the details that need to be provided to post that item. When these details have been provided, the coach user can confirm to post the item. Then the item is sent to the database to be able to be spread to all other users. Going back from the database, a notification will be send around to all users that are subscribed to the selected feed item type.

FIGURE 10: CREATING A FEED ITEM
Adding a note:
For a coach user to add a note to a feed item, the user must first follow the sign in sequence to sign in to the ClubNet app. When the coach user is signed in, the coach user presses a feed item to pop up the details page of that item. In this pop up, the coach user can pick the option to add a note to the feed item, which will pop up a field where the coach user can type the text for the note. After finishing filling in this field, the coach user confirm to save the note on the server. Then the note will be stored on the server. When the note is stored, the window closes and the note can be found on the feed item.

FIGURE 11: ADDING A NOTE
Editing a note:
The procedure for editing a note is practically the same as making a note. The only difference is, instead of filling in a new note, the coach user gets the option to edit the current note.

FIGURE 12: EDITING A NOTE
**Sticky a feed item:**

For a coach user to sticky a feed item, the coach user must first follow the sign in sequence to sign in to the ClubNet app. When the coach user is signed in, the coach user presses a feed item to pop up the details page of that item. In this pop up, the coach user can pick the option to sticky the feed item. If the coach user decides to pick this option, the decision is prompted to the database and finally confirmed. The stickied item is then visible in the feed.

![Diagram: Sticky a Feed Item](image)

**FIGURE 13: STICKY A FEED ITEM**
Viewing feed item history:
Viewing a feed item history is similar to viewing the feed with the exception that all other feed items other than the ones the user wishes to see are excluded by a filter. A filter with every item checked is equivalent to seeing the entire feed. For a user to view the feed item history, the user must first follow the sign in sequence to sign in to the ClubNet app. When the user is signed in, the user can press the filter button to pop up a list of feed item filters. The user can then pick any combination of these filters. The request then gets send to the database with the chosen filter, which will return a list of the items corresponding to that filter. These items are then displayed to the user in the feed.

FIGURE 14: VIEWING FEED ITEM HISTORY
Chatting:
For a user to chat with other club members, the user must first follow the sign in sequence to sign in to the ClubNet app. When the user is signed in, the user can press the chat button to send a request to the database to show all the previous chats (if any), and all other users this user can chat with. The user can then pick one of these chats to display the chat history. The user can then decide to send a message in this chat or not. If the user sends a message, the message gets stored in the database and sent to the user the message was meant for.

FIGURE 15: CHATTING
Voting on an exercise poll:
For a player user to vote on an exercise poll, the player user must first follow the sign in sequence to sign in to the ClubNet app. When the player user is signed in, the player user must first go to an exercise poll item. If there is such an item, the player user can then pick one of the three vote options of that exercise poll. When the player user picks and confirms a vote, the vote is stored on the database and is then displayed to the player user in the exercise poll feed item.

---

**FIGURE 16: VOTING ON AN EXERCISE POLL**
Responding to a practicalities form:
For a player user to respond to a practicalities form, the player user must first follow the sign in sequence to sign in to the ClubNet app. When the player user is signed in, the player user must first go to a practicalities form item. If there is such an item, the player user can then, if needed, fill in the right details for the form. Once this is done, or it was not needed, the player user can confirm the response to the form. Then the response is stored on the database and is displayed to the user back in the practicalities form feed item.

![Diagram](image)

**FIGURE 17: RESPONDING TO A PRACTICALITIES FORM**
Withdrawing an indication from a practicalities form:

For a player user to withdraw an indication to a practicalities form, the player user must first follow the sign in sequence to sign in to the ClubNet app. When the player user is signed in, the player user must first go to a practicalities form item on which the player user has already indicated a response. Then the player user can go to the details page of the practicalities form item and press withdraw. When the player user confirms this withdraw, the response is removed from the database through storing the withdraw. The player user will then see the practicalities form item as it was before responding to it.

FIGURE 18: WITHDRAWING FROM A PRACTICALITIES FORM
Scheduling club bettings:
For a PR user to schedule the club bettings, the pr user must first follow the sign in sequence to sign in to the Web Interface app. When the PR user is signed in, it can press on the schedule betting button to go to the details page of scheduling bettings. Once there, the PR user can select a scheduling option and confirm it. This will then be stored to the database.

FIGURE 19: SCHEDULING CLUB BETTINGS
Setting maximum number of heroes items per month:
For a PR user to set the maximum number of heroes items per month, the PR user must firstollow the sign in sequence to sign in to the Web Interface app. When the PR user is signed in,
the PR user can go to the rules settings page, which will then display all the possible rules the
PR user can set. Here, the PR user can select the maximum amount of heroes items per month
and set it to the preferred number. The PR user must then confirm this for it to be stored on
the database.

FIGURE 20: MAXIMUM HEROES ITEMS
Approving/Declining a heroes item:
For a PR user to approve or decline of a heroes item, the PR user must first follow the sign in sequence to sign in to the Web Interface app. When the PR user is signed in, the PR user can go to the pending items page. This will prompt a request to the database to get all the pending items, which will then be returned and displayed to the PR user. Here the PR user can decide for each item individually to approve or decline it. This decision is then stored in the database. Approved heroes items will be shown in the feeds of users, declined heroes items will not be shown.

FIGURE 21: APPROVING/DECLINING A HEROES ITEM
Creating an exercise suggestion invitation:
For a coach user to create an exercise suggestion invitation, the coach user must first follow the sign in sequence to sign in to the ClubNet app. When the coach user is signed in, the coach user can decide to create a feed item. The coach user must then choose to create an exercise suggestion item, which will prompt the database to get a list of all possible players to invite for an exercise suggestion. The coach can then pick a player, fill in the details of the exercise suggestion and send it to the database. Then the corresponding player is sent a notification for suggesting an exercise.

FIGURE 22: CREATING AN EXERCISE SUGGESTION INVITATION
**Suggest an exercise:**

For a player user to respond to a suggest exercise invitation, the player user must first follow the sign in sequence to sign in to the ClubNet app. When the player user is signed in, the player user can go to the corresponding exercise suggestion invitation in the feed. The player user can then press the item to see the details. Here, the player user can suggest an exercise and confirm it. The suggestion is then stored on the database and a notification is sent to the coach.

![Diagram](image.png)

**FIGURE 23: SUGGEST AN EXERCISE**
3 SPECIFIC REQUIREMENTS

In this chapter we state the software requirements and constraints of the product. The product will adhere to all of these requirements. To prioritize how important these requirements are, we use the MoSCoW model. The capital letters in MoSCoW stand for: [3]

M **Must have**: requirements that are fundamental to the solution. Without these the solution will be unworkable and useless. Must haves define the Minimum Usable Subset which an Agile Project Framework project guarantees to deliver.

S **Should have**: important requirements for which there is a short-term workaround. Normally classed as mandatory when more time is available, but without them the business objective will still be met.

C **Could have**: for requirements that can more easily be left out.

W **Won’t have**: for requirements that can be included in later development. Won’t Haves are excluded from plans for the current delivery.
3.1 FUNCTIONAL REQUIREMENTS

The functional requirements are grouped based on the class diagram structure. In each section there is one subsection for all attributes and several subsections for all functions. The following sections of functional requirements will have the structure of the example shown below.

Attributes:

SR 1

Attribute name: Attribute type
Attribute description.

SR 1

Specific requirements on attributes. If there is any.

Brief description of the function

SR 1

Function name with parameters
Function description.

SR 1

Parameter name: Parameter type
Parameter description.

SR1

Specific requirements on the function or parameters.

Precondition: Precondition description.
Postcondition: Postcondition description.
3.1.1 USER

Attributes:

SR 1  
*must have*

*clubID*: String
The unique identifier of a club.

SR 2  
*must have*

*email*: String
A String that stores the email of the user.

SR 3  
*must have*

*password*: String
A String that stores the password of the user.

SR 4  
*must have*

*userID*: String
The unique identifier of a user.

SR 5  
*must have*

*userName*: String
A String that stores the name of the user.

SR 6  
*could have*

*bettingScores*: int[]
An associative array of (String, integer) pairs that stores the betting results of all seasons of each user.

Changing displaying language

SR 7  
*should have*

*changeLanguage*(String lang)
A function that changes the displaying language of the ClubNet app.

SR 8  
*should have*

*lang*: String
A String that stores the ISO 639-1 code that specifies the language to display in the ClubNet app.

**Precondition**: None.

**Postcondition**: The displaying language is changed to the one specified by *lang*.

Changing password

SR 9  
*should have*

*changePassword*(String pw)
A function that changes the password of the currently logged in user.
**SR 10**  
*should have*

**pw: String**  
A String that stores the new password.

**Precondition:** None.

**Postcondition:** The password is updated in the database to the one specified by attribute *pw*.

---

**Changing user name**

**SR 11**  
*should have*

**changeUsername(String name)**

A function that changes the user name of the logged in user.

**SR 12**  
*should have*

**name: String**

A String that stores the new user name.

**Precondition:** None.

**Postcondition:** The user name is updated in the database and user interface.

---

**Responding to a feed item**

**SR 13**  
*must have*

**respondTo(FeedItem item, String value)**

A function that registers a new response from a user to a feed item.

**SR 14**  
*must have*

**item: FeedItem**

A FeedItem object that represents the feed item to respond.

**SR 15**  
*must have*

**value: String**

A String that stores the value of the response.

**SR 16**  
*must have*

To respond to specific feed item, *value* must follow the following formats:

- Suggestion: A String that stores plain text that can also contain URLs.
- Sponsoring: A String that holds a number.
- PracticalityForm: A String that holds a number.
- ExercisePoll: A String that holds the ID of an exercise.
- BettingRound: A String that holds the results of all matches.

**Precondition:** The user selected a feed item to respond to.

**Postcondition:** The response to the feed item is stored in the database.
Logging in

SR 17

logIn(String email, String password)

A function that lets the user log in to ClubNet.

SR 18

email: String

A String that stores the email of the user.

SR 19

password: String

A String that stores the encrypted password using SHA-2 standard.

Precondition: None.

Postcondition: If the credentials match, the user is logged in to ClubNet. Otherwise a warning message is displayed.

Logging out

SR 20

logOut()

A function that let the user log out from ClubNet.

Precondition: The user is logged in.

Postcondition: The user is logged out.

Receiving notifications

SR 21

receiveNotification(String[] nots)

A function that receives a notification from the back-end server.

SR 22

nots: String[]

The notification to be received.

Precondition: A new feed item to which the user subscribed to is created.

Postcondition: The notification is received in the front-end.

Resetting password

SR 23

resetPassword()

A function that resets the password of the logged in user.

Precondition: None.

Postcondition: An email with the new password is sent to the email address of the user.
Subscribing to feed item types

SR 24 should have subscribeTo(String[] types)
A function that lets the user subscribe to a list of feed item types.

SR 25 should have types: String[]
The list of types to subscribe to. The list should contain a subset of the following types with no repetition: {Heroes, BettingRound, Suggestion, Sponsoring, PracticalityForm, ExercisePoll}.

Precondition: The user selected a list of item types to subscribe to.
Postcondition: The feed items with types that are not selected are now shown in the feed.

Updating the user information

SR 26 should have updateInfo(String[] info)
A function that updates the user information in the database.

SR 27 should have info: String[]
The list of user attributes with the new values of the fields to be updated in the database. The list should contain only a subset of the following attributes of the user with no repetition: {username, password}.

Precondition: There is a change in the user information.
Postcondition: The changed user information is stored in the database. The front-end displays the new user information.

Withdrawing a response to a feed item.

SR 28 must have withdrawResponse(FeedItem item)
A function that withdraws the response of a user to a feed item.

SR 29 must have item: FeedItem
A FeedItem object from which the user wants to withdraw a response.

SR 30 must have
A user can only withdraw his response to a feed item of type PracticalityForm.

Precondition: A feed item is selected by the user to which he has responded.
Postcondition: The response of the user to the feed item is deleted from the database.
3.1.2 COACH

Attributes:
SR 31  
`teamID`: String  
The unique identifier of a team in the club.

Adding a note to a feed item
SR 32  
`addNote(String itemID, String note)`  
A function that lets the coach to add a note to a feed item.

SR 33  
`itemID`: String  
The unique identifier of the feed item to which the coach wants to add a note.

SR 34  
`note`: String  
A String that holds the content of the note, which must contain at least one character.

**Precondition**: The coach selected a feed item to which he wants to add a note. The coach typed at least one character in the note.

**Postcondition**: The note to be added is stored in the database and the front-end displays the feed item along with the note.

Creating a feed item
SR 35  
`createItem(String[] info)`  
A function that lets a coach or PR user create a new feed item.

SR 36  
`info`: String[]  
An associative array that stores the information needed to create a specific feed item. It must contain attributes specified in the ER diagram for a specific feed item type.

SR 37  
A coach can only create feed items of the following types: Heroes, Sponsoring, ExercisePoll, PracticalityForm and suggestion.

SR 37a  
When a new heroes feed item is created while the maximum number of heroes items is reached, the system sends a notification to the PR users.

SR 38  
A PR user can only create feed items of the following types: Sponsoring, Heroes and Betting-
Precondition: The logged in user is of type coach or PR user.
Postcondition: A new feed item is stored in the database and a notification is sent to the users who subscribed to the item type of the created feed item.

**Editing a note**

**SR 39**

*editNote(Note note, String text)*

A function that lets the coach to edit a note which is created by him.

**SR 40**

*note: Note*

A Note object represents the note which was added by the user.

**SR 41**

*text: String*

A String that holds the new note content.

**Precondition:** The logged in user is of type coach. The user selected a feed item to which he added a note.
**Postcondition:** The note of the feed item is updated.

**Retracting a feed item**

**SR 42**

*retract(FeedItem item)*

A function that lets a coach or PR user to retract a feed item.

**SR 43**

*item: FeedItem*

A FeedItem object that represents the feed item to be retracted.

**SR 44**

A coach can only retract a feed item which is created by him.

**SR 45**

A PR user can only retract a feed item of the following types: Sponsoring, Heroes and BettingRound.

**Precondition:** The logged in user is of type coach or PR user. The user has selected a feed item that he has the right to retract.
**Postcondition:** The selected feed item is deleted from the database, and does no longer show on the front-end.

**Stickying a feed item**
SR 46  must have

\textit{sticky} (FeedItem item)

A function that lets a coach or PR user to sticky a feed item.

SR 47  must have

item: FeedItem

A FeedItem object that represents the feed item to be stickied.

SR 48  must have

A coach can sticky a feed item in his team of the following types: PracticalityForm, Suggestion and ExercisePoll.

SR 49  could have

A PR user can sticky a feed item of the following types: Sponsoring, BettingRound and Heroes.

\textbf{Precondition}: The logged in user is of type coach or PR user. The user has selected a feed item that he has the right to sticky. The selected feed item is public but has not ended.

\textbf{Postcondition}: The selected feed item shows on the top of the feed of the users who can view it. The stickied feed items are sorted on the date of feed item creation.

3.1.3  \textbf{PLAYER}

Attributes:

SR 50  must have

teamID: String

The unique identifier of a team in the club.

3.1.4  \textbf{PR USER}

The requirements listed in this subsection apply to the web interface.

\textbf{Adding an external sponsor}

SR 51  could have

\textit{addExternalSponsor} (Sponsoring item, String name, Double amount)

A function that lets a PR user to add an external sponsor who is not a member of the club but wishes to sponsor.

SR 52  could have

item: Sponsoring object

A Sponsoring object that represents the sponsoring feed item.

SR 53  should have

name: String
A String that holds the name of the external sponsor.

SR 54 could have

amount: double

A double that holds the amount to be sponsored.

**Precondition:** The PR user selected a sponsoring feed item. The feed item has not ended yet.

**Postcondition:** The external sponsor is stored in the database. The raised amount is increased by the amount specified in the new sponsorship. If the raised amount equals to target amount, the sponsorship ends and the feed item is locked.

---

**Adding a new user account**

SR 55 must have

`addUser(String[] info)`

A function that lets a PR user to create a new user account.

SR 56 must have

`info: String[]`

A list of attributes that are needed to add a new user. It must contain attributes specified in the ER diagram for the specific user type with no repetition.

**Precondition:** The to be added user is of type general club member or player.

**Postcondition:** A new user account with specified information is stored in the database. A confirmation email is sent to the email of the user.

---

**Deleting a user account**

SR 57 must have

`deleteUser(String id)`

A function that lets a PR user to delete a user account.

SR 58 must have

`id: String`

The unique identifier of the user to be deleted.

**Precondition:** The PR user selected a user account to be deleted.

**Postcondition:** The selected user account is deleted from the database.

---

**Approving a heroes feed item**

SR 59 could have

`approveHeroes(Heroes item)`

A function that lets a PR user approve a heroes feed item.

SR 60 could have

`item: Heroes object`
A Heroes object that represents the heroes feed item to be approved.

**Precondition:** There is a heroes feed item to be approved.

**Postcondition:** The heroes feed item is published.

### Retracting a heroes feed item

**SR 61**

`retractHeroes(Heroes item)`

A function that lets a PR user retract a heroes feed item.

**SR 62**

*item: Heroes object*

A Heroes object that represents the heroes feed item to be retracted.

**Precondition:** There is a heroes feed item that needs approval from the PR user.

**Postcondition:** The heroes feed item is removed from the database.

### Creating a new betting pool

**SR 63**

`createBettingPool(String season, String[][] matches, int perWeek)`

A function that lets a PR user create a new betting pool for a season. It creates a betting round feed item for each week in the season with a number of exercises.

**SR 64**

*season: String*

A String that holds the season. It should follow the following format: "2016-1".

**SR 65**

*matches: String[][]*

A two-dimensional array that stores all the matches in the season.

**SR 66**

*perWeek: int*

An integer that specifies how many matches should be in a weekly betting round. It can only take integer values of 3, 4 or 5.

**SR 67**

The system automatically assigns a set of matches from `matches` to each betting round feed item.

**Precondition:** There is no betting pool created for the season.

**Postcondition:** A set of betting round feed items are created but are not published.

### Setting the club logo

**SR 68**

*should have*
setClubLogo(String pic)
A function that lets a PR user to set the logo of his club.

SR 69  

should have

pic: String
A String that holds the URL to the logo.

Precondition: None.
Postcondition: The logo file is stored in the database.

Setting the club name

SR 70  

should have

setClubName(String name)
A function that lets the PR user set the name of his club.

SR 71  

should have

name: String.
A non-empty String that holds the name of the club to update.

Precondition: None.
Postcondition: The name of the club is set in the database.

Setting the color scheme of the club

SR 72  

should have

setColorSchema(String[] colors)
A function that lets the PR user to set the color scheme of his club.

SR 73  

should have

colors: String[]
A String array that stores the colors in the color schema. Each color is represented using six-digit, three-byte hexadecimal.

Precondition: None.
Postcondition: The color schema of the club is set in the database. The front-end displays according to the color scheme.

Setting the limit of heroes feed items

SR 74  

could have

setHeroesLimit(int limit)
A function that lets the PR user set how many heroes items can be created per month.

SR 75  

could have

limit: int
The amount of heroes items that can be created per month. It can only take a positive integer
number.

Precondition: None.
Postcondition: The limit of heroes feed item is stored in the database.

Viewing the sponsorships of a sponsoring feed item
SR 76

viewSponsorships(FeedItem item)
A function that lets the PR user view the sponsorships of a sponsoring feed item.

SR 77

item: FeedItem
A FeedItem object that represents the sponsoring feed item.

Precondition: The PR user selected a sponsoring feed item.
Postcondition: The sponsorships of the sponsoring feed item are displayed.

3.1.5 FEED

Attributes:
SR 78

visibleTypes: Boolean[]
An associative array of (String, Boolean) pairs that is used to determine which types of feed items are visible.

Filtering feed items on their types
SR 79

filter()
A function that uses visibleTypes to filter the feed items displayed in the user feed on their types.

Precondition: The user selected a set of item types to display in his feed.
Postcondition: The feed displays only the feed item of the types that the user has selected in the filter.

Updating the feed
SR 80

updateFeed()
A function that retrieves feed items that can be viewed by the user from database and displays them in the user feed.

Precondition: None
Postcondition: The retrieved feed items are displayed in the user feed.
### 3.1.6 FEEDITEM

**Attributes:**

**SR 81**  
*creationDate*: Date  
A Date object that holds the timestamp of the creation of the feed item.

**SR 82**  
*description*: String  
A String that holds the description written by the maker of the feed item.

**SR 83**  
*id*: String  
The unique identifier of a feed item.

**SR 84**  
*sticky*: Boolean  
A Boolean that is either true, the item is sticky, or false, the item is not sticky.

**SR 85**  
*title*: String  
A String that holds the title of the feed item.

**SR 86**  
*type*: String  
A String that holds the type of the feed item.

#### Updating the information of a feed item

**SR 87**  
*updateInfo(String[] info)*  
A function that lets a coach or PR user update the information of a feed item in the database.

**SR 88**  
*info*: The list of feed item attributes with new values for the fields to be updated. The list should contain only the following attributes for the specific feed item types:

- **Heroes**: title, description.
- **Suggestion**: title, description.
- **Sponsoring**: targetAmount, title, description.
- **PracticalityForm**: targetValue, repeatInterval, title.
- ExercisePoll: title, finalPublic, intPublic.
- BettingRound: No attribute is allowed to be updated

\[ \text{SR 89} \]

A coach or PR user can update the information of a feed item in his feed.

**Precondition:** The information of a feed item is changed by the coach who created it.

**Postcondition:** The changed information of the feed item is stored in the database.

### 3.1.7 NOTE

**Attributes:**

- **SR 90**
  - `text`: String
    - A String that holds the text of the note.

### 3.1.8 EXERCISE

**Attributes:**

- **SR 91**
  - `icon`: String
    - A String that holds the URL for the icon used to show the exercise.

- **SR 92**
  - `name`: String
    - A String that holds the name of the exercise.

### 3.1.9 TRAININGSCHEDULE

**Attributes:**

- **SR 93**
  - `time`: Date
    - A Date that holds the timestamp of when the training session is supposed to start.

**Adding exercises to a training**

- **SR 94**
  - `addExercises(Exercise[] exs)`
    - A function that adds candidate exercises to a training associated with an exercise poll feed item.
This function is called automatically when a coach creates an exercise poll feed item. The exercises to be added are pre-defined.

**SR 95**

exs: Exercise[]

A non-empty array containing Exercise objects that need to be added.

**Precondition:** A coach executed the functionality to create a new exercise poll feed item.

**Postcondition:** The candidate exercises are added to the training which is associated to an exercise poll feed item.

### 3.1.10 CHATABLE

**Chatting with a player or coach**

**SR 96**

`chatWith(String id)`

A function that lets a coach or player chat with another coach.

**SR 97**

`id`: String

The unique identifier of the user with whom the coach wants to chat.

**SR 98**

Only a coach can initiate a chat.

**SR 99**

A coach can only initiate a chat with another user in his team.

**Precondition:** The coach has selected a user to chat with. The selected user can be a coach or a player from the same team.

**Postcondition:** A chat session is established between the coach and the selected user. The chat history is stored in the database.

### 3.1.11 SUGGESTION FEED ITEM

**Attributes:**

**SR 100**

`deadline`: Date

A Date object that holds the deadline of the suggestion.

**SR 101**

`playerID`: String

The unique identifier of the player who is allowed to suggest.
SR 101a  
If playerID is undefined, then the ID of a random player in the team is assigned.

SR 102  
suggestion: String  
A String that holds the exercise that the player suggests. The exercise can be one of those stored in CoachAssist.

Choosing exercise of a suggestion feed item
SR 103  
setSuggestion(String suggestion) A function that registers the suggestion from a player to a suggestion feed item. This function is called automatically when the system registers a response to a suggestion feed item.

SR 104  
suggestion: String  
A String that stores the suggestion provided by the player. It can contain URLs.

Precondition: A player selected the feed item and has typed a suggestion. The deadline of the suggestion feed item has not passed yet.
Postcondition: The suggestion to the suggestion feed item is stored in the database. A notification is sent to the coach who created this exercise suggestion feed item.

3.1.12 HEROES
Attributes:
SR 105  
image: String  
A String that holds the URL of the image used in the heroes item.

3.1.13 BETTING ROUND
Attributes:
SR 106  
deadline: Date  
A Date object that holds the deadline of the betting round.

SR 107  
matches: String[][]  
A two-dimensional String array that holds the team names of the teams in the matches.
SR 108

results: String[]

A String array that stores the results of the matches in a betting round. Each element can only take value of either win, lose, tie or none.

---

Getting the results of a betting round feed item

SR 109

getResults()

A function that gets the results of the matches in a betting round feed item.

Precondition: None

Postcondition: The function returns a list of the results of each match in the betting round.

---

3.1.14 SPONSORING

Attributes:

SR 110
deadline: Date

A Date object that holds the deadline of the sponsoring event.

SR 111
raisedAmount: double

A double that holds the sum of the amounts that have been sponsored.

SR 112
sponsorships: double[

An associative array of (String, double) pairs that stores which user sponsored how much.

SR 113
targetAmount: double

A double that holds the target amount that needs to be raised.

SR 114

The value of targetAmount is always greater or equal to raisedAmount.

---

Adding a sponsor to a sponsoring feed item

SR 115

addSponsor(String userID, double amount)

A function that adds a new sponsor to a sponsoring feed item. This function is called automatically when the system registeres a response to a sponsoring feed item.
SR 116 should have
userID: String
The unique identifier of the user that sponsors the event.

SR 117 should have
amount: double
A double that holds the amount sponsored.

**Precondition:** A user selected the sponsor feed item and typed the amount he wants to contribute. The deadline of the item has not passed yet. The target amount is not yet met.

**Postcondition:** The sponsor is stored in the database. The raised amount is increased by the amount specified in the new sponsor. If the raised amount is equal to the target amount, the feed item closes and the sponsoring event ends.

Checking whether a sponsoring feed item has ended

SR 118 should have
checkEnd
A function that checks whether a sponsoring feed item has ended. If the target amount is reached or the deadline has passed, the feed item closes and the sponsoring event ends.

**Precondition:** None.

**Postcondition:** The function returns true if the feed item has ended. Otherwise false.

3.1.15 **PRACTICALITY FORM**

**Attributes:**

SR 119 must have
achievedValue: int
An integer that holds the sum of the contributions so far.

SR 120 must have
contributions: int[]
An integer array that holds the pairs of contributions and userID’s.

SR 121 must have
locked: Boolean
A Boolean that holds whether the form is locked or not.

SR 122 should have
repeatInterval: String
A String that holds how often the form should be repeated. It can take only value a of daily, weekly, or monthly.
must have target: String
A String that holds the target to be reached by the form. It can take one of the following pre-defined values: Laundry, Driving and Absence.

must have targetValue: int
An integer that holds the value of the target to be reached.

must have The value of targetValue is always greater or equal to achievedValue.

must have The value of locked is true if and only if achievedValue is equal to targetValue.

Adding a contribution to a practicality form

must have addContribution(String id, int amount)
A function that registers a new contribution to a practicality form feed item. This function is called automatically when the system registers a response to a practicality form feed item.

must have id: String.
The unique identifier of the user who makes the contribution.

must have amount: int
The amount the user wants to contribute. It can only take a value of 0 when the targetValue is 0.

Precondition: A user selected the practicality form feed item. The item is not locked.
Postcondition: The contribution is stored in the database. If the target value is greater than 0, then the raised value is increased by the amount specified in the new contribution.

Withdrawing a contribution from a practicality form

must have withdraw(String userID)
A function that withdraws a contribution of a user from a practicality form. This function is called automatically when the system withdraws a response to a practicality form feed item.

must have userID: String
The unique identifier of a user.

Precondition: The user selected a practicality form feed item to which he contributed.
**Postcondition:** The contribution of the user is removed from the database.

### 3.1.16 EXERCISE POLL

**Attributes:**

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SR 132</td>
<td>ended</td>
<td>A Boolean that indicates whether the voting is open or not.</td>
</tr>
<tr>
<td>SR 133</td>
<td>finalPublic</td>
<td>A Boolean that indicates whether the final results are visible to the players or not.</td>
</tr>
<tr>
<td>SR 134</td>
<td>intPublic</td>
<td>A Boolean that indicates whether the intermediate results are visible to the players or not.</td>
</tr>
<tr>
<td>SR 135</td>
<td>nrVotes</td>
<td>An integer that indicates the amount of players that have voted on the poll.</td>
</tr>
<tr>
<td>SR 136</td>
<td>nrVoters</td>
<td>An integer that indicates the amount of players that can vote on the poll.</td>
</tr>
<tr>
<td>SR 137</td>
<td>results</td>
<td>An associative array of (String, integer) pairs that stores the votes for each exercise.</td>
</tr>
</tbody>
</table>

The value of `ended` is true if any or both of the following conditions hold:

- `nrVotes` is equal to `nrVoters`
- The deadline has passed.

**Getting the candidate exercises of an exercise poll**

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SR 139</td>
<td>getExercises</td>
<td>A function that returns a list of IDs of the candidate exercises of an exercise poll.</td>
</tr>
</tbody>
</table>

**Precondition:** None.

**Postcondition:** The function returns a list of IDs of the candidate exercises. In case the candidate exercises are not added yet, it returns an empty list.
Getting the winner of an exercise poll

SR 140

**getWinner()**

A function that returns the exercise ID of the winner of an exercise poll.

**Precondition**: None.

**Postcondition**: If all players have casted their votes or the deadline has passed, the function returns the exercise with the most votes. In case of a tie, a random exercise is returned. Otherwise the function returns null.

---

Getting the results of an exercise poll

SR 141

**getResults()**

A function that returns the results of an exercise poll.

**Precondition**: The intermediate or final results are public.

**Postcondition**: The function returns an associative array of a mapping from each candidate exercise to its votes.

---

Registering a new vote

SR 142

**newVote(String id)**

A function that registers a new vote casted by a player. This function is called automatically when the system registers a new response to an exercise poll feed item.

SR 143

**id**: String

The unique identifier of the exercise that gets a new vote.

**Precondition**: There is a new response to an exercise poll feed item; The feed item has not ended yet.

**Postcondition**: The new vote is stored in the database. The total number of votes is increased by one. The number of votes on the exercise is increased by one. If all players have casted their votes, the feed item ends.

---

3.1.17 BETTING POOL

Attributes:

SR 144

**season**: String

could have
A String that holds which season this betting pool belongs to.

**Getting the leader board of a betting pool**

**SR 145**

*getLeaderBoard(int limit)*

A function that returns the leader board of a betting pool.

**SR 146**

*limit: int*

An integer that specifies how many leaders should be returned.

**Precondition:** None.

**Postcondition:** The function returns a list of leaders together with the points earned by them.
3.2 NON-FUNCTIONAL REQUIREMENTS

3.2.1 CLUBNET ENVIRONMENT

SR 147  
*must have*

The ClubNet app uses the Angular-Translate internationalization service to support multiple languages.

SR 148  
*must have*

The ClubNet app is available in Dutch.

SR 149  
*should have*

The ClubNet app is available in English.

SR 150  
*must have*

The ClubNet app runs on smartphones running Android 4.1 (Jelly Bean) and higher.

SR 151  
*must have*

The ClubNet app runs on smartphones running iOS 7.0 and higher.

SR 152  
*must have*

The ClubNet app runs on tablet devices running iOS 7.0 and higher.

SR 153  
*must have*

The ClubNet app runs on tablet devices running Android 4.1 (Jelly Bean) and higher.

SR 154  
*must have*

The ClubNet app runs in portrait mode.

SR 155  
*could have*

ClubNet runs in landscape mode.

SR 156  
*must have*

The ClubNet app uses Cordova to export the code to native Android applications.

SR 157  
*must have*

The ClubNet app uses Cordova to export the code to native IOS applications.

SR 158  
*must have*

The ClubNet app is implemented using the Ionic framework.

SR 159  
*must have*

The ClubNet app uses AngularJS for dynamic views in the front-end.

SR 160  
*must have*

The ClubNet app uses the Meteor framework for the back-end server.

SR 161  
*must have*

The ClubNet database is implemented using MongoDB.
3.2.2 WEB ENVIRONMENT

SR 162 must have
The web interface runs and displays correctly in Chrome version 48 and newer.

SR 163 should have
The web interface runs and displays correctly in Firefox version 44 and newer.

SR 164 should have
The web interface runs and displays correctly in Internet Explorer version 11 and newer.

SR 165 should have
The web interface runs and displays correctly in Safari version 8 and newer.

SR 166 must have
The web interface uses AngularJS for dynamic views.

3.2.3 BACK-END

SR 167 must have
The back-end server can call the API for retrieving data on the back-end server of CoachAssist.

SR 168 should have
The back-end server can call the API for sending data on the back-end server of CoachAssist.

SR 169 must have
The back-end server can handle concurrent requests from multiple front-end request.

SR 170 must have
The back-end database stores consistent data.

SR 171 must have
The back-end database cannot be accessed directly outside of the back-end server.

3.2.4 FRONT-END

SR 172 must have
The ClubNet app supports a view for the feed.

SR 173 should have
The ClubNet app supports a view for the details of a feed item.

SR 174 must have
The ClubNet app supports a view for creating a feed item.

SR 175 must have
The ClubNet app supports a sidebar for navigation.
### 3.2.5 Supportability

| SR 181 | must have |
| ClubNet is built on a single code-base. |

| SR 182 | must have |
| The ClubNet system is implemented in a modularized style. |

| SR 183 | must have |
| All functionality implemented on the back-end server can be tested. |

| SR 184 | must have |
| The back-end server can be extended to have more functionality. |

### 3.2.6 Reliability

| SR 185 | must have |
| The back-end server is available at least 24 hours 350 days per year. |

### 3.2.7 Performance

| SR 186 | must have |
| The back-end server must respond to front-end requests in 1 second. |

| SR 187 | must have |
| The content of a view should be loaded within 5 seconds. |

<p>| SR 188 | should have |
| The content of a view should be loaded within 3 seconds. |</p>
<table>
<thead>
<tr>
<th>SR 189</th>
<th>could have</th>
<th>The content of a view should be loaded within 1 seconds.</th>
</tr>
</thead>
<tbody>
<tr>
<td>SR 190</td>
<td>must have</td>
<td>A user interaction should result in visual feedback within 1 second.</td>
</tr>
<tr>
<td>SR 191</td>
<td>could have</td>
<td>A user interaction should result in visual feedback within 0.5 seconds.</td>
</tr>
<tr>
<td>SR 192</td>
<td>could have</td>
<td>A user interaction should result in visual feedback within 0.2 seconds.</td>
</tr>
</tbody>
</table>

### 3.2.8 USABILITY

<table>
<thead>
<tr>
<th>SR 193</th>
<th>must have</th>
<th>80% of the users will be satisfied with the usability of the ClubNet app.</th>
</tr>
</thead>
<tbody>
<tr>
<td>SR 194</td>
<td>must have</td>
<td>95% of the users will be satisfied with the usability of the web interface.</td>
</tr>
<tr>
<td>SR 195</td>
<td>must have</td>
<td>A user must be able to perform a use case without any hints within 10 minutes.</td>
</tr>
<tr>
<td>SR 196</td>
<td>should have</td>
<td>A user must be able to perform a use case without any hints within 5 minutes.</td>
</tr>
<tr>
<td>SR 197</td>
<td>could have</td>
<td>A user must be able to perform a use case without any hints within 2 minutes.</td>
</tr>
<tr>
<td>SR 198</td>
<td>must have</td>
<td>A user must be able to perform any functionality within 10 user actions.</td>
</tr>
<tr>
<td>SR 199</td>
<td>should have</td>
<td>A user must be able to perform any functionality within 5 user actions.</td>
</tr>
<tr>
<td>SR 200</td>
<td>could have</td>
<td>A user must be able to perform any functionality within 3 user actions.</td>
</tr>
</tbody>
</table>
# 4 REQUIREMENTS TRACEABILITY MATRIX

## 4.1 URD TO SRD

<table>
<thead>
<tr>
<th>URD</th>
<th>SRD</th>
</tr>
</thead>
<tbody>
<tr>
<td>URF1-1</td>
<td>55, 56</td>
</tr>
<tr>
<td>URF1-2</td>
<td>55, 56</td>
</tr>
<tr>
<td>URF1-3</td>
<td>57, 58</td>
</tr>
<tr>
<td>URF1-4</td>
<td>57, 58</td>
</tr>
<tr>
<td>URF1-5</td>
<td>17, 18, 19</td>
</tr>
<tr>
<td>URF1-6</td>
<td>20</td>
</tr>
<tr>
<td>URF1-7</td>
<td>17, 18, 19</td>
</tr>
<tr>
<td>URF1-8</td>
<td>20</td>
</tr>
<tr>
<td>URF1-9</td>
<td>9, 10</td>
</tr>
<tr>
<td>URF1-10</td>
<td>11</td>
</tr>
<tr>
<td>URF1-11</td>
<td>23, 24, 25</td>
</tr>
<tr>
<td>URF1-12</td>
<td>78, 80</td>
</tr>
<tr>
<td>URF1-13</td>
<td>78</td>
</tr>
<tr>
<td>URF1-14</td>
<td>78</td>
</tr>
<tr>
<td>URF1-15</td>
<td>78, 79</td>
</tr>
<tr>
<td>URF1-16</td>
<td>24, 25</td>
</tr>
<tr>
<td>URF1-17</td>
<td>68, 69</td>
</tr>
<tr>
<td>URF1-18</td>
<td>72, 73</td>
</tr>
<tr>
<td>URF1-19</td>
<td>70, 71</td>
</tr>
<tr>
<td>URF1-20</td>
<td>96, 97, 98, 99</td>
</tr>
<tr>
<td>URF1-21</td>
<td>96, 97, 98, 99</td>
</tr>
<tr>
<td>URF1-22</td>
<td>32, 33, 34</td>
</tr>
<tr>
<td>URF1-23</td>
<td>39, 40, 41</td>
</tr>
<tr>
<td>URF1-24</td>
<td>39, 40, 41</td>
</tr>
<tr>
<td>URF1-25</td>
<td>80</td>
</tr>
<tr>
<td>URF1-26</td>
<td>46, 47, 48, 49</td>
</tr>
<tr>
<td>URF1-27</td>
<td>46, 47, 48, 49</td>
</tr>
<tr>
<td>URF1-28</td>
<td>46, 47, 48, 49</td>
</tr>
<tr>
<td>URF1-29</td>
<td>46, 47, 48, 49</td>
</tr>
<tr>
<td>URF1-30</td>
<td>46, 47, 48, 49</td>
</tr>
<tr>
<td>URF1-31</td>
<td>46, 47, 48, 49</td>
</tr>
<tr>
<td>URF1-32</td>
<td>11, 12</td>
</tr>
<tr>
<td>URF1-33</td>
<td>11, 12</td>
</tr>
<tr>
<td>URF1-34</td>
<td>11, 12</td>
</tr>
<tr>
<td>---------</td>
<td>--------</td>
</tr>
<tr>
<td>URF1-35</td>
<td>11, 12</td>
</tr>
<tr>
<td>URF1-36</td>
<td>42, 43, 44</td>
</tr>
<tr>
<td>URF2-1</td>
<td>35, 36, 37</td>
</tr>
<tr>
<td>URF2-2</td>
<td>36, 85</td>
</tr>
<tr>
<td>URF2-3</td>
<td>36, 93</td>
</tr>
<tr>
<td>URF2-4</td>
<td>36, 93</td>
</tr>
<tr>
<td>URF2-5</td>
<td>36, 134</td>
</tr>
<tr>
<td>URF2-6</td>
<td>36, 133</td>
</tr>
<tr>
<td>URF2-7</td>
<td>35</td>
</tr>
<tr>
<td>URF2-8</td>
<td>137, 141</td>
</tr>
<tr>
<td>URF2-9</td>
<td>137, 141</td>
</tr>
<tr>
<td>URF2-10</td>
<td>13, 15, 16, 142, 143</td>
</tr>
<tr>
<td>URF2-11</td>
<td>140</td>
</tr>
<tr>
<td>URF2-12</td>
<td>140</td>
</tr>
<tr>
<td>URF2-13</td>
<td>140</td>
</tr>
<tr>
<td>URF2-14</td>
<td>140</td>
</tr>
<tr>
<td>URF2-15</td>
<td>141</td>
</tr>
<tr>
<td>URF3-1</td>
<td>35, 36, 37</td>
</tr>
<tr>
<td>URF3-2</td>
<td>35</td>
</tr>
<tr>
<td>URF3-3</td>
<td>36, 85</td>
</tr>
<tr>
<td>URF3-4</td>
<td>36, 124</td>
</tr>
<tr>
<td>URF3-5</td>
<td>36, 122</td>
</tr>
<tr>
<td>URF3-6</td>
<td>36, 122</td>
</tr>
<tr>
<td>URF3-7</td>
<td>36, 123</td>
</tr>
<tr>
<td>URF3-8</td>
<td>36, 123</td>
</tr>
<tr>
<td>URF3-9</td>
<td>36, 126</td>
</tr>
<tr>
<td>URF3-10</td>
<td>127, 128, 129, 13, 15, 16</td>
</tr>
<tr>
<td>URF3-11</td>
<td>28, 29, 30, 130, 131</td>
</tr>
<tr>
<td>URF3-12</td>
<td>126</td>
</tr>
<tr>
<td>URF3-15</td>
<td>123</td>
</tr>
<tr>
<td>URF3-16</td>
<td>124</td>
</tr>
<tr>
<td>URF3-17</td>
<td>15, 129</td>
</tr>
<tr>
<td>URF3-18</td>
<td>123</td>
</tr>
<tr>
<td>URF3-19</td>
<td>123</td>
</tr>
<tr>
<td>URF3-20</td>
<td>120</td>
</tr>
<tr>
<td>URF4-1</td>
<td>35, 36, 37</td>
</tr>
<tr>
<td>URF4-2</td>
<td>35</td>
</tr>
<tr>
<td>URF4-3</td>
<td>113</td>
</tr>
<tr>
<td>URF4-4</td>
<td>110</td>
</tr>
<tr>
<td>--------</td>
<td>-----</td>
</tr>
<tr>
<td>URF4-5</td>
<td>85</td>
</tr>
<tr>
<td>URF4-6</td>
<td>82</td>
</tr>
<tr>
<td>URF4-7</td>
<td>15, 115, 116, 117</td>
</tr>
<tr>
<td>URF4-8</td>
<td>111</td>
</tr>
<tr>
<td>URF4-9</td>
<td>112, 76, 77</td>
</tr>
<tr>
<td>URF4-10</td>
<td>42, 43, 45</td>
</tr>
<tr>
<td>URF4-11</td>
<td>42, 43, 44</td>
</tr>
<tr>
<td>URF4-12</td>
<td>51, 52, 53, 54</td>
</tr>
<tr>
<td>URF4-13</td>
<td>53</td>
</tr>
<tr>
<td>URF4-14</td>
<td>54</td>
</tr>
<tr>
<td>URF4-15</td>
<td>111, 113</td>
</tr>
<tr>
<td>URF5-1</td>
<td>63, 64, 65, 66, 67</td>
</tr>
<tr>
<td>URF5-2</td>
<td>63, 64, 65, 66, 67</td>
</tr>
<tr>
<td>URF5-3</td>
<td>66</td>
</tr>
<tr>
<td>URF5-4</td>
<td>63</td>
</tr>
<tr>
<td>URF5-5</td>
<td>65</td>
</tr>
<tr>
<td>URF5-6</td>
<td>65</td>
</tr>
<tr>
<td>URF5-7</td>
<td>145, 146</td>
</tr>
<tr>
<td>URF5-8</td>
<td>13, 14, 15, 16, 106</td>
</tr>
<tr>
<td>URF5-9</td>
<td>6, 108</td>
</tr>
<tr>
<td>URF5-10</td>
<td>145, 146</td>
</tr>
<tr>
<td>URF5-11</td>
<td>42, 43, 45</td>
</tr>
<tr>
<td>URF5-12</td>
<td>109</td>
</tr>
<tr>
<td>URF5-13</td>
<td>31</td>
</tr>
<tr>
<td>URF5-14</td>
<td>50</td>
</tr>
<tr>
<td>URF6-1</td>
<td>35, 36, 37</td>
</tr>
<tr>
<td>URF6-2</td>
<td>35, 36, 38</td>
</tr>
<tr>
<td>URF6-3</td>
<td>74, 75</td>
</tr>
<tr>
<td>URF6-4</td>
<td>75</td>
</tr>
<tr>
<td>URF6-5</td>
<td>37a</td>
</tr>
<tr>
<td>URF6-6</td>
<td>59, 60</td>
</tr>
<tr>
<td>URF6-7</td>
<td>59</td>
</tr>
<tr>
<td>URF6-8</td>
<td>61, 62</td>
</tr>
<tr>
<td>URF6-9</td>
<td>35</td>
</tr>
<tr>
<td>URF6-10</td>
<td>42, 43, 45</td>
</tr>
<tr>
<td>URF6-11</td>
<td>85</td>
</tr>
<tr>
<td>URF6-12</td>
<td>82</td>
</tr>
<tr>
<td>URF6-13</td>
<td>105</td>
</tr>
<tr>
<td>Code</td>
<td>Pages</td>
</tr>
<tr>
<td>--------</td>
<td>-------</td>
</tr>
<tr>
<td>URF6-14</td>
<td>85</td>
</tr>
<tr>
<td>URF6-15</td>
<td>82</td>
</tr>
<tr>
<td>URF6-16</td>
<td>105</td>
</tr>
<tr>
<td>URF6-17</td>
<td>79, 80</td>
</tr>
<tr>
<td>URF7-1</td>
<td>101</td>
</tr>
<tr>
<td>URF7-2</td>
<td>101a</td>
</tr>
<tr>
<td>URF7-3</td>
<td>13, 14, 15, 16, 103, 104</td>
</tr>
<tr>
<td>URF7-4</td>
<td>102</td>
</tr>
<tr>
<td>URF7-5</td>
<td>104</td>
</tr>
<tr>
<td>URF7-6</td>
<td>104</td>
</tr>
<tr>
<td>URF7-7</td>
<td>35</td>
</tr>
<tr>
<td>URF7-8</td>
<td>103</td>
</tr>
<tr>
<td>URF7-9</td>
<td>15, 16</td>
</tr>
<tr>
<td>URF7-10</td>
<td>102</td>
</tr>
<tr>
<td>URC1-1</td>
<td>144</td>
</tr>
<tr>
<td>URC1-2</td>
<td>145</td>
</tr>
<tr>
<td>URC1-3</td>
<td>146</td>
</tr>
<tr>
<td>URC2-1</td>
<td>148</td>
</tr>
<tr>
<td>URC2-2</td>
<td>150</td>
</tr>
<tr>
<td>URC2-3</td>
<td>149</td>
</tr>
<tr>
<td>URC2-4</td>
<td>150</td>
</tr>
<tr>
<td>URC2-5</td>
<td>151</td>
</tr>
<tr>
<td>URC2-6</td>
<td>152</td>
</tr>
<tr>
<td>URC2-7</td>
<td>153, 154, 155, 178</td>
</tr>
<tr>
<td>URC3-1</td>
<td>159</td>
</tr>
<tr>
<td>URC3-2</td>
<td>160</td>
</tr>
<tr>
<td>URC3-3</td>
<td>161</td>
</tr>
<tr>
<td>URC3-4</td>
<td>162</td>
</tr>
<tr>
<td>URC4-1</td>
<td>164</td>
</tr>
<tr>
<td>URC4-2</td>
<td>165</td>
</tr>
<tr>
<td>URC5-1</td>
<td>179, 180, 181</td>
</tr>
<tr>
<td>URC6-1</td>
<td>192</td>
</tr>
<tr>
<td>URC6-2</td>
<td>193</td>
</tr>
<tr>
<td>URC6-3</td>
<td>194</td>
</tr>
<tr>
<td>URC6-4</td>
<td>195</td>
</tr>
<tr>
<td>URC6-5</td>
<td>196</td>
</tr>
<tr>
<td>URC6-6</td>
<td>197</td>
</tr>
<tr>
<td>URC7-1</td>
<td>184</td>
</tr>
<tr>
<td>URC7-2</td>
<td>185</td>
</tr>
</tbody>
</table>
### 4.2 SRD TO URD

<table>
<thead>
<tr>
<th>SRD</th>
<th>URD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>URF1-1, URF1-2, URF1-13, URF4-1, URF5-4</td>
</tr>
<tr>
<td>2</td>
<td>URF1-5, URF1-7</td>
</tr>
<tr>
<td>3</td>
<td>URF1-5, URF1-7</td>
</tr>
<tr>
<td>4</td>
<td>URF1-1, URF1-2</td>
</tr>
<tr>
<td>5</td>
<td>URF1-1, URF1-2</td>
</tr>
<tr>
<td>6</td>
<td>URF5-7, URF5-9, URF5-10</td>
</tr>
<tr>
<td>7</td>
<td>URC1-1, URC1-2, URC1-3</td>
</tr>
<tr>
<td>8</td>
<td>URC1-1, URC1-2, URC1-3</td>
</tr>
<tr>
<td>9</td>
<td>URC1-11</td>
</tr>
<tr>
<td>10</td>
<td>URC1-11</td>
</tr>
<tr>
<td>11</td>
<td>URF1-32, URF1-33, URF1-34, URF1-35</td>
</tr>
<tr>
<td>12</td>
<td>URF1-10, URF1-32, URF1-33, URF1-34, URF1-35</td>
</tr>
<tr>
<td>13</td>
<td>URF2-10, URF3-10, URF4-7, URF5-8, URF7-3</td>
</tr>
<tr>
<td>14</td>
<td>URF2-10, URF3-10, URF4-7, URF5-8, URF7-3</td>
</tr>
<tr>
<td>15</td>
<td>URF2-10, URF3-10, URF4-7, URF5-8, URF7-3</td>
</tr>
<tr>
<td>16</td>
<td>URF2-10, URF3-10, URF4-7, URF5-8, URF7-3</td>
</tr>
<tr>
<td>17</td>
<td>URF1-5, URF1-7</td>
</tr>
<tr>
<td>18</td>
<td>URF1-5, URF1-7</td>
</tr>
<tr>
<td>19</td>
<td>URF1-5, URF1-7</td>
</tr>
<tr>
<td>20</td>
<td>URF1-6, URF1-8</td>
</tr>
<tr>
<td>21</td>
<td>URF1-16, URF2-7, URF3-2, URF4-2, URF5-4, URF7-7, URF7-8</td>
</tr>
<tr>
<td>22</td>
<td>URF1-16, URF2-7, URF3-2, URF4-2, URF5-4, URF7-7, URF7-8</td>
</tr>
<tr>
<td>23</td>
<td>URF1-11</td>
</tr>
<tr>
<td>24</td>
<td>URF1-16</td>
</tr>
<tr>
<td>25</td>
<td>URF1-16</td>
</tr>
<tr>
<td>26</td>
<td>URF1-9</td>
</tr>
<tr>
<td>27</td>
<td>URF1-10</td>
</tr>
<tr>
<td>28</td>
<td>URF3-11</td>
</tr>
<tr>
<td>29</td>
<td>URF3-11</td>
</tr>
<tr>
<td>30</td>
<td>URF3-11</td>
</tr>
</tbody>
</table>
31  URF1-1, URF1-2, URF1-14, URF1-21, URF1-30, URF2-1, URF2-7, URF3-1, URF3-2
32  URF1-22
33  URF1-22
34  URF1-22
35  URF2-1, URF3-1, URF4-1, URF5-1, URF5-2, URF6-1, URF6-2, URF7-1
36  URF2-1, URF3-1, URF4-1, URF5-1, URF5-2, URF6-1, URF6-2, URF7-1
37  URF6-1, URF4-1, URF2-1, URF3-1, URF7-1
38  URF4-1, URF6-1, URF5-1, URF5-2
39  URF1-23, URF1-24
40  URF1-23, URF1-24
41  URF1-23, URF1-24
42  URF1-36, URF4-10, URF4-11, URF5-11
43  URF4-10, URF4-11, URF5-11
44  URF4-11
45  URF4-10, URF5-11
46  URF1-26, URF1-27, URF1-28, URF1-29, URF1-30, URF1-31
47  URF1-26, URF1-27, URF1-28, URF1-29, URF1-30, URF1-31
48  URF1-27, URF1-30, URF1-31
49  URF1-28, URF1-26, URF1-29
50  URF1-1, URF1-2, URF1-14, URF1-21, URF1-30, URF2-1, URF2-7, URF3-1, URF3-2
51  URF4-12, URF4-13, URF4-14
52  URF4-12, URF4-13, URF4-14
53  URF4-12, URF4-13, URF4-14
54  URF4-12, URF4-13, URF4-14
55  URF1-1, URF1-2
56  URF1-1, URF1-2
57  URF1-3, URF1-4
58  URF1-3, URF1-4
59  URF6-6, URF6-7
60  URF6-6, URF6-7
61  URF6-8
62  URF6-8
63  URF5-1
64  URF5-1
65  URF5-5, URF5-6
66  URF5-3
67  URF5-2
URF1-17
URF1-17
URF1-19
URF1-19
URF1-18
URF1-18
URF6-3
URF6-3
URF4-9
URF4-9
URF1-15
URF1-15
URF1-12, URF1-13, URF1-14
URF1-13
URF4-6, URF6-12, URF6-15
URF2-1, URF3-1, URF4-1, URF5-1, URF5-2, URF6-1, URF6-2, URF7-1
URF1-26, URF1-27, URF1-28, URF1-29, URF1-30, URF1-31
URF2-2, URF3-3, URF4-5, URF6-11, URF6-14
URF2-1, URF3-1, URF4-1, URF5-2, URF6-1, URF6-2, URF7-1
URF2-2, URF2-5, URF2-6, URF3-17, URF3-3, URF3-4, URF3-6, URF4-3,
URF4-5, URF4-6, URF6-11, URF6-14, URF6-12, URF6-15
URF2-2, URF2-5, URF2-6, URF3-17, URF3-3, URF3-4, URF3-6, URF4-3,
URF4-5, URF4-6, URF6-11, URF6-14, URF6-12, URF6-15
URF2-2, URF2-5, URF2-6, URF3-17, URF3-3, URF3-4, URF3-5, URF3-6,
URF3-7, URF4-4, URF4-5, URF4-6, URF6-10, URF6-11, URF6-12, URF6-
13, URF6-14, URF6-15, URF6-16
URF1-22, URF1-23
URF2-1, URF2-10
URF2-1, URF2-10
URF2-4, URF2-3
URC4-1
URC4-1
URF1-20, URF1-21
URF1-20, URF1-21
URF1-20, URF1-21
URF1-20, URF1-21
URF7-1
URF7-1, URF7-2
URF7-3, URF7-5, URF7-6
<table>
<thead>
<tr>
<th>Page</th>
<th>Requirement Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>103</td>
<td>URF7-3</td>
</tr>
<tr>
<td>104</td>
<td>URF7-3, URF7-5, URF7-6</td>
</tr>
<tr>
<td>105</td>
<td>URF6-13, URF6-16</td>
</tr>
<tr>
<td>106</td>
<td>URF5-8</td>
</tr>
<tr>
<td>107</td>
<td>URF5-2, URF5-3</td>
</tr>
<tr>
<td>108</td>
<td>URF5-12</td>
</tr>
<tr>
<td>109</td>
<td>URF5-12</td>
</tr>
<tr>
<td>110</td>
<td>URF4-4</td>
</tr>
<tr>
<td>111</td>
<td>URF4-8</td>
</tr>
<tr>
<td>112</td>
<td>URF4-9</td>
</tr>
<tr>
<td>113</td>
<td>URF4-3</td>
</tr>
<tr>
<td>114</td>
<td>URF4-9</td>
</tr>
<tr>
<td>115</td>
<td>URF4-7</td>
</tr>
<tr>
<td>116</td>
<td>URF4-7</td>
</tr>
<tr>
<td>117</td>
<td>URF4-7</td>
</tr>
<tr>
<td>118</td>
<td>URF4-15</td>
</tr>
<tr>
<td>119</td>
<td>URF3-12, URF3-9</td>
</tr>
<tr>
<td>120</td>
<td>URF3-20</td>
</tr>
<tr>
<td>121</td>
<td>URF3-12, URF3-9</td>
</tr>
<tr>
<td>122</td>
<td>URF3-6</td>
</tr>
<tr>
<td>123</td>
<td>URF3-7, URF3-8, URF3-15, URF3-18, URF3-19</td>
</tr>
<tr>
<td>124</td>
<td>URF3-4, URF3-16</td>
</tr>
<tr>
<td>125</td>
<td>URF3-9, URF3-12</td>
</tr>
<tr>
<td>126</td>
<td>URF3-9, URF3-12</td>
</tr>
<tr>
<td>127</td>
<td>URF3-10, URF3-17</td>
</tr>
<tr>
<td>128</td>
<td>URF3-10, URF3-17</td>
</tr>
<tr>
<td>129</td>
<td>URF3-10, URF3-17</td>
</tr>
<tr>
<td>130</td>
<td>URF3-11</td>
</tr>
<tr>
<td>131</td>
<td>URF3-11</td>
</tr>
<tr>
<td>132</td>
<td>URF2-11, URF2-12</td>
</tr>
<tr>
<td>133</td>
<td>URF2-6</td>
</tr>
<tr>
<td>134</td>
<td>URF2-5</td>
</tr>
<tr>
<td>135</td>
<td>URF2-12</td>
</tr>
<tr>
<td>136</td>
<td>URF2-12</td>
</tr>
<tr>
<td>137</td>
<td>URF2-15</td>
</tr>
<tr>
<td>138</td>
<td>URF2-11, URF2-12</td>
</tr>
<tr>
<td>139</td>
<td>URF2-10, URF2-15</td>
</tr>
<tr>
<td>140</td>
<td>URF2-11, URF2-12, URF2-13, URF2-14</td>
</tr>
<tr>
<td>141</td>
<td>URF2-15</td>
</tr>
<tr>
<td>Page</td>
<td>Requirement List</td>
</tr>
<tr>
<td>------</td>
<td>------------------</td>
</tr>
<tr>
<td>142</td>
<td>URF2-10</td>
</tr>
<tr>
<td>143</td>
<td>URF2-10</td>
</tr>
<tr>
<td>144</td>
<td>URF5-1</td>
</tr>
<tr>
<td>145</td>
<td>URF5-10</td>
</tr>
<tr>
<td>146</td>
<td>URF5-10</td>
</tr>
<tr>
<td>147</td>
<td>URC1-1</td>
</tr>
<tr>
<td>148</td>
<td>URC1-2</td>
</tr>
<tr>
<td>149</td>
<td>URC1-3</td>
</tr>
<tr>
<td>150</td>
<td>URC2-2</td>
</tr>
<tr>
<td>151</td>
<td>URC2-1</td>
</tr>
<tr>
<td>152</td>
<td>URC2-3</td>
</tr>
<tr>
<td>153</td>
<td>URC2-4</td>
</tr>
<tr>
<td>154</td>
<td>URC2-5</td>
</tr>
<tr>
<td>155</td>
<td>URC2-6</td>
</tr>
<tr>
<td>156</td>
<td>URC2-7</td>
</tr>
<tr>
<td>157</td>
<td>URC2-7</td>
</tr>
<tr>
<td>158</td>
<td>URC2-7</td>
</tr>
<tr>
<td>159</td>
<td>URC2-7</td>
</tr>
<tr>
<td>160</td>
<td>URC2-7</td>
</tr>
<tr>
<td>161</td>
<td>URC2-7</td>
</tr>
<tr>
<td>162</td>
<td>URC3-1</td>
</tr>
<tr>
<td>163</td>
<td>URC3-2</td>
</tr>
<tr>
<td>164</td>
<td>URC3-3</td>
</tr>
<tr>
<td>165</td>
<td>URC3-4</td>
</tr>
<tr>
<td>166</td>
<td>URC2-7</td>
</tr>
<tr>
<td>167</td>
<td>URC4-1</td>
</tr>
<tr>
<td>168</td>
<td>URC4-2</td>
</tr>
<tr>
<td>169</td>
<td>URC7-4, URC7-5, URC7-6</td>
</tr>
<tr>
<td>170</td>
<td>URC5-1</td>
</tr>
<tr>
<td>171</td>
<td>URC5-2</td>
</tr>
<tr>
<td>172</td>
<td>URF1-12</td>
</tr>
<tr>
<td>173</td>
<td>URF1-13</td>
</tr>
<tr>
<td>174</td>
<td>URF2-1, URF3-1, URF4-1, URF5-2, URF6-1, URF6-2, URF7-1</td>
</tr>
<tr>
<td>175</td>
<td>URF1-9, URF1-10, URF1-16</td>
</tr>
<tr>
<td>176</td>
<td>URF1-1, URF1-2, URF1-3, URF1-4</td>
</tr>
<tr>
<td>177</td>
<td>URF1-12</td>
</tr>
<tr>
<td>178</td>
<td>URF1-13</td>
</tr>
<tr>
<td>179</td>
<td>URF2-1, URF3-1, URF4-1, URF5-2, URF6-1, URF6-2, URF7-1</td>
</tr>
<tr>
<td>180</td>
<td>URF1-9, URF1-10, URF1-16</td>
</tr>
<tr>
<td>Line</td>
<td>URN</td>
</tr>
<tr>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>181</td>
<td>URC2-7</td>
</tr>
<tr>
<td>182</td>
<td>URC5-1</td>
</tr>
<tr>
<td>183</td>
<td>URC5-1</td>
</tr>
<tr>
<td>184</td>
<td>URC5-1</td>
</tr>
<tr>
<td>185</td>
<td>URC5-1</td>
</tr>
<tr>
<td>186</td>
<td>URC7-1, URC7-2, URC7-3</td>
</tr>
<tr>
<td>187</td>
<td>URC7-1</td>
</tr>
<tr>
<td>188</td>
<td>URC7-2</td>
</tr>
<tr>
<td>189</td>
<td>URC7-3</td>
</tr>
<tr>
<td>190</td>
<td>URC7-4</td>
</tr>
<tr>
<td>191</td>
<td>URC7-5</td>
</tr>
<tr>
<td>192</td>
<td>URC7-6</td>
</tr>
<tr>
<td>193</td>
<td>Negotiated with customer.</td>
</tr>
<tr>
<td>194</td>
<td>Negotiated with customer.</td>
</tr>
<tr>
<td>195</td>
<td>URC6-1</td>
</tr>
<tr>
<td>196</td>
<td>URC6-2</td>
</tr>
<tr>
<td>197</td>
<td>URC6-3</td>
</tr>
<tr>
<td>198</td>
<td>URC6-4</td>
</tr>
<tr>
<td>199</td>
<td>URC6-5</td>
</tr>
<tr>
<td>200</td>
<td>URC6-6</td>
</tr>
</tbody>
</table>
A USER INTERFACE - APP

A.1 LOG IN SCREEN

When a user first opens up the ClubNet application, they will find themselves at the Log In screen. A user must first log in before they can make use of the functionality of ClubNet. After entering the e-mail associated with their account as well as their password and pressing the Log In button, they are directed to the feed of the application. Whenever the user fills in wrong credentials, an error message is displayed. The same error message is displayed when the user does not fill in a field before pressing the Log In button, or when the user tries to log in with an account that is of type PR. Additionally, the Log In screen contains a Forgot Password button that redirects the user to the Forgot Password Screen.

![Log In Screen](image)

**FIGURE 24: LOG IN SCREEN**
A.2 FORGOT PASSWORD SCREEN AND RESET PASSWORD SCREEN

When a user presses the Forgot Password button on the Log In page, they are redirected to the Forgot Password screen. Here they can fill in the e-mail address associated with their account and press the Send Email button to receive an email with a special link. By clicking the link in the mail, the user is directed to a Reset Password page. Once they fill in a new password and a confirmation of the new password on the Reset Password page and press the Reset Password button, their password will be reset and they are logged in automatically.

![Forgot Password Screen](image1)

![Reset Password Screen](image2)

**FIGURE 25:** THE TWO SCREENS FOR RESETTING A LOST PASSWORD
A.3 FEED

Once a user has logged in using their account, they will end up on the feed of the application. Here, they can see various different feed items with information relevant to them, their team, and their club. Each feed item can be expanded by pressing the Read More button, or simply by pressing anywhere on the item. When a feed item is expanded, more information can be seen and it can then be closed again by pressing a new button that appears on the bottom of a feed item.

When logged in as a coach, the feed contains a bar at the bottom of the page that allows the coach to add new feed items, as can be seen in Figure 26. Players and regular club members cannot see this bar.

![Figure 26: The Feed](image_url)
A.3.1 CREATING A NEW ITEM

As mentioned above, a coach can create new feed items for his team (or for the club, if allowed). This is done by pressing the + button on the bottom of the screen. When this button is pressed a drop-up menu is shown as in Figure 27. Here, the coach can select what type of feed item to create. The drop-up menu can be closed by pressing the icon again or by pressing anywhere on the screen next to it.

![Creating a New Item](image)

**FIGURE 27: CREATING A NEW ITEM**
A.3.2 CREATING AN EXERCISE POLL

After pressing the + button to create a new item and then selecting the Voting option a full screen overlay appears in which a coach has to fill in multiple fields. He can select a training which he marked for exercise voting in CoachAssist from the CoachAssist database in the top right corner. Then he must specify a title, the deadline, and whether to show the intermediate and final results of the poll to the team. Lastly he can set a minimum number of voters needed to select a winning exercise when the deadline passes. When all the required fields are filled in the Post button will be more saturated and enabled. Closing the pop-up to cancel creating a new item is simply done by pressing close in the top right corner.

<table>
<thead>
<tr>
<th>New poll</th>
<th>Close</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training</td>
<td>Select a training</td>
</tr>
<tr>
<td>Title</td>
<td></td>
</tr>
<tr>
<td>dd-mm-yyyy</td>
<td></td>
</tr>
<tr>
<td>Show intermediate results to the team</td>
<td>☐</td>
</tr>
<tr>
<td>Show final results to the team</td>
<td>☐</td>
</tr>
<tr>
<td>Number of voters</td>
<td></td>
</tr>
</tbody>
</table>

FIGURE 28: creating a poll
A.3.3 MANIPULATING A FEED ITEM

After posting one or more feed items a coach can manipulate them by pressing the three dots in the top right corner of each feed item. Pressing these dots will show a small drop down list with the following items: Edit, Delete, and Leave a Note, which can be seen in Figure 29. Editing an item will simply show the same pop-up as creating it, so that each field can be edited easily. Pressing delete will display a small pop-up to confirm the deletion of that item. When a coach wants to leave a note it shows a small pop-up as well, this time with a text field to fill in whatever note the coach wants to leave. The drop down menu can be closed by pressing the icon again or by pressing anywhere on the screen next to it.

FIGURE 29: MANIPULATE A FEED ITEM
A.3.4 FILTER

To filter the feed on only those item types that are interesting to the user, he can press the filter icon on the right in the top bar of the app. Pressing this icon will show a drop down list with all the item types in the application. Each of these types can then be checked or unchecked to respectively show and hide them on the feed. The drop down menu can be closed by pressing the icon again or by pressing anywhere on the screen next to it.

FIGURE 30: FILTER ON ITEM TYPES
A.4 SIDEMENU

When a user presses the burger menu in the top left corner of the screen a sidemenu slides into view. Here they can select a multitude of different pages to navigate to, as well as log out. On the top of the sidemenu their club logo is shown with a small personalized greeting. Closing the sidemenu is done automatically when any of the items in the list are pressed, or when pressing anywhere next to it on the screen, or by pressing the burger menu button again.

FIGURE 31: THE SIDEMENU OF THE APP
A.5  CHAT

A big part of the communication from coach to pupils within the app is the chat. Coaches and players can chat directly, or coaches can start a chat with multiple players at once (like a one-to-many group chat). Players are not able to chat with each other.

A.5.1  CHAT SCREEN

When a user presses chat in the sidemenu he will be brought into the chats screen. Here the user can view previously closed or still opened chats. Each chat will show the name of the user and the last message sent or received in that chat, as well as the time of that message. In this screen users can start a new chat by pressing the + in the top right corner of the screen.

FIGURE 32: THE CHAT SCREEN
A.5.2 NEW CHAT SCREEN

When a user presses the + button to start a new chat a list of users he can chat with is displayed. Each user will be listed with his name and user account type, together with a small image. Pressing any of these players will open up the chat screen where message can be sent and received.

<table>
<thead>
<tr>
<th>New Chat</th>
<th>Cancel</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Avatar] Frank The Coach coach</td>
<td></td>
</tr>
<tr>
<td>![Avatar] Daniel van der Laan player</td>
<td></td>
</tr>
<tr>
<td>![Avatar] Kevin van Eenige player</td>
<td></td>
</tr>
<tr>
<td>![Avatar] Jarno Verhagen player</td>
<td></td>
</tr>
<tr>
<td>![Avatar] Jeroen Brouns player</td>
<td></td>
</tr>
</tbody>
</table>

FIGURE 33: SELECTING A NEW CHAT
A.5.3 CHATTING WITH SOMEONE

When a user presses the chat with another user it will open that chat and shows the previously sent and received messages, just like in other chat applications. Typing a new message is done simply by pressing the input field at the bottom of the screen. This will open up the device keyboard and let the user type a message.

FIGURE 34: CHATTING WITH SOMEONE
A.6 PROFILE SCREEN

Pressing the Profile button in the sidemenu brings the user to their profile page. Here, they can change some user information as well as their password. When the user updates their password, they are brought back to the feed.

FIGURE 35: PROFILE SCREEN
**B USER INTERFACE - WEB**

**B.1 LOG IN PAGE**

When someone opens up the web interface a simple log in page will be shown. Here the PR user can fill in his email and password to then press the Log In button which will log him in if the credentials are correct. If the PR person does not fill in correct credentials, they receive an error message. This message is also shown if the user tries to log in with an account that is not of type PR. In case the PR user forgot his password, they can press the Forgot Password? button which will create a small pop-up for password retrieval.

![Log In Page](figure36.png)

**FIGURE 36: LOG IN PAGE OF THE WEB INTERFACE**
B.2 FORGOT PASSWORD PAGE AND RESET PASSWORD

After pressing Forgot Password? this pop-up will show in which the user only has to fill in their email address, and press Send. This will send an email to the user to reset their password. This email holds a link to a special page where the PR user can type in their password twice to reset their password.

FIGURE 37: FORGOT PASSWORD PAGE OF THE WEB INTERFACE
B.3 SIDEMENU

After logging in to the web interface on the side of each page is the sidemenu. From here the PR user can navigate to most other pages, to prevent deeply nested pages. When hovering over the bar the hovered on item is highlighted, as well as the page the user is currently on. The last button, Log Out, will log out the PR user and bring him back to the log in screen.

**FIGURE 38:** HOVERING OVER AN ITEM IN THE SIDEMENU
B.4 FEED PAGE

The standard page a PR user sees when logging in to the web interface is the feed page. This page can also be reached by pressing the Feed button in the sidemenu. On this page feed items will be shown that are published on club level. Some of these items might need regulation by the PR user. For instance, the number of Heroes items in a club per month has a maximum and after that they need to be accepted or denied by the PR user. This feed will work the same as in the app, namely it will load a fixed amount of items, and when scrolling down it will load more, just like modern social media feeds (e.g. Facebook and Twitter).

![Feed Page Screenshot](image)

**FIGURE 39: THE FEED PAGE**
B.5 BETTING PAGE

When the PR user presses the Club Betting button in the sidemenu a club betting page will be shown. Here they can set up the betting rounds by selecting the teams for each match and filling in the entire form. On this page they will also be able to fill in the outcome of the match after the match has been played. When the outcome has been filled in, the bets placed by users in the application will be checked against the outcome, and points will be allocated accordingly.

FIGURE 40: THE BETTING PAGE
B.6 CLUB SETTINGS PAGE

When the PR user presses the Club Settings button in the sidemenu the club settings page will be shown. This page is mainly there for customization of the web interface and application. The PR user can set the club name, the club logo, and can pick a three color colorscheme. These colors propagate to the web interface as seen in the sidemenu, and in the app they will have a similar impact.

FIGURE 41: THE CLUB SETTINGS PAGE
B.7 PROFILE PAGE

When the PR user presses the Profile button in the sidemenu he will be brought into his profile page. Here he can change his personal information and change his password easily.

FIGURE 42: THE PROFILE PAGE OF THE PR USER
B.8 MEMBERS PAGE

When the PR user presses the Members button in the sidemenu he will be brought into the members overview page, which is arguably the most useful page of the web interface. On this page a complete overview of the members of the PR users club will be shown. For each member it will show their first and last name together with their email address, the team they play in or coach, and their account type. Additionally, it shows whether a user has verified his e-mail address or not. General member accounts and player accounts can be edited and deleted. Coach accounts cannot be edited or deleted because they will be imported from the CoachAssist database, so they will have to be changed via CoachAssist. PR accounts cannot be edited or deleted as well as PR users have to administrate the club and they can change their own account on the profile page. New administrators have to be added by ClubNet database affiliates.

FIGURE 43: THE MEMBERS OVERVIEW PAGE
B.8.1 ADD MEMBER

When the PR user presses the Add Member button on the members page he will be brought into an add member page. Here the PR user can create a new account for a member of the club. The PR user must specify the first and last name, the email address, and a team the member plays in. In case the PR user does not select a team (‘None’, as seen in Figure 44), the new user will automatically become a General Member. Pressing the Add member button will return the PR user to the members page and will send an email to the specified email address to confirm their registration for ClubNet.

FIGURE 44: THE ADD MEMBER PAGE
B.8.2 EDIT MEMBER

When the PR user presses the orange edit icon behind a user account he will be brought into an edit member page. Here he can edit the first and last name of a user as well as the team he is playing in. In case the PR user does not assign a team, the account will automatically be set to General Member, as is mentioned below the Team drop down list. Pressing the Save Changes button will save the changes and return the PR user to the members page.

FIGURE 45: THE EDIT A MEMBER PAGE
B.8.3 DELETE MEMBER

When the PR user presses the red trash can icon to the right of a user account a confirmation pop-up will show to ask the user if he is sure he wants to delete said account. The delete button is in red to accentuate its impact. Needless to say, the delete button will remove the user from the database. This pop-up can be closed by pressing cancel or clicking anywhere on the screen next to it.

FIGURE 46: DELETING A MEMBER
C TRANSITIONS

In the previous section, all the different user interfaces have been discussed. To show how the transitions between those user interfaces work, we present the following Petri-nets that model the behaviour.

From every single place, the user is able to close the web interface or the mobile application. This is a functionality that has not been modelled in the Petri nets below.

Furthermore, due to the user-friendliness constraint there are as little different screens as possible, and both the web interface and app therefore have a sidemenu [Figure 31] in which the users can go to almost all other views.

Note that the following models are Petri nets and the places (circles/ovals) represent screens while the transitions (boxes) represent actions taken by the user, for instance clicking on a button or filling in a form and submitting it.

Another functionality that has not been modelled is the back button of Android devices. This button will simply bring you back to the previous screen, except with logging in and logging out. When you log out and press back you will not be brought back into the app. Another reason to not model this button is that iOS devices do not have a button like this, and since we develop for both Android and iOS we tried to keep the models as general as possible.

As well as the back button of Android devices, pressing backspace or the back (or forward) button of any web browser has also not been modelled for the web interface. These buttons do what they should, as they return you to the previous page, if possible. This means that you cannot log out and then press back to get back into the web interface again. Lastly, a user can close any pop-up, drop down, or drop up lists by clicking anywhere next to these items. This will also not be modelled as it is general functionality for the entire app, and would show up unnecessarily much in the model.
C.1 LOGGING IN

When first starting up both the app or the web interface, a user will see a log in screen [24 and 36]. Here they can either log in with their (correct) credentials, or click a "Forgot Password" button that will bring them to a screen where they can request a new password. The normal behavior would be to log in with the right credentials, after which any user in the web interface or app will find themselves on the feed as can be seen in Figure 26 and Figure 39. This feed place stands for both the feed place in the model of the app (in C2) and the feed place in the model of the web interface (C3). The Login Screen place stands for both Log In Screen places in the model in C2 and C3. As seen the Log In Screen place has an input tag. This is there to connect with the log out of the app and web models. The web and app models both have an input tag at their Feed Page places that takes tokens from the output place in the Log In model.

When a user presses the Forgot Password button, they will be shown the forgot password screen [Figure 37 and Figure 25]. Here they can fill in their email address and they will receive an email with a token which they can then use to create a new password. After creating a new password the user will be brought back to the Log In Screen in both the app and the web interface.

FIGURE 47: LOGIN TRANSITIONS
C.2 APP TRANSITIONS

The application model has some color coding for each specific chain of action taken by the user to keep it more clear. After logging in a user enters this model (the app) in the Feed Page place, as seen from the description of logging in in C1, and the input tag on that place.

When in the feed a user can do several things, next to simply scrolling through the feed and reacting on any feed items that show up. Reacting and scrolling are not modelled as it does not change the view for most of the feed items and to have a more generalized model. Any user can use the filter by pressing the filter button on the top right hand of the screen [Figure 26]. When this button is pressed, a list of filters (checkboxes) is displayed which can be used to check and uncheck to respectively show or hide these type of items from the feed [Figure 30]. The user can then close this drop-down by pressing the filter icon again, or by the two general closing ways as described in the introduction of appendix C. A coach user can also create new feed items by pressing the plus button in the coach bar [Figure 26]. A small drop-up list [Figure 27] will show in which the coach can then select what type of item to create. When selecting an item type the drop-up list will disappear and a pop-up appears on the screen with fields needed to create that specific item type [Figure 28]. After filling in the required fields...
the coach can then press the "Post" button, which will post the item and return the coach to the feed.

From most screens in the application a user can press the menu button in the top left corner which will open up the sidemenu [Figure 31] in which they can select the following pages: "Feed", "Profile", "Settings", "Chat", and "Log Out". The only screens from which a user cannot press the menu button are the Log In Screen and while looking at the drop-up list and the pop-up, as they first have to close those. Pressing this menu button is modelled by the six small transitions (labeled 1,2,3,4,5,6).

When pressing 'Profile' in the sidemenu the user will be brought to his profile page [Figure 35], where he can change his password and update some user information. Updating the information will not take the user to a new page (and is subject to change) so it has not been modelled. Change password can be done by filling in the three required fields and then pressing the "Change Password" button which will then change the users password and close the profile page, bringing the user back to the feed.

When pressing 'Settings' in the sidemenu the user will be brought to his settings page where he can change some of the settings in the application. This can be done simply by changing any of the settings, which will be auto saved. The user can then leave this screen by pressing the menu button again and select a different item from the sidemenu.

When pressing 'Chat' in the sidemenu the user will be brought to the chat page [Figure 32] where they can select existing chats with users or in the case of a coach user they can also create a new chat. Player users can only view the chats they have had with coaches and reply on them by pressing an existing chat which will bring them to the chat with that coach. Coach users can do the same, but to create a chat they can press the plus button in the top right hand corner. This will bring them to the chat selection screen [Figure 33], which shows a list of users they can start a chat with. When the coach then presses any of these users he will be brought into the chat page [Figure 34]. Any user that can reach these chat pages can then leave by pressing the menu button and selecting a different item from the sidemenu.

When pressing 'Log Out' a user is logged out and gets brought back to the Log In Screen [Figure 24], and the Petri net ends up in the Log In model again, which is described in C1.
The web interface model has some color coding for each specific chain of action taken by the PR user to keep it more clear. After logging in a PR user enters this model (the web interface) in the Feed Page place, as seen from the description of logging in in C1, and the input tag at the Feed Page place. Leaving any page in the web interface can be done either by pressing backspace, pressing the back button of a web browser, or by selecting a different page from the sidemenu [Figure 38]. The sidemenu has been modelled the same way as the sidemenu in the application, and hovering over the sidemenu is represented by the seven small transitions (labeled 1, 2, 3, 4, 5, 6, 7).

The feed page [Figure 39] allows PR users to edit, accept/deny, or delete feed items that need regulation (e.g. a Heroes story). The PR user can leave this page by hovering over the sidemenu.
(modelled by the ‘Sidemenu Hovered’ place) and then clicking on any of the six buttons in the
sidemenu.
When the PR user presses the Club Settings button he gets into the settings page, where the
club info can be updated and this new info can then be saved by pressing the Save button on
the bottom of the page [Figure 41].
When the PR user presses the Profile button he gets taken to the profile page [Figure 42]. On
this page he can change his password by filling in the form and pressing the update password
button. The PR user can also update some personal information by filling out that form (chang-
ing the values in that form) and then press the update profile button.
When the PR user presses the club betting button he will be brought into the betting page [Fig-
ure 40], which in the future will house functionality to set up a betting pool.
When the PR user presses the members button they will be brought into the members page
[Figure 43]. This page houses the most functionality for the web interface, and is arguably the
most important page of the web interface. Here the PR user can add members (accounts) to
ClubNet by pressing the add member button, which will open an add member page [Figure 44]
where the PR user can then fill in a form to add a member. By submitting the form (pressing
the add member button at the bottom) an account will be added to the database and the PR
user returns to the members page. A PR user can delete accounts of members (only for players
and general members) by pressing the trash can icon (delete button)[Figure 46]. A PR user can
also edit accounts by pressing the edit icon (button), which will bring him into the edit mem-
ber page[Figure 45]. Here the information of the selected account can be changed, and saved
by pressing the save changes button, which will bring the PR user back to the members page
again.
When pressing the Log Out button the PR user is logged out and gets brought back to the Log In
Screen [Figure 36], and the Petri net ends up in the Log In model again, which is described in C1.