Abstract

This document contains the architectural design of the ClubNet software system, which is developed by The Brofessionals development team. The architecture satisfies the requirements in the Software Requirements Document (SRD) and the requirements in the User Requirements Document [1]. This document complies with the Software Engineering Standard, as specified by the European Space Agency [6].
## Contents

1 Introduction .................................................. 8  
  1.1 Purpose .................................................. 8  
  1.2 Scope ..................................................... 8  
  1.3 Definitions and Abbreviations .......................... 9  
    1.3.1 Definitions ........................................... 9  
    1.3.2 Abbreviations .................................... 10  
  1.4 List of References ...................................... 10  
  1.5 Overview ................................................ 11  

2 System Overview ............................................. 12  
  2.1 Background .............................................. 12  
  2.2 Basic Design and Context ............................... 12  
  2.3 Design Decisions ......................................... 13  
    2.3.1 Frameworks ......................................... 13  
    2.3.2 MongoDB ........................................... 15  
    2.3.3 Model-View-Controller Pattern .................... 15  
    2.3.4 Client-Server Style ............................... 16  
    2.3.5 Modularization .................................. 18  
    2.3.6 Back-End ........................................... 18  
    2.3.7 Front-End .......................................... 18  

3 System Context ............................................... 20  
  3.1 Retrieving Coach Accounts .............................. 20  
    3.1.1 Implementation of the Interface ................ 20  
  3.2 Retrieving Training Data ................................ 20  
    3.2.1 Implementation of the Interface ................ 21  
  3.3 Matches in a Season ................................... 21  
    3.3.1 Implementation of the Interface ................ 21  

4 System Design ............................................... 22  
  4.1 Design Method .......................................... 22  
  4.2 Decomposition Description ............................ 22  
    4.2.1 ClubNet Back-End ................................ 22  
    4.2.2 ClubNet App Front-End ......................... 24  
    4.2.3 File Structure .................................... 26  
    4.2.4 Database Design ................................ 28
5 Component Descriptions

5.1 Back-End

5.1.1 Database
5.1.2 Database API
5.1.3 Access control
5.1.4 Accounts API
5.1.5 Authentication API
5.1.6 Chat
5.1.7 Exercise Voting
5.1.8 Practicality
5.1.9 Feed

5.2 ClubNet App Front-End

5.2.1 Settings
5.2.2 Chat
5.2.3 Profile
5.2.4 Side Menu
5.2.5 Access Control
5.2.6 Login
5.2.7 Enrollment
5.2.8 Feed
5.2.9 Feed Item

5.3 Web Interface Front-End

5.3.1 Registration
5.3.2 Accounts
5.3.3 Club Setting

6 Feasibility and Resource Estimates

6.0.1 Resource requirements
6.0.2 Performance

7 Requirements Traceability Matrix

7.1 SR to Components
7.2 Components to SR

TU/e ClubNet | Architectural Design Document
DOCUMENT STATUS SHEET

GENERAL

Document title: Architectural Design Document v1.0.0
Identification: ADD/1.0.0
Authors: S.Chen, T. Sostak, K. Verhaegh, K. van Eenige
Document status: Final version
<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Author(s)</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0.1</td>
<td>02-05-2016</td>
<td>T. Sostak, K. Verhaegh</td>
<td>Initial document structure + ch 1</td>
</tr>
<tr>
<td>0.0.2</td>
<td>04-05-2016</td>
<td>S. Chen</td>
<td>Chapter 2 draft</td>
</tr>
<tr>
<td>0.0.3</td>
<td>06-05-2016</td>
<td>J. Brouns, S. Chen, K. van Eenige</td>
<td>Improved Chapter 2</td>
</tr>
<tr>
<td>0.0.4</td>
<td>10-05-2016</td>
<td>S. Chen, K. van Eenige</td>
<td>Chapter 3 draft</td>
</tr>
<tr>
<td>0.0.5</td>
<td>12-05-2016</td>
<td>S. Chen</td>
<td>Chapter 4 draft</td>
</tr>
<tr>
<td>0.0.7</td>
<td>17-05-2016</td>
<td>S. Chen, T. Sostak</td>
<td>Chapter 5 draft</td>
</tr>
<tr>
<td>0.0.8</td>
<td>23-05-2016</td>
<td>S. Chen, T. Sostak</td>
<td>Improved Chapter 5</td>
</tr>
<tr>
<td>0.0.9</td>
<td>23-05-2016</td>
<td>S. Chen</td>
<td>Improved Chapter 5</td>
</tr>
<tr>
<td>0.1.0</td>
<td>25-05-2016</td>
<td>S. Chen</td>
<td>Improved Chapter 3 &amp; 4</td>
</tr>
<tr>
<td>0.1.1</td>
<td>27-05-2016</td>
<td>S. Chen</td>
<td>Chapter 6 draft</td>
</tr>
<tr>
<td>0.1.2</td>
<td>02-06-2016</td>
<td>S. Chen</td>
<td>Chapter 7 draft</td>
</tr>
<tr>
<td>0.1.3</td>
<td>07-06-2016</td>
<td>S. Chen</td>
<td>Improved Chapter 7</td>
</tr>
<tr>
<td>1.0.0</td>
<td>13-06-2016</td>
<td>S. Chen</td>
<td>Finished version</td>
</tr>
</tbody>
</table>
DOCUMENT CHANGE RECORDS

GENERAL

Date: 27-05-2016
Document Title: Architectural Design Document
Document Identifier: ADD/1.0.0

CHANGES

<table>
<thead>
<tr>
<th>Section</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter 6</td>
<td>Created separate section for performance</td>
</tr>
</tbody>
</table>
1 INTRODUCTION

1.1 PURPOSE

The Architectural Design Document (ADD) contains the architectural design for the ClubNet system that will be implemented by the Brofessionals development team. The document describes the decomposition of the software into separate components. For each component it describes the its interfaces and dependencies to any other components and which software requirements from the SRD[2] they fulfill. It also describes a short overview and the context of the system, and an estimation of the feasibility and resource requirements are given.

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 at football clubs in organizing training 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 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 all type of sports in the future.
# 1.3 DEFINITIONS AND ABBREVIATIONS

## 1.3.1 DEFINITIONS

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</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 [8]</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. [7]</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 subcribed 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 Technolo-</td>
<td>A software engineering company situated in the Netherlands serving the</td>
</tr>
<tr>
<td>gies B.V.</td>
<td>role of client.</td>
</tr>
<tr>
<td>Ionic</td>
<td>A complete open-source SDK for hybrid mobile app development [9]</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 [10]</td>
</tr>
<tr>
<td>MongoDB</td>
<td>A cross-platform NoSQL database [11]</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>
<tr>
<td>SRD</td>
<td>Software Requirements Document</td>
</tr>
<tr>
<td>ADD</td>
<td>Architectural Design Document</td>
</tr>
<tr>
<td>DDD</td>
<td>Detailed Design Document</td>
</tr>
</tbody>
</table>

1.4 LIST OF REFERENCES

References


1.5 OVERVIEW

The remainder of this document consists of six chapters. Chapter 2 gives a short overview of the system with an introduction to the context and design of the system, as well as the background of the project. Chapter 3 describes the relation with each external interface the system uses. Chapter 4 gives the design of the system including the design method used and the description of the decomposition of the system. Each component is then further described in detail in chapter 5. Chapter 6 gives the estimation of the feasibility and the computer resources needed to build, operate, and maintain the software. Chapter 7 gives the requirements traceability matrix which shows how each software requirement of the SRD[2] is linked to the components described in previous chapters of the ADD.
2 SYSTEM OVERVIEW

The ClubNet system is a software system. The system contains a mobile app and a web interface that will both be used by the end users. The system provides interfaces that can be used by the mobile app and the web interface to use shared functionality and data.

For a description of ClubNet, see the User Requirements Document (URD) [1]. For a description of the relevant background of the ClubNet system and the environment in which the system will operate, see sections 2.3 and 2.4 of the Software Requirements Document (SRD) [2] and section 2.5 of the URD [1].

2.1 BACKGROUND

The ClubNet system is conceived by Intuitive Technologies B.V. and developed by The Professionals. The ClubNet system aims to improve the communication between members within amateur football clubs by providing a user-friendly app and a web interface. The web interface will be used exclusively by the club’s PR managers for management purposes. The ClubNet app is meant to be used by all other members in the club.

The ClubNet system has no preceding project. However, it is related to an external system, namely CoachAssist. CoachAssist is developed by Intuitive Technologies B.V. and aims at facilitating coaches in managing and creating training schedules. The ClubNet system will retrieve coach accounts and training schedule related information from CoachAssist.

2.2 BASIC DESIGN AND CONTEXT

The ClubNet system consists of the ClubNet app and a web interface. The ClubNet app is stored and runs locally on the mobile device of a user and is connected to the ClubNet server. Only coaches, players and general club members will use the ClubNet app. The web interface runs locally in the web browser of a user and is connected to the same server. Only the PR user will use the web interface.

The ClubNet server stores the data needed to support the functionality of the ClubNet system in a database. It also stores the files needed to communicate with other components. The ClubNet app and web interface will not have direct access to the data stored in the ClubNet server. Instead, the ClubNet server handles the requests and modifies the data independently.

The CoachAssist server will store a part of the data that is needed for the correct functioning of the ClubNet system. Thus it will be connected by the ClubNet server to retrieve data.

The basic overview of the system can be found in Figure 1.
2.3 DESIGN DECISIONS

In this section we will explain the important design decisions we made when choosing the technologies and the general design principles of the system.

2.3.1 FRAMEWORKS

This section discusses the technology that is used in the development of the app and web interface for both front-end and the back-end along with their possible alternatives.

Meteor: Meteor is a free and open-source JavaScript web application development framework written using Node.js. Meteor has the advantages of rapid prototyping and producing cross-platform (web, Android, iOS) code. The publish–subscribe pattern of Meteor automatically propagates data changes to clients without requiring the developer to write any synchronization code. The ClubNet system uses Meteor for the back-end implementation to utilize its real-time data propagation. By using Meteor, we can also save much work in handling the connections with front-ends and the Node.js server. There are several alternatives for the back-end implementation, as listed below.

• MEAN stack: The Mean stack is essentially a bundle of MongoDB, Express, AngularJS and Node.js. Unlike Meteor, the developers need to implement all basic communication between those mentioned components manually. This gives much freedom to the developers to create more complex system. Meteor on the opposite, gives these for free. The ClubNet system has a rather simple design and functionality. Meteor is chosen for fast development and easy maintenance.
• Sails: Sails is a classic MVC (Model-View-Controller) framework. It has an active development community and many useful available plug-ins. Sails does not restrict the choice of database, storage or front-end. However, comparing to Meteor, it is a rather massive framework that is ideal to develop more complex systems. In contrast, Meteor is a more light-weighted framework with more basic functionality available. Besides, Meteor itself supports automatic and reactive data transfer and has integration with a rich MongoDB library.

Comparing to other alternatives, Meteor is considerably light-weighted while also providing rich and useful functionality for the reactive data transfer and storage. Besides the advantages Meteor has, the client, Intuitive Technologies.B.V., has the preference of using Meteor as the framework to use, since the already existing CoachAssist system uses it as well.

Ionic: Ionic is a complete open-source SDK for hybrid mobile app development. It is built on top of AngularJS and Apache Cordova. Apps that are developed using Ionic framework can be distributed through native app stores to be installed on devices by leveraging Cordova. Ionic is used for the development of front-end implementation to have a single-code base for the ClubNet app and deploy it on both iOS and Android platforms. There is one alternative that is also widely used for hybrid mobile app development.

• React Native: React Native is a framework for building native apps using React. Unlike Ionic, it uses the React component model to render to native views. This brings a lot of advantages: the apps built using React native use less memory and thus run more smoothly. However, React native is more difficult to set up and the developers of The Brofessionals need a learning period before getting used to it. On the contrast, Ionic is rather easy to set up and easy to use. Ionic uses standard web technologies including HTML, CSS and AngularJS, which are well mastered by The Brofessionals.

Taking the experiences of the developers and time limit into consideration, Ionic is chosen to achieve fast development. The performance difference between using Ionic and React native is minimum since the ClubNet app does not have complex functionality other than data binding and API calls to back-end.

Bootstrap UI: Bootstrap is a free and open-source front-end library for creating websites and web applications. It contains HTML- and CSS-based design templates for typography, forms, buttons, navigation and other interface components, as well as optional JavaScript extensions. It aims to ease the development of dynamic websites and web applications. There are several more alternatives for the Bootstrap UI but the main known competitor is the Foundation framework.
• Foundation: Foundation is a responsive front-end framework, providing responsive grid, UI elements and templates for typography, forms, buttons, navigation and other components. The main advantages of the Foundation is that it is more novel then all other front-end frameworks and it has a more sophisticated grid system. The latter feature is not very relevant for the ClubNet web interface, however novelty is always better as long as it stays in stable and usable version.

In general, there is not much difference between the frameworks and they are both sophisticated enough to be used in this kind of application. Bootstrap contains boiler-plated code that can be used by a developer to create out-of-the-box user interface elements. In this way, Bootstrap improves rapid development. Bootstrap UI was chosen to deliver the best possible UI experience since most of the The Broffesionals team members already were familiar with the framework and it was also the client’s preference as this framework is also used in his other application CoachAssist.

2.3.2 MONGODB

Meteor uses MongoDB, a NoSQL database for data storage. MongoDB is a free and open-source cross-platform document-oriented database. MongoDB uses JSON-like documents with dynamic schemas to store data in certain types of applications easier and faster. We chose MongoDB as our database because Meteor natively supports it. The ability of storing unstructured data also makes ClubNet system scalable and easy to add future functionality.

There is no significant advantage of choosing MongoDB over traditional relational database like MySQL. Both databases would serve the functional needs of ClubNet system well. However, the client, IntuitiveTechnologies B.V. has the preference of using MongoDB to easily integrate ClubNet system to CoachAssist in the future.

2.3.3 MODEL-VIEW-CONTROLLER PATTERN

As described before, the ClubNet system uses the Ionic framework for front-end development. Ionic builds on top of AngularJS to create a powerful SDK well-suited for building rich and robust mobile apps for the app store and the mobile web. AngularJS is a Model-View-Controller (MVC) based framework. By using Ionic, we fixed the architectural pattern of the user interface implementation to the MVC pattern.

The MVC pattern contains three interconnected parts, as listed below:

• Model: Responsible for the representation of the data. The model may also contain state information.

• View: Represents the visualization of the data that the model contains.

• Controller: Link between the end-user and the system. It receives user input from the view and makes calls to the system by notifying the model.
The MVC pattern achieves the separation between information, presentation and user interaction. When a model object value changes, a notification is sent to the view and to the controller. The view object will then update itself and the controller modifies the view object if the logic requires so. The user input is sent to the controllers and if a change is required, the controller then updates the model.

The diagram illustrating the MVC architectural pattern that will be used in the ClubNet system is shown in Figure 2.

![MVC Diagram](image)

**FIGURE 2: MODEL-VIEW-CONTROLLER**

### 2.3.4 CLIENT-SERVER STYLE

The intended use of the application is characterized by the multiple clients and one server, to this end the client-server architectural style is used in the design of ClubNet system. There are two layers, one for the client front-ends and one for the back-end server. These two layers will communicate with each other through the Internet. On the client side there is no processing beyond data bindings, while all processing is done on the server side. The connector of these two layers is the back-end procedure calls.

The illustration of the client-server architectural style can be found in Figure 3.
There is also an alternative architectural style: Peer-to-peer. By contrast, in the peer-to-peer style, every client can function as both client and server. The peer-to-peer architectural style has a distributed application architecture that partitions tasks and resources between peers of clients. It has the advantage of utilizing the processing power and storage space of all clients without the need of the central coordination by a dedicated server. However, this architectural style does not fit the ClubNet system well. The ClubNet system does not need to do intensive data processing, thus the distributed processing network is not needed. A distributed data storage makes it difficult to achieve data integrity and centralized management of data.

With the client-server architectural style, we can achieve the data centralization which is important for data integrity. This also enables simultaneous data access from multiple front-ends and gives a clear separation of front-end and back-end. The disadvantages of this architectural style is that there is a single point of failure and the performance of the system is limited to the network bandwidth (amount of requests). The prior disadvantage can be made up for by setting up data recovery procedures and the latter disadvantage is insignificant considering the limited amount of users.
2.3.5 MODULARIZATION

Since the back-end is implemented using Meteor, there are many functionalities that come for free. These functionalities can be organized as different modules in the ClubNet system. However, we decided not to modularize these functionalities in the system since the interfaces are available directly from Meteor. Developers can reuse them by simply making a call to the existing Meteor functions. In particular, the functionality listed below will not be modularized:

- Account management: Meteor has a set of built-in functions for account management. They include creating, deleting, updating and viewing user accounts.
- Reactive data transfer: This functionality is used to support the real-time communication between front-end and back-end. Any update on the back-end will automatically propagate to front-end. This is supported by the built-in publish and subscribe functions.

The details about the modularization of the system can be found in System Design section.

2.3.6 BACK-END

The back-end server is driven by Meteor, as described before. The back-end server is where the database of ClubNet system is deployed. There is only one central database that will provide data for all front-ends. A centralized database is preferred over a distributed database to achieve ease of maintenance and data integrity. A centralized database is also easier for different end-users to use due to the simplicity of having a single database design. Any update made to the database can be available to all end-users immediately upon querying.

To avoid unintended data modification, the database will only be accessed by the back-end server itself. To retrieve or modify data, an API call to the server has to be made by the front-end. The back-end server is also responsible for executing the business logic to respond to all user events. This means that all data computations will be done only on the server. This prevents the front-end from executing false logic that leads to false requests of data modification. All API calls to external systems will also be centrally managed by the back-end server.

2.3.7 FRONT-END

The front-end consists of two parts, the ClubNet app and the web interface. The ClubNet app is developed using Ionic, as described before. The web interface is developed using Bootstrap UI framework (which includes AngularJS). These two parts will be deployed on the same back-end server together for easy maintenance. To achieve the requirement of a single-code base, the code of these two parts are stored together in one project development folder.

The front-end is where the data presentation and user events happen. For the sake of maintenance and simple design, the front-end should be lightweight. It should be only responsible for data binding and informing the back-end server about user events, such as logging in, refreshing a page, editing information etc.. The back-end server will then process the data gathered.
from the front-end and give responses accordingly. The separation of the data binding and data manipulation delivers a simpler design of the system and achieves high maintainability. The business logic and rules can be changed centrally on the server and the data presentation in all front-ends will be updated automatically.

The system is designed with the assumption of unreliable front-ends. That means the front-ends might tend to break the security and data integrity of the system. This is solved by allowing front-ends only to send requests for data retrieval and modification and by only letting the back-end server validate and respond to the requests.
3 SYSTEM CONTEXT

This chapter describes the connections the ClubNet system has with external systems. The ClubNet system needs a set of interfaces to connect to the CoachAssist server. These interfaces will be used to retrieve information related to coach accounts and training schedules. The ClubNet system also needs the interface for loading the matches in a season.

3.1 RETRIEVING COACH ACCOUNTS

The ClubNet back-end server should be able to retrieve data of coach user accounts from the CoachAssist system. The data retrieval is done via API calls to the CoachAssist server and the CoachAssist server returns JavaScript objects that stores the data in JSON format. The information of a coach user includes the following attributes:

- _id: String. The unique identifier of the coach
- firstName: String. The first name of the coach
- lastName: String. The last name of the coach
- club: String. The name of the club which the coach belongs to
- dateOfBirth: Date. The birth date of the coach
- team: String. The name of the team the coach belongs to
- createdAt: Date. The date at which the coach account is created

3.1.1 IMPLEMENTATION OF THE INTERFACE

This interface will not be developed by The Brofessionals and is not part of the ClubNet system. Instead, the client, IntuitiveTechnologies B.V. will implement this interface. The ClubNet system only needs to make an API call to retrieve the data. Each API call requires the club ID as input and returns all coach accounts in that club.

Due to the delay of the API implementation by IntuitiveTechnologies B.V., The Brofessionals uses the coach accounts created in the ClubNet database directly for development instead of calling the API.

3.2 RETRIEVING TRAINING DATA

The ClubNet back-end server should be able to retrieve data of the training schedules of a team from the CoachAssist system. The data retrieval is done via API calls to the CoachAssist server and the CoachAssist server returns JavaScript objects that store the data in JSON format. The information related to training schedules includes the following attributes:
• _id: String. The unique identifier of the training.
• date: Date. The date at which the training takes place.
• exercises: An array of exercises. Each element has the following attributes:
  – _id: String. The unique identifier of the exercise.
  – name: String. The name of the exercise.
  – icon: String. The URL to the icon of the exercise.

### 3.2.1 IMPLEMENTATION OF THE INTERFACE

The interfaces will not be developed by The Brofessionals and is not part of the ClubNet system. Instead, the client, IntuitiveTechnologies B.V. will implement this interface. The ClubNet system only needs to make an API call to retrieve the data. Each API call requires the club ID and team ID and returns all existing training schedules of that team.

Due to the delay of the API implementation by IntuitiveTechnologies BV., the Brofessionals uses the training schedule related data which is created in the ClubNet database directly for development instead of calling the API.

### 3.3 MATCHES IN A SEASON

The ClubNet system will have the functionality that supports the club betting in a club. To this end, all the matches that can be bet on in a season should be provided. The client, IntuitiveTechnologies B.V. will provide this information. The information about the matches will be stored in a .xlsx formatted file. The .xlsx file will contain a table in which each row stores the information about a specific match. From left to right, each column will contain the following attributes: match ID, date, home team name, away team name, goals of home team, goals of away team. A example of the content of the file is shown in Figure 4.

![Figure 4: Sample Match Data](image)

<table>
<thead>
<tr>
<th>Match ID</th>
<th>Date</th>
<th>Home Team</th>
<th>Away Team</th>
<th>Goals Home</th>
<th>Goals Away</th>
</tr>
</thead>
<tbody>
<tr>
<td>54454</td>
<td>9/26/15</td>
<td>Rood Wit’62 F1</td>
<td>Rhode F1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>54435</td>
<td>9/26/15</td>
<td>Mierlo Hout F1</td>
<td>Avanti’31 F1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>54412</td>
<td>9/26/15</td>
<td>Blauw Geel’38/JUMBO F1</td>
<td>Deurne F1</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>54459</td>
<td>9/26/15</td>
<td>Schijndel/DE WIT F1</td>
<td>Gemert F1</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>60697</td>
<td>9/26/15</td>
<td>Liesel F1G</td>
<td>MULO F2</td>
<td>8</td>
<td>2</td>
</tr>
</tbody>
</table>

### 3.3.1 IMPLEMENTATION OF THE INTERFACE

On the back-end server, a dedicated function will be used to load the .xlsx file and interpret the data. This function will be called when the file is uploaded from the web interface.

Due to the limited time, this interface is not implemented and thus the functionality is not supported by the current version of the ClubNet system.
4  SYSTEM DESIGN

This chapter describes the technical aspects of the design of the ClubNet system. Specifically, the design method and system decomposition with actual components are described in this chapter.

4.1  DESIGN METHOD

Following the client-server architectural style, the front-end and back-end are designed separately. The front-end is responsible for data-binding and informing user events. The back-end is responsible for database storage and access, user events handling, external API calls and computation.

The back-end is designed with the modularized style in mind. The modularization is based on the classification of functionality to achieve high cohesion and reusability. The coupling between modules is minimized to achieve high maintainability. To ensure data integrity and privacy, all direct database access should be centralized. The design of each module should achieve high scalability.

The front-end is designed according to the AngularJS MVC pattern. Every front-end feature should be modularized. The modularization should aim for high maintainability and scalability.

4.2  DECOMPOSITION DESCRIPTION

The decomposition of the ClubNet system into components is based on the requirements of the URD[1] and the SRD[2]. Section 4.2.1 lists the components in the back-end server and section 4.2.2 lists the components in the front-end. The details of these components are further described in chapter 5.

4.2.1  CLUBNET BACK-END

The ClubNet back-end has the following components:

- **Database**: The database component contains the data model and stores the data in MongoDB according to it. MongoDB provides interfaces for direct data insertion, updating, retrieving, and deletion.

- **Database API**: This component contains all interfaces that are used to retrieve, insert, update and delete the data in the MongoDB collections. The different between the Database API and MongoDB interfaces is that the Database API includes data processing before the direct database modification. These interfaces will be called by server-side components.
• **Chat**: This component implements the functionality and interfaces that are used for chat functionality.

• **Authentication API**: This component provides the interfaces that are used for logging in and out. It uses the built-in Meteor authentication methods for the direct logging in and out before executing application specific procedures.

• **Accounts API**: This component provides the interfaces for account management including creation, deletion, and updating. It uses the built-in Meteor account management methods for direct account creation, deletion and updating before executing application specific procedures.

• **Access control**: This component is responsible for checking the access rights of each user. It controls whether a user can create, view, edit and delete feed items of a certain type or a chat session.

• **Feed**: This component implements the functionality and interfaces that are related to the general functionality of feed items such as adding note to and stickying a feed item.

• **Exercise voting**: This component implements functionality and interfaces that are related to the exercise voting functionality.

• **Suggestion**: This component implements functionality and interfaces that are related to the exercise suggestion functionality.

• **Practicalities**: This component implements functionality and interfaces that are related to the practicalities functionality.

• **Club betting**: This component implements functionality and interfaces that are related to the club betting functionality.

• **Heroes**: This component implements functionality and interfaces that are related to the heroes functionality.

• **Sponsoring**: This component implements functionality and interfaces that are related to the sponsoring functionality.

Except the database component, each components need certain functionality in one or more other components to achieve its own functionality. The dependency relation is shown in Figure 5. An arrow from component A to B indicates that A requires certain functionality of B.
Except the Database component, all other components need certain database access and thus require the Database API component. The Database API requires Access control component for checking whether a database access can take place. The Chat and Feed components also require Access control component to check whether the user has the rights to perform certain operations.

4.2.2 CLUBNET APP FRONT-END

The ClubNet app has the following components for the front-end functionality:

- **Login**: This component contains the model, view, controller, and services for logging in/out and resetting a password.
- **Profile**: This component contains the model, view, controller, and services for displaying and changing profile details.
- **Setting**: This component contains the model, view, controller, and services for displaying and changing settings.
- **Feed**: This component contains the model, view, controller, and services for the feed.
- **Feed item**: This component contains the model, view, controller, and services for the displaying and editing the information of a single feed item.
- **Chat**: This component contains the model, view, controller, and services for the chatting functionality.
• **Access control**: This component contains the model, view, controller, and services for the access control functionality.

• **Side menu**: This component contains the model, view, controller, and services for the displaying the side menu.

• **Enrollment**: This component contains the model, view, controller, and services for the enrollment functionality.

The web interface has the following components for the front-end functionality:

• **Login**: This component contains the model, view, controller, and services for logging in-/out and retrieving a lost password.

• **Accounts**: This component contains the model, view, controller, and services for accounts management.

• **Profile**: This component contains the model, view, controller, and services for displaying and editing the profile details.

• **Setting**: This component contains the model, view, controller, and services for displaying and changing the settings.

• **Feed**: This component contains the model, view, controller, and services for the feed.

• **Feed item**: This component contains the model, view, controller, and services for displaying and editing the information of a single feed item.

• **Registration**: This component contains the model, view, controller, and services for account registration.

• **Betting pool**: This component contains the model, view, controller, and services for the creation and editing of the betting pool.

• **Club settings**: This component contains the model, view, controller, and services for the club settings.

Except the Enrollment and Login components, each components need certain functionality in one or more other components to achieve its own functionality. The dependency relation is shown in Figure 5. An arrow from component A to B indicates that A requires certain functionality of B.
4.2.3 FILE STRUCTURE

The development project should have the following file structure:
/...
[client] ........................................................................................................ Main app directory
  [app] ........................................................................................................ App front-end source code
    [css] ...................................................................................................... App CSS files
    [js]
    [controllers] ......................................................................................... App controllers
    [services] ............................................................................................... App services
    config.js
    routes.js
    app.js
    directives.js ......................................................................................... App directives
  [lib] ........................................................................................................... Ionic and Cordova libraries
    [views] .................................................................................................. App HTML files
  [web] ........................................................................................................ Web front-end source code
    [js]
    [controllers] ......................................................................................... Web controllers
    [services] ............................................................................................... Web services
    web.js
    [views] .................................................................................................. Web HTML files
  index.html
    main.js
[public] ..................................................................................................... Public files
  [app] ........................................................................................................ App public files
    [css]
    [fonts]
    [img]
  [web] ........................................................................................................ Web public files
    [css]
    [img]
    [exercise_images] ..................................................................................... Public images
    [feed_icons] ............................................................................................ Public icons
[server] ........................................................................................................ Server file directory
  [Database]
  [Schemes] .................................................................................................. Database schemes
  [DatabaseAPI]
  [AccountsAPI]
  [AuthenticationAPI]
  [Chat]
  [AccessControl]
  [Sponsoring]
  [ExerciseVoting]
  [Heroes]
  [Practicality]
  [Suggestion]
  [Heroes]
  [Feed]
  config.push.json ....................................................................................... Push-notification configuration
  mobile-config.json .................................................................................... App configuration
4.2.4 DATABASE DESIGN

As we will use a MongoDB database, the data will be stored in different collections. A collection in MongoDB is equivalent to a table in a relational database. Unlike MySQL, MongoDB itself does not enforce any rigid database schema. This gives us the possibility to insert documents that have different schemas to one collection. For example, instead of having four different tables for coach, player, general club member and PR users, we can have only one user collection that stores the data of the users of all types. The list of collections are shown below.

- **Clubs**: A collection that stores the information of each club.
- **User**: A collection that stores user accounts of all types.
- **FeedItems**: A collection that stores the feed items of all types.
- **Responses**: A collection that stores the responses to each feed item.
- **Accesses**: A collection that stores the access matrix of all user types.
- **Chats**: A collection that stores the information of all chat sessions.
- **Messages**: A collection that stores the chat history of all chat sessions.
- **FeedItemTypes**: A collection that stores the information of all feed item types.

An important feature of MongoDB is that a document can be embedded in another document, which creates a tree structure. This embedded document feature provides fast access to data and eliminates the need of expensive join and reference operations.

Notes and betting results are modelled as embedded documents since they need only be accessed via a user account and need no complex computation other than read and write. On the contrary, responses are modelled as a separate collection since they need to be accessed via both a user or a feed item. If they are modelled as embedded documents, it would be difficult to either query all responses of a feed item or of a user. It is also impossible to perform sorting on embedded documents. The same reasoning goes for the access matrix, exercises, matches and chat history. The embedded documents and their parent collection are listed below.

- **Users**: Notes (in user of type coach), betting results.
- **Access**: Access matrix.
- **FeedItems**: Exercises (in feed item of type ExercisePoll), matches (in feed item of type BettingRound).
- **Chat**: Messages.
The schemes that are applied to each collection can be found in Figure 8. Each collection contains one or more schemes. The lines that connect a pair of two attributes represent the reference relation in MongoDB. The detailed implementation of the database schemes can be found in the Detailed Design Document[3].

FIGURE 8: DATABASE SCHEMES
5 COMPONENT DESCRIPTIONS

This chapter describes the components mentioned in section 4.2 Decomposition description. References to relevant user requirements and software requirements are given for each component. The Club betting, Suggestion, and Sponsoring will not be implemented, and thus will not be described in this section.

5.1 BACK-END

5.1.1 DATABASE

Identifier
Database.

Type
Program.

Purpose

Function
The database is a standalone component that stores the data model and the data according to it.

Subordinates
This component has the following sub-components:

- Schemes: Schemes are the specification of the data model. They define the format of the stored data.

- MongoDB database: The MongoDB database stores the collections of data.

Dependencies
This component does not depend on any other component.

Interfaces
MongoDB and Meteor provide the interfaces for querying and modifying the data stored in the database.

- Insert
  - Input: The document to be inserted.
  - Output: None. The document is inserted to the database.

- Query
- Input: Query condition.
- Output: The documents that satisfying the query condition.

- **Update**
  - Input: Query condition, modification statements.
  - Output: None. The documents that satisfy the query condition are updated according to the modification statements.

- **Delete**
  - Input: Query condition.
  - Output: None. The documents that satisfy the query condition are deleted from the database.

**Resources**
The following resources are required:

- The Simple-Scheme Meteor package. For managing database schemes and validate the documents against them.
- The Collection2 Meteor package. For creating and modifying the MongoDB collections.

**References**
The ER-diagram can be found in the Software Requirements Document [2] section 2.7.4. The database design can be found in section 4.2.4. The details of the database schemes can be found in the Detailed Design Document [3] in the appendix.

**Processing**
Upon the deployment of the back-end server, the collections will be created in the MongoDB database and the schemes will be attached accordingly. After the one of the interfaces is called, direct data retrieval or modification takes place in the MongoDB database.

**Data**
This component stores the data of the ClubNet system.

### 5.1.2 DATABASE API

**Type**
Package.

**Purpose**
SR72, SR73, SR74, SR75, SR80, SR87, SR88, SR96, SR97, SR103, SR104, SR109, SR115, SR116,
June 30, 2016

**Function**
This component contains all the interfaces for the database access. It serves as a middle ware between the database and all other components that need to query, insert, update, or delete documents in the database.

**Subordinates**
This component has no subordinate.

**Dependencies**
The functions in this component will only be executed after one of the other components (except the Database component) has made a call to an interface of this component.

**Interfaces**
This component provides the following interfaces:

- Inserting a new document of a club, feed item, response, access matrix, feed item type or chat history
  - Input: The document to be inserted.
  - Output: The ID of the inserted document. If the insertion failed, an error message is thrown.

- Updating the document of a club, feed item, response, access matrix, feed item type or chat history
  - Input: A document contains the new attributes to update and the ID of the document to be updated.
  - Output: The updated document. If the updating failed, an error message is thrown.

- Querying for the documents in any collection
  - Input: The query condition.
  - Output: The documents that satisfy the query condition. If no document satisfied, a null object is returned.

- Deleting the documents in any collection
  - Input: The ID of the document to be deleted.
  - Output: The deleted document. If the deletion failed, an error message is thrown.

**Resources**
The following resource is required:

- Collection2 Meteor package: For direct database access.
References
The details of the implementation of this component can be found in the DDD [3] in the appendix.

Processing
After the interface for inserting a new document is called, the input document will first be cleaned and extended if needed. Then the Access control component will be executed to check whether the insertion is allowed. Finally the input document will be validated against the corresponding schema. Only when the schema validation succeeded, the document is inserted.

After the interface for updating a new document is called, the Access control component will be executed to first check whether the updating is allowed. Then the attributes to update will be validated against the corresponding schema. Only when the schema validation succeeded, the document is updated.

When the interfaces for querying and deleting are called, the Access control component will be executed to check whether the deletion is allowed. Only then the document is deleted.

Data
This component does not track or store any data.

5.1.3 ACCESS CONTROL

Type
Package.

Purpose

Function
The Access control component is responsible for checking whether a user has the right to perform an operation. There are four rights defined, namely, create, view, delete and edit. These four rights are applied on feed item types and chat session.

Subordinates
This component has no subordinate.

Dependencies
This component will only be executed after one of the following components has made a call to the interfaces of this component: Chat, Database API, and Feed component.

Interfaces
This component provides the following interfaces:

- Setting the access matrix of a user type
  - Input: Access matrix and user type.
  - Output: None.
• Checking the permission of an operation
  - Input: Operation type, subject of the operation and user type.
  - Output: True if the operation is allowed. False otherwise.

Resources
This component requires no external resource.

References
The details of the implementation of this component can be found in the DDD [3] in the appendix.

Processing
After one of the interfaces is called, the Database API component will be executed to retrieve or modify rights related data.

Data
This component does not track or store any data.

5.1.4 ACCOUNTS API

Type
Package.

Purpose

Function
This component is responsible for the account management including creating, deleting and updating.

Subordinates
This component has no subordinate.

Dependencies
This component does not depend on any other component.

Interfaces
This component provides the following interfaces:

• Creating a user account
  - Input: A document that stores the information of the new user.
  - Output: The ID of the created user. If the creation failed, an error message is thrown.

• Updating a user account
  - Input: The ID of the user and a document that contains the attributes to update.
  - Output: The document of the updated user. If the updating failed, an error message is thrown.
• Retrieving user information
  – Input: The ID of the user.
  – Output: The document of the user with the specified ID. If there is no user with the specified ID then a null object is returned.

Resources
The following resource is required:
  • Accounts-password Meteor package. For creating, deleting, and updating of an account.

Processing
After the interface for creating a user account is called, the information of the user account will first be extended to match the database scheme. Then accounts-password package will be used to first insert the new account and then send an email to the user to notify the registration. There is no complex processing for updating or retrieving user accounts.

Data
This component does not store or track any data.

5.1.5 AUTHENTICATION API

Type
Package.

Purpose
SR17, SR18, SR19, SR20.

Function
This component is responsible for the user logging in/out functionality.

Dependencies
This component does not depend on any other component.

Interfaces
This component provides the following interfaces:

• Log in
  – Input: The email and password.
  – Output: None if the logging in succeeded. Otherwise an error message is thrown.

• Log out
  – Input: None.
  – Output: None if the logging out succeeded. Otherwise an error message is thrown.

Resources
The following resource is required:
Accounts-password Meteor package: For direct logging in/out.

Processing
There is no complex processing other than simple method calls to the accounts-password package.

Data
This component does not store or track any data.

5.1.6 CHAT

Type
Package
Purpose

Function
This component implements the chatting functionality. It is responsible for creating a chat session and recording the chat history.

Dependencies
This component does not depend on any other component.

Interfaces
This component provides the following interfaces:

- Creating a chat.
  - Input: The IDs of the users in the chatting session.
  - Output: The ID of the newly created chat session.

- Sending a message.
  - Input: The addresser ID, the chat ID, and the message.
  - Output: None.

- Getting all chat sessions of a user.
  - Input: The user ID.
  - Output: All chat sessions associated with the user.

- Getting all messages of a chat session.
  - Input: The chat session ID.
  - Output: All messages in the chat sessions.
Resources
No external resource is needed.

Processing
After the interface for creating a chat is called, a chat session document is then created and inserted in the database.
There is no complex internal processing for the rest interfaces other than simple database access using Database API component.

Data
This component does not store or track any data.

5.1.7 EXERCISE VOTING

Type
Package.

Purpose

Function
This component is responsible for the functionality related to exercise voting user story. The functionality includes getting the voting result, getting the winner, and registering a new vote.

Dependencies
This component does not depend on any other component.

Interfaces
This component includes the following interfaces:

- Getting the voting result
  - Input: The ID of the voting poll feed item.
  - Output: An array of the number of votes of each exercise.

- Getting the winner
  - Input: The ID of the voting poll feed item.
  - Output: The ID of the winning exercise.

- Registering new vote
  - Input: The IDs of the voter, the exercise to vote, and the voting exercise feed item.
  - Output: None. The response is stored in the database.

Resources
This component requires no external resource.

Processing
After the interfaces for getting the voting result or getting the winner are called, the Database API component will be executed to retrieve the corresponding data. The retrieved data is then processed to get the output value.

After the interface for registering new vote is called, the input data is then first processed and the Database API component modifies the database accordingly.

Data
This component does not store or track any data.

5.1.8 PRACTICALITY

Type
Package.

Purpose
SR35, SR125, SR126, SR127, SR130.

Function
This component is responsible for the functionality related to practicality user story. The functionality includes registering a new contribution, withdrawing a contribution and getting the contributions of a practicality feed item.

Dependencies
This component does not depend on any component.

Interfaces
This component provides the following interfaces:

- Registering a new contribution
  - Input: The ID of the contributor and the amount of contribution.
  - Output: None. The new contribution is stored in the database.

- Withdrawing a contribution
  - Input: The ID of the contributor.
  - Output: None. The contribution of the contributor is deleted from the database.

- Getting the contributions
  - Input: The ID of the practicality feed item.
  - Output: An array of the information of each contribution. The information contains the ID of the contributor and the amount been contributed.

Resources
This component requires no external resource.

Processing
After the interfaces for registering a new contribution is called, the input data is then processed and the Database API component modifies the database accordingly.

After the interfaces for withdrawing a contribution and getting the contributions are called, the Database API component will retrieve corresponding data from the Database component. The retrieved data is then processed to get the output value.

**Data**
This component does not store or track any data.

### 5.1.9 Feed

**Type**
Package.

**Purpose**
SR46, SR32, SR39.

**Function**
The Feed component is responsible for the general functionality such as adding note and sticking a feed item.

**Subordinates**
None.

**Dependencies**
This component depends on the Database API component.

**Interfaces**
This component provides the following interfaces:

- **Adding a new note for a specific feed item**
  - Input: The note object to be added for an item.
  - Output: None.

- **Updating the note for a specific feed item**
  - Input: The note object to be updated for an item.
  - Output: None.

- **Making an item sticky**
  - Input: ID of an item which needs to be made sticky.
  - Output: None.

**Resources**
This component requires no external resource.

**References**
The details of the implementation of this component can be found in the DDD [3] in the appendix.

**Processing**
There is no complex internal processing other than Database API calls.

**Data**
This component does not track or store any data.

### 5.2 CLUBNET APP FRONT-END

#### 5.2.1 SETTINGS

**Type**
Module.

**Purpose**

**Function**
This component is responsible for the front-end functionality of user settings including changing the system language.

**Subordinates**
This component has the following sub-components:

- Controllers: The logic layer which handles the user interaction and data binding. It contains the Angular controllers and services.

- Views: The data representation layer which contains the HTML and CSS files for displaying the settings view. The Angular model is embedded in the HTML files.

**Dependencies**
This component does not depend on any other component.

**Interfaces**
This component provides the following interfaces:

- Changing language
  - Input: The ISO 639-2 code of a language.
  - Output: None. The system language is changed to the specified one.

**References**
The details of the implementation of this component can be found in the DDD [3] in the appendix.

**Resources**
The following external resource is required:
• Angular-translate package: To support multiple system languages.

Processing
The views sub-component displays the settings view and informs the controller’s sub-component about the user events. The interfaces will be called when it is needed to respond to certain user events. After the interface is called, the angular-translate package takes care of the change of system language.

Data
This component stores which language to display.

5.2.2 CHAT

Type
Module.

Purpose
SR96.

Function
This component is responsible for the front-end functionality of chat.

Subordinates
This component has the following sub-components:

• Controllers: The logic layer which handles the user interaction and data binding. It contains the Angular controllers and services.

• Views: The data representation layer which contains the HTML and CSS files for displaying the chat dialog and contact list. The Angular model is embedded in the HTML files.

Dependencies
This component does not depend on any other component.

Interfaces
This component provides the following interfaces:

• Creating a chat.
  - Input: The ID of the recipient.
  - Output: None. The chat dialog is displayed.

• Sending a message.
  - Input: The message to send.
  - Output: None. The new message is displayed in the chat dialog.
• Getting all chat sessions of a user.
  - Input: None.
  - Output: All chat sessions associated with the user are displayed.

• Showing a chat dialog.
  - Input: A chat object.
  - Output: None. The chat dialog of that chat session is displayed.

References
The details of the implementation of this component can be found in the DDD [3] in the appendix.

Resources
This component requires no external resource.

Processing
The views sub-component displays the chat views and informs the controller’s sub-component about the user events. The interfaces will be called when it is needed to respond to certain user events.
After the interfaces are called, an API call is made to the back-end for sending and retrieving data. Finally the views component is updated to display the corresponding chat views.

Data
This component stores the information of all retrieved chat sessions.

5.2.3 PROFILE

Type
Module.

Purpose
SR9, SR10, SR11, SR12, SR26, SR27.

Function
This module is responsible for the front-end functionality of changing a user profile.

Subordinates
This component has the following sub-components:
  • Controllers: The logic layer which handles the user interaction and data binding. It contains the Angular controllers.
  • Views: The data representation layer which contains the HTML and CSS files for displaying the side menu. The Angular model is embedded in the HTML files.

Dependencies
This component does not depend on any component.
Interfaces
This component provides the following interfaces:

- Updating user profile
  - Input: An array of attributes to update.
  - Output: None.

References
The details of the implementation of this component can be found in the DDD [3] in the appendix.

Resources
This component requires no external resource.

Processing
After the interface is called, the input data is sent as parameters of an API call to back-end. The user profile is then updated in the back-end. Finally the views component is updated to display information.

Data
This component stores the information of the logged in user account.

5.2.4 SIDE MENU

Type
Module.

Purpose
SR5, SR20.

Function
This component is responsible for the functionality of the side menu. The side menu only does the data representation.

Subordinates
This component has the following sub-components:

- Controllers: The logic layer which handles the user interaction and data binding. It contains the Angular controllers.

- Views: The data representation layer which contains the HTML and CSS files for displaying the side menu. The Angular model is embedded in the HTML files.

Dependencies
This component does not depend on any other component.

Interfaces
This component provides no interfaces.
References
The details of the implementation of this component can be found in the DDD [3] in the appendix.

Resources
This component requires no external resource.

Processing
This component only displays the side menu and directs the system to other components when the user clicked the corresponding button.

Data
This component stores the club related information such as logo and front-end color schemes.

5.2.5 ACCESS CONTROL

Type
Module.

Purpose
SR37, SR38, SR44, SR45, SR48, SR49.

Function
This component is responsible for checking whether a user has the permission to perform an operation.

Subordinates
This component has no subordinate.

Dependencies
This component will only be executed after one of the following components made an interface call to it: Side menu, Chats, Feed, and Feed item component.

Interfaces
This component provide the following interface:

- Getting permission
  - Input: The object of the operation, the operation type.
  - Output: True if the operation is allowed. False otherwise.

References
The details of the implementation of this component can be found in the DDD [3] in the appendix.

Resources
This component requires no external resource.

Processing
After the interface is called, the input data is sent as parameters in an API call to the back-end. The output of the API call is then returned.
Data
This component does not store or trace any data.

5.2.6 LOGIN

Type
Module.

Purpose
SR17, SR20.

Function
This component is responsible for the front-end functionality of logging in/out.

Subordinates
This component has the following sub-components:

- Controllers: The logic layer which handles the user interaction and data binding. It contains the Angular controllers and services.

- Views: The data representation layer which contains the HTML and CSS files for displaying the feed view. The Angular model is embedded in the HTML files.

Dependencies
This component does not depend on any component.

Interfaces
This component provides the following interfaces:

- Logging in
  - Input: Email address and password.
  - Output: None. The feed view is displayed if the logging in succeeded.

- Logging out
  - Input: None.
  - Output: None. The logging in view is displayed if the logging out succeeded.

Resources
This component requires no external resource.

References
The details of the implementation of this component can be found in the DDD [3] in the appendix.

Processing
The views sub-component displays the feed view and informs the controller’s sub-component about the user events. The interfaces will be called when it is needed to respond to certain user
events. After the interfaces are called, an API call is made to the back-end for logging in/out. Finally the views component is updated to display information.

Data
This component does not store or trace any data.

5.2.7 ENROLLMENT

Type
Module.

Purpose
SR55.

Function
This component is responsible for the front-end functionality of enrollment.

Subordinates
This component has the following sub-components:

- Controllers: The logic layer which handles the user interaction and data binding. It contains the Angular controllers and services.

- Views: The data representation layer which contains the HTML and CSS files for displaying the feed view. The Angular model is embedded in the HTML files.

Dependencies
This component does not depend on any component.

Interfaces
This component provides the following interfaces:

- Enrolling a new user
  - Input: Meteor enrollment token, password.
  - Output: None. The feed view is displayed if the enrollment succeeded.

Resources
This component requires no external resource.

References
The details of the implementation of this component can be found in the DDD [3] in the appendix.

Processing
The views sub-component displays the feed view and informs the controller’s sub-component about the user events. The interfaces will be called when it is needed to respond to certain user events.
After the interface is called, the Meteor built-in enrollment function takes care of the enrollment.

Data
This component does not store or trace any data.

5.2.8 FEED

Type
Module.

Purpose

Function
The function is to display feed items and filter them by their types.

Subordinates
This component has the following sub-components:

- Controllers: The logic layer which handles the user interaction and data binding. It contains the Angular controllers and services.

- Views: The data representation layer which contains the HTML and CSS files for displaying the feed view. The Angular model is embedded in the HTML files.

Dependencies
This component will only be executed after the Authentication component is executed and user is logged in.

Interfaces
This component provides the following interfaces.

- Loading newest feed items.
  - Input: An array of the feed item types to load.
  - Output: None. Ten feed items of the specified types are loaded and displayed in the feed view.

- Scrolling to view more feed items.
  - Input: None.
  - Output: None. Three more feed items are loaded and displayed at the bottom of the feed view.

- Setting the feed item filter.
  - Input: A list of feed item types to filter on.
Output: None. The filter is updated.

Resources
This component requires no external resource.

References
The details of the implementation of this component can be found in the DDD [3] in the appendix.

Processing
The views sub-component displays the feed view and informs the controller’s sub-component about the user events. The interfaces will be called when it is needed to respond to certain user events.

For the interfaces of loading the newest feed item and scrolling to view more feed items, it first processes the input data and then makes API calls to the back-end to send data retrieval requests. Finally the views sub-component is updated to display new data.

For the interface of setting the feed item filter no API call to the back-end will be made. It requires the Access control component to load the feed item types that can be viewed by the user. Only the filter in the controllers sub-component is updated.

Data
The following data is stored in this component:

- The array of feed item types to view is stored in the controllers sub-component.
- The list of retrieved feed items is stored in the controllers sub-component.

5.2.9 FEED ITEM

Type
Module.

Purpose

Function
The function is to provide the functionality that is general to all feed items, including editing, deleting, sticking a feed item, registering a response and adding a note.

Subordinates
This component has the following sub-components:

- Controllers: The logic layer which handles the user interaction and data binding. It contains the Angular controllers and services.
- Views: The data representation layer which contains the HTML and CSS files for displaying the feed view. The Angular model is embedded in the HTML files.
Dependencies
This component will only be executed after the feed component is executed and the feed view is ready to display.

Interfaces

- Opening the item operations (edit, delete, make a note, sticky) popover.
  - Input: None.
  - Output: None. The item operations popover is displayed.

- Showing full/shrunk details.
  - Input: The feed item object.
  - Output: None. Show full details if the feed item was shrunk. Otherwise show shrunk details.

- Editing the feed item.
  - Input: The values to be updated.
  - Output: None.

- Deleting the feed item.
  - Input: The feed item object.
  - Output: None. The feed item is deleted from the feed item list.

- Adding a note
  - Input: The feed item object and the note string.
  - Output: None.

- Registering a response
  - Input: The value of the response.
  - Output: None.

Resources
This component requires no external resource.

References
The details of the implementation of this component can be found in the DDD [3] in the appendix.

Processing
The views sub-component displays the feed view and informs the controller’s sub-component about the user events. The interfaces will be called when it is needed to respond to certain user
events.
For the interfaces of editing or deleting the feed item, registering a response and adding a note, it first process the input data and then makes API calls to the back-end to send data modification requests. Finally the views sub-component is updated to display new data.

For the interface of opening the item operations popover and showing full/shrunk details, no API call to the back-end is made. Only the views component is updated to display information.

Data
This component does not trace or store any data.

5.3 WEB INTERFACE FRONT-END

The Betting pool, Feed, and Feed item components will not be implemented. Thus they will not be included in this section.

The Profile, Login and Side menu components in the web interface front-end have almost the same specifications as those in the ClubNet app front-end. The only difference is the implementation details of the user interface. Thus only the Registration, Accounts, and Club setting component are described below.

5.3.1 REGISTRATION

Type
Module.

Purpose
SR55, SR56, SR57, SR58.

Function
This component is responsible for the front-end functionality of registering a new user in a club.

Subordinates
This component has the following sub-components:

- Controllers: The logic layer which handles the user interaction and data binding. It contains the Angular controllers and services.
- Views: The data representation layer which contains the HTML and CSS files for displaying the registration view. The Angular model is embedded in the HTML files.

Dependencies
This component does not depend on any component.

Interfaces
This component provides the following interfaces:

- Registering a new user
- Input: The first name, last name, email address, and team (if applicable) of the new user.
- Output: None.

Resources
This component requires no external resource.

References
The details of the implementation of this component can be found in the DDD [3] in the appendix.

Processing
The views sub-component displays the registration view and informs the controller’s sub-component about the user events. The interfaces will be called when it is needed to respond to certain user events. After the interface is called, the input data is sent as parameters in an API call to the back-end.

Data
This component does not store or trace any data.

5.3.2 ACCOUNTS

Type
Module.

Purpose
SR11, SR26, SR55, SR57.

Function
This component is responsible for the front-end functionality of accounts management in a club.

Subordinates
This component has the following sub-components:

- Controllers: The logic layer which handles the user interaction and data binding. It contains the Angular controllers and services.
- Views: The data representation layer which contains the HTML and CSS files for displaying the accounts management view. The Angular model is embedded in the HTML files.

Dependencies
This component does not depend on any component.

Interfaces
This component provides the following interfaces:

- Editing the information of a new user
- Input: The new first name, last name, email address, and team (if applicable) of the new user.
- Output: None.

- Deleting a user account.
  - Input: The selected user object.
  - Output: None.

Resources
This component requires no external resource.

References
The details of the implementation of this component can be found in the DDD [3] in the appendix.

Processing
The views sub-component displays the accounts management view and informs the controller’s sub-component about the user events. The interfaces will be called when it is needed to respond to certain user events.
After the interface is called, the input data is sent as parameters in an API call to the back-end.

Data
This component stores an array of the accounts in the club.

5.3.3 CLUB SETTING

Type
Module.

Purpose
SR68, SR69, SR70, SR71, SR72, SR73, SR74, SR75.

Function
This component is responsible for the front-end functionality of setting the club name, color scheme, and club logo.

Subordinates
This component has the following sub-components:

- Controllers: The logic layer which handles the user interaction and data binding. It contains the Angular controllers and services.

- Views: The data representation layer which contains the HTML and CSS files for displaying the club setting view. The Angular model is embedded in the HTML files.

Dependencies
This component does not depend on any component.
Interfaces
This component provides the following interfaces:

- Setting the club name
  - Input: The club name.
  - Output: None.

- Setting the color scheme
  - Input: An array of three colors in the hexadecimal formats.
  - Output: None.

- Setting the club logo
  - Input: The image of the club log.
  - Output: None.

Resources
This component requires no external resource.

References
The details of the implementation of this component can be found in the DDD [3] in the appendix.

Processing
The view sub-component displays the club setting view and informs the controller's sub-component about the user events. The interfaces will be called when it is needed to respond to certain user events.

After the interface is called, the input data is sent as parameters in an API call to the back-end.

Data
This component stores the values of settings of the club.
6 FEASIBILITY AND RESOURCE ESTIMATES

This section gives an estimation of the computer resources which are needed to develop and use ClubNet. The usage of ClubNet is divided into a server side and a client side.

6.0.1 RESOURCE REQUIREMENTS

The requirements for the development of ClubNet are:

- **CPU**: \( \geq 1.0 \text{ GHz x86 or equivalent} \)
- **Memory**: \( \geq 2 \text{ GB RAM} \)
- **Hard disk**: \( \geq 1 \text{ GB free on disk} \)
- **Operating system**: Windows 7 and above
- **Software**: Any text editor able to handle JavaScript code (WebStorm recommended), Ionic 1.3.1, Node.js 4.4.3, Meteor 1.3, Cordova 6.0.0, AngularJS 1.3.11, Mocha 2.4.5_2

The requirements for operating ClubNet are:

- **ClubNet app**:
  - **Memory**: \( \geq 256 \text{ MB available on the mobile phone} \)
  - **Storage**: \( \geq 25 \text{ MB free} \)
  - **Platform**: Android 4.1 and above, iOS 7.0 and above

- **Web interface**:
  - **CPU**: \( \geq 1.0 \text{ GHz x86 or equivalent} \)
  - **Memory**: \( \geq 2 \text{ GB RAM} \)
  - **Hard disk**: \( \geq 1 \text{ GB free on disk} \)
  - **Software**: Chrome 48 and above or, Firefox 44 and above or, Internet Explorer 11 and above or, Safari 8 and above

- **Back-end**:
  - **CPU**: \( \geq 1.0 \text{ GHz x86 or equivalent} \)
  - **Memory**: \( \geq 2 \text{ GB RAM} \)
  - **Hard disk**: \( \geq 1 \text{ GB free on disk} \)
  - **Operating system**: Windows 7 and above
  - **Software**: Meteor 1.3, Node.js 4.4.3

6.0.2 PERFORMANCE

With the requirements specified in section Resource requirements being met, the following performance estimation should be met:
<table>
<thead>
<tr>
<th>Component</th>
<th>Response Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Development</td>
<td>≤ 800 ms response time per test case, ≤ 50000 ms for building the app</td>
</tr>
<tr>
<td>ClubNet app</td>
<td>≤ 200 ms response time per user action</td>
</tr>
<tr>
<td>Web interface</td>
<td>≤ 200 ms response time per user action</td>
</tr>
<tr>
<td>Back-end</td>
<td>≤ 200 ms response time per request</td>
</tr>
</tbody>
</table>
7 REQUIREMENTS TRACEABILITY MATRIX

This section describes how the software requirements (SR) in the Software Requirements Document[2] are related to the software components described in section Component Description. A mapping from the software requirements to the corresponding software components and one from other way are presented below.

7.1 SR TO COMPONENTS

<table>
<thead>
<tr>
<th>SR</th>
<th>Back-End Components</th>
<th>Front-End Components</th>
</tr>
</thead>
<tbody>
<tr>
<td>SR1</td>
<td>Accounts API</td>
<td></td>
</tr>
<tr>
<td>SR2</td>
<td>Accounts API, Authentication API</td>
<td></td>
</tr>
<tr>
<td>SR3</td>
<td>Accounts API, Authentication API</td>
<td></td>
</tr>
<tr>
<td>SR4</td>
<td>Accounts API</td>
<td></td>
</tr>
<tr>
<td>SR5</td>
<td>Accounts API</td>
<td></td>
</tr>
<tr>
<td>SR6</td>
<td>Club betting</td>
<td></td>
</tr>
<tr>
<td>SR7</td>
<td></td>
<td>Settings</td>
</tr>
<tr>
<td>SR8</td>
<td></td>
<td>Settings</td>
</tr>
<tr>
<td>SR9</td>
<td>Accounts API</td>
<td>Profile</td>
</tr>
<tr>
<td>SR10</td>
<td>Accounts API, Authentication API</td>
<td>Profile</td>
</tr>
<tr>
<td>SR11</td>
<td>Accounts API</td>
<td>Profile</td>
</tr>
<tr>
<td>SR12</td>
<td>Accounts API</td>
<td>Profile</td>
</tr>
<tr>
<td>SR13</td>
<td>Database API</td>
<td>Feed item</td>
</tr>
<tr>
<td>SR14</td>
<td></td>
<td>Feed item</td>
</tr>
<tr>
<td>SR15</td>
<td></td>
<td>Feed item</td>
</tr>
<tr>
<td>SR16</td>
<td>Database</td>
<td>Feed item</td>
</tr>
<tr>
<td>SR17</td>
<td>Authentication API</td>
<td>Login</td>
</tr>
<tr>
<td>SR18</td>
<td>Authentication API</td>
<td>Login</td>
</tr>
<tr>
<td>SR19</td>
<td>Authentication API</td>
<td>Login</td>
</tr>
<tr>
<td>SR20</td>
<td>Authentication API</td>
<td>Login</td>
</tr>
<tr>
<td>SR21</td>
<td>Exercise voting, Practicality</td>
<td></td>
</tr>
<tr>
<td>SR22</td>
<td>Exercise voting, Practicality</td>
<td></td>
</tr>
<tr>
<td>SR23</td>
<td>Accounts API</td>
<td>Settings</td>
</tr>
<tr>
<td>SR24</td>
<td>Accounts API</td>
<td>Settings</td>
</tr>
<tr>
<td>SR25</td>
<td>Accounts API</td>
<td>Settings</td>
</tr>
<tr>
<td>SR26</td>
<td>Accounts API</td>
<td>Settings</td>
</tr>
<tr>
<td>SR27</td>
<td>Accounts API</td>
<td>Settings</td>
</tr>
<tr>
<td>SR28</td>
<td>Database API</td>
<td>Feed item</td>
</tr>
<tr>
<td>SR29</td>
<td></td>
<td>Feed item</td>
</tr>
<tr>
<td>SR30</td>
<td>Practicality</td>
<td>Feed item</td>
</tr>
<tr>
<td>------</td>
<td>--------------</td>
<td>-----------</td>
</tr>
<tr>
<td>SR31</td>
<td>Accounts API</td>
<td>Feed</td>
</tr>
<tr>
<td>SR32</td>
<td>Feed</td>
<td>Feed</td>
</tr>
<tr>
<td>SR34</td>
<td>Feed</td>
<td>Feed</td>
</tr>
<tr>
<td>SR35</td>
<td>Database API</td>
<td>Feed</td>
</tr>
<tr>
<td>SR36</td>
<td>Feed</td>
<td>Feed item</td>
</tr>
<tr>
<td>SR37</td>
<td>Access control</td>
<td>Feed</td>
</tr>
<tr>
<td>SR37a</td>
<td>Heroes</td>
<td>Feed</td>
</tr>
<tr>
<td>SR38</td>
<td>Access control</td>
<td>Feed</td>
</tr>
<tr>
<td>SR39</td>
<td>Feed</td>
<td>Feed</td>
</tr>
<tr>
<td>SR40</td>
<td>Feed</td>
<td>Feed</td>
</tr>
<tr>
<td>SR41</td>
<td>Feed</td>
<td>Feed</td>
</tr>
<tr>
<td>SR42</td>
<td>Database API</td>
<td>Feed item</td>
</tr>
<tr>
<td>SR43</td>
<td>Feed</td>
<td>Feed item</td>
</tr>
<tr>
<td>SR44</td>
<td>Access control</td>
<td>Feed</td>
</tr>
<tr>
<td>SR45</td>
<td>Access control</td>
<td>Feed</td>
</tr>
<tr>
<td>SR46</td>
<td>Feed</td>
<td>Feed</td>
</tr>
<tr>
<td>SR47</td>
<td>Feed</td>
<td>Feed</td>
</tr>
<tr>
<td>SR48</td>
<td>Access control</td>
<td>Feed</td>
</tr>
<tr>
<td>SR49</td>
<td>Access control</td>
<td>Feed</td>
</tr>
<tr>
<td>SR50</td>
<td>Accounts API</td>
<td>Accounts</td>
</tr>
<tr>
<td>SR51</td>
<td>Sponsoring</td>
<td>Feed item</td>
</tr>
<tr>
<td>SR52</td>
<td>Sponsoring</td>
<td>Feed item</td>
</tr>
<tr>
<td>SR53</td>
<td>Sponsoring</td>
<td>Feed item</td>
</tr>
<tr>
<td>SR54</td>
<td>Sponsoring</td>
<td>Feed item</td>
</tr>
<tr>
<td>SR55</td>
<td>Accounts API</td>
<td>Accounts</td>
</tr>
<tr>
<td>SR56</td>
<td>Accounts</td>
<td>Accounts</td>
</tr>
<tr>
<td>SR57</td>
<td>Accounts API</td>
<td>Accounts</td>
</tr>
<tr>
<td>SR58</td>
<td>Accounts API</td>
<td>Accounts</td>
</tr>
<tr>
<td>SR59</td>
<td>Heroes</td>
<td>Feed item</td>
</tr>
<tr>
<td>SR60</td>
<td>Feed</td>
<td>Feed item</td>
</tr>
<tr>
<td>SR61</td>
<td>Database API</td>
<td>Feed item</td>
</tr>
<tr>
<td>SR62</td>
<td>Feed</td>
<td>Feed item</td>
</tr>
<tr>
<td>SR63</td>
<td>Club betting</td>
<td>Club betting</td>
</tr>
<tr>
<td>SR64</td>
<td>Club betting</td>
<td>Club betting</td>
</tr>
<tr>
<td>SR65</td>
<td>Club betting</td>
<td>Club betting</td>
</tr>
<tr>
<td>SR66</td>
<td>Club betting</td>
<td>Club betting</td>
</tr>
<tr>
<td>SR67</td>
<td>Club betting</td>
<td>Club setting</td>
</tr>
<tr>
<td>SR68</td>
<td>Club setting</td>
<td>Club setting</td>
</tr>
<tr>
<td>SR69</td>
<td>Club setting</td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>--------------</td>
<td></td>
</tr>
<tr>
<td>SR70</td>
<td>Club setting</td>
<td></td>
</tr>
<tr>
<td>SR71</td>
<td>Club setting</td>
<td></td>
</tr>
<tr>
<td>SR72</td>
<td>Club setting</td>
<td></td>
</tr>
<tr>
<td>SR73</td>
<td>Club setting</td>
<td></td>
</tr>
<tr>
<td>SR74</td>
<td>Club setting</td>
<td></td>
</tr>
<tr>
<td>SR75</td>
<td>Club setting</td>
<td></td>
</tr>
<tr>
<td>SR76</td>
<td>Sponsoring</td>
<td></td>
</tr>
<tr>
<td>SR77</td>
<td>Sponsoring</td>
<td></td>
</tr>
<tr>
<td>SR78</td>
<td>Feed</td>
<td></td>
</tr>
<tr>
<td>SR79</td>
<td>Feed</td>
<td></td>
</tr>
<tr>
<td>SR80</td>
<td>Database</td>
<td></td>
</tr>
<tr>
<td>SR81</td>
<td>Database</td>
<td></td>
</tr>
<tr>
<td>SR82</td>
<td>Database</td>
<td></td>
</tr>
<tr>
<td>SR83</td>
<td>Database</td>
<td></td>
</tr>
<tr>
<td>SR84</td>
<td>Database</td>
<td></td>
</tr>
<tr>
<td>SR85</td>
<td>Database</td>
<td></td>
</tr>
<tr>
<td>SR86</td>
<td>Database</td>
<td></td>
</tr>
<tr>
<td>SR87</td>
<td>Database</td>
<td></td>
</tr>
<tr>
<td>SR88</td>
<td>Feed item</td>
<td></td>
</tr>
<tr>
<td>SR89</td>
<td>Access control</td>
<td></td>
</tr>
<tr>
<td>SR90</td>
<td>Database</td>
<td></td>
</tr>
<tr>
<td>SR91</td>
<td>Database</td>
<td></td>
</tr>
<tr>
<td>SR92</td>
<td>Database</td>
<td></td>
</tr>
<tr>
<td>SR93</td>
<td>Provided by IntuitiveTechnologies B.V.</td>
<td></td>
</tr>
<tr>
<td>SR94</td>
<td>Exercise voting</td>
<td></td>
</tr>
<tr>
<td>SR95</td>
<td>Provided by IntuitiveTechnologies B.V.</td>
<td></td>
</tr>
<tr>
<td>SR96</td>
<td>Chat</td>
<td></td>
</tr>
<tr>
<td>SR97</td>
<td>Chat</td>
<td></td>
</tr>
<tr>
<td>SR98</td>
<td>Access control</td>
<td></td>
</tr>
<tr>
<td>SR99</td>
<td>Access control</td>
<td></td>
</tr>
<tr>
<td>SR100</td>
<td>Database</td>
<td></td>
</tr>
<tr>
<td>SR101</td>
<td>Database</td>
<td></td>
</tr>
<tr>
<td>SR101a</td>
<td>Suggestion</td>
<td></td>
</tr>
<tr>
<td>SR102</td>
<td>Database</td>
<td></td>
</tr>
<tr>
<td>SR103</td>
<td>Suggestion</td>
<td></td>
</tr>
<tr>
<td>SR104</td>
<td>Database</td>
<td></td>
</tr>
<tr>
<td>SR105</td>
<td>Feed item</td>
<td></td>
</tr>
<tr>
<td>SR106</td>
<td>Database</td>
<td></td>
</tr>
<tr>
<td>SR107</td>
<td>Feed item</td>
<td></td>
</tr>
<tr>
<td>SR108</td>
<td>Database</td>
<td></td>
</tr>
<tr>
<td>SR109</td>
<td>Feed item</td>
<td></td>
</tr>
<tr>
<td>SR110</td>
<td>Database</td>
<td></td>
</tr>
<tr>
<td>SR111</td>
<td>Feed item</td>
<td></td>
</tr>
<tr>
<td>SR112</td>
<td>Database</td>
<td></td>
</tr>
<tr>
<td>SR113</td>
<td>Feed item</td>
<td></td>
</tr>
<tr>
<td>SR114</td>
<td>Database</td>
<td></td>
</tr>
<tr>
<td>SR115</td>
<td>Sponsoring</td>
<td></td>
</tr>
<tr>
<td>SR116</td>
<td>Accounts API</td>
<td></td>
</tr>
<tr>
<td>SR117</td>
<td>Feed item</td>
<td></td>
</tr>
<tr>
<td>SR118</td>
<td>Sponsoring</td>
<td></td>
</tr>
<tr>
<td>SR119</td>
<td>Database</td>
<td></td>
</tr>
<tr>
<td>SR120</td>
<td>Feed item</td>
<td></td>
</tr>
<tr>
<td>SR121</td>
<td>Database</td>
<td></td>
</tr>
<tr>
<td>SR122</td>
<td>Database</td>
<td></td>
</tr>
<tr>
<td>SR123</td>
<td>Database</td>
<td></td>
</tr>
<tr>
<td>SR124</td>
<td>Feed item</td>
<td></td>
</tr>
<tr>
<td>SR125</td>
<td>Database</td>
<td></td>
</tr>
<tr>
<td>SR126</td>
<td>Practicality</td>
<td></td>
</tr>
<tr>
<td>SR127</td>
<td>Practicality</td>
<td></td>
</tr>
<tr>
<td>SR128</td>
<td>Accounts API</td>
<td></td>
</tr>
<tr>
<td>SR129</td>
<td>Feed item</td>
<td></td>
</tr>
<tr>
<td>SR130</td>
<td>Practicality</td>
<td></td>
</tr>
<tr>
<td>SR131</td>
<td>Accounts API</td>
<td></td>
</tr>
<tr>
<td>SR132</td>
<td>Database</td>
<td></td>
</tr>
<tr>
<td>SR133</td>
<td>Database</td>
<td></td>
</tr>
<tr>
<td>SR134</td>
<td>Database</td>
<td></td>
</tr>
<tr>
<td>SR135</td>
<td>Database</td>
<td></td>
</tr>
<tr>
<td>SR136</td>
<td>Database</td>
<td></td>
</tr>
<tr>
<td>SR137</td>
<td>Database</td>
<td></td>
</tr>
<tr>
<td>SR138</td>
<td>Exercise voting</td>
<td></td>
</tr>
<tr>
<td>SR139</td>
<td>Exercise voting</td>
<td></td>
</tr>
<tr>
<td>SR140</td>
<td>Exercise voting</td>
<td></td>
</tr>
<tr>
<td>SR141</td>
<td>Exercise voting</td>
<td></td>
</tr>
<tr>
<td>SR142</td>
<td>Exercise voting</td>
<td></td>
</tr>
<tr>
<td>Reference</td>
<td>Feature</td>
<td>Provider</td>
</tr>
<tr>
<td>-----------</td>
<td>---------</td>
<td>----------</td>
</tr>
<tr>
<td>SR143</td>
<td>Feed item</td>
<td></td>
</tr>
<tr>
<td>SR144</td>
<td>Feed item</td>
<td></td>
</tr>
<tr>
<td>SR145</td>
<td>Database</td>
<td></td>
</tr>
<tr>
<td>SR146</td>
<td>Club betting</td>
<td></td>
</tr>
<tr>
<td>SR147</td>
<td>Club betting</td>
<td>Provided by angular-translate package</td>
</tr>
<tr>
<td>SR148</td>
<td></td>
<td>Provided by angular-translate package</td>
</tr>
<tr>
<td>SR149</td>
<td></td>
<td>Provided by angular-translate package</td>
</tr>
<tr>
<td>SR150</td>
<td></td>
<td>Provided by Ionic</td>
</tr>
<tr>
<td>SR151</td>
<td></td>
<td>Provided by Ionic</td>
</tr>
<tr>
<td>SR152</td>
<td></td>
<td>Provided by Ionic</td>
</tr>
<tr>
<td>SR153</td>
<td></td>
<td>Provided by Ionic</td>
</tr>
<tr>
<td>SR154</td>
<td></td>
<td>Provided by Ionic</td>
</tr>
<tr>
<td>SR155</td>
<td></td>
<td>Provided by Ionic</td>
</tr>
<tr>
<td>SR156</td>
<td></td>
<td>Provided by Ionic</td>
</tr>
<tr>
<td>SR157</td>
<td></td>
<td>Provided by Ionic</td>
</tr>
<tr>
<td>SR158</td>
<td></td>
<td>Provided by Ionic</td>
</tr>
<tr>
<td>SR159</td>
<td></td>
<td>Provided by Ionic</td>
</tr>
<tr>
<td>SR160</td>
<td></td>
<td>Provided by Meteor</td>
</tr>
<tr>
<td>SR161</td>
<td>Database</td>
<td></td>
</tr>
<tr>
<td>SR162</td>
<td></td>
<td>All</td>
</tr>
<tr>
<td>SR163</td>
<td></td>
<td>All</td>
</tr>
<tr>
<td>SR164</td>
<td></td>
<td>All</td>
</tr>
<tr>
<td>SR165</td>
<td></td>
<td>All</td>
</tr>
<tr>
<td>SR166</td>
<td></td>
<td>All</td>
</tr>
<tr>
<td>SR167</td>
<td>Accounts API, Exercise voting</td>
<td></td>
</tr>
<tr>
<td>SR168</td>
<td>Accounts API, Exercise voting</td>
<td>Provided by Meteor</td>
</tr>
<tr>
<td>SR169</td>
<td></td>
<td>Database API</td>
</tr>
<tr>
<td>SR170</td>
<td></td>
<td>Database, Database API</td>
</tr>
<tr>
<td>SR171</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SR172</td>
<td></td>
<td>Feed</td>
</tr>
<tr>
<td>SR173</td>
<td></td>
<td>Feed item</td>
</tr>
<tr>
<td>SR174</td>
<td></td>
<td>Feed</td>
</tr>
<tr>
<td>SR175</td>
<td></td>
<td>Side menu</td>
</tr>
<tr>
<td>SR176</td>
<td></td>
<td>Side menu</td>
</tr>
<tr>
<td>SR177</td>
<td></td>
<td>Feed</td>
</tr>
<tr>
<td>SR178</td>
<td></td>
<td>Feed item</td>
</tr>
<tr>
<td>Component</td>
<td>SR</td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>----</td>
<td></td>
</tr>
</tbody>
</table>

### 7.2 COMPONENTS TO SR

#### 7.2.1 BACK-END
### Authentication API
- SR17, SR18, SR19, SR20

### Accounts API
- SR9, SR10, SR11, SR12, SR23, SR26, SR27, SR55, SR57

### Access control

### Chat
- SR96, SR97, SR98, SR99

### Sponsoring
- SR35, SR37, SR38, SR51, SR76, SR115, SR118

### Club betting
- SR35, SR63, SR67, SR109, SR145

### Heroes
- SR35, SR59, SR37a

### Suggestion
- SR35, SR103

### Practicality
- SR28, SR30, SR35, SR37, SR125, SR126, SR127, SR130

### Exercise voting
- SR35, SR37, SR138, SR139, SR140, SR141, SR142

### Feed
- SR46, SR32, SR39

### 7.2.2 FRONT-END

<table>
<thead>
<tr>
<th>Component</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Login</td>
<td>SR17, SR18, SR19, SR20</td>
</tr>
<tr>
<td>Side menu</td>
<td>SR5, SR20</td>
</tr>
<tr>
<td>Profile</td>
<td>SR9, SR10, SR11, SR12, SR26, SR27</td>
</tr>
<tr>
<td>Settings</td>
<td>SR24, SR25, SR7, SR8</td>
</tr>
<tr>
<td>Chat</td>
<td>SR96, SR97, SR98, SR99</td>
</tr>
<tr>
<td>Club setting</td>
<td>SR68, SR69, SR70, SR71, SR72, SR73, SR74, SR75</td>
</tr>
<tr>
<td>Registration</td>
<td>SR55, SR56, SR57, SR58</td>
</tr>
<tr>
<td>Betting pool</td>
<td>SR63, SR64, SR65, SR66, SR145, SR146</td>
</tr>
</tbody>
</table>