Handover Document

Connected.Football

Lucas Gehlen, Marco Kull, Patrick Richter, Sebastian Wilczek

Fontys University of Applied Sciences

Venlo, January 14, 2019

Contents

List of Figures

List of Listings

List of Abbreviations

ADB Android Debugging Bridge

AVD Android Virtual Device

NPM Node.js Package Manager

SVN Subversion

USB Universal Serial Bus

1 Introduction

This document is the handover report of the Software Factory project *Connected.Football*, which was worked on by students of the seventh semester of the Software Engineering / Business Informatics course of Fontys University of Applied Sciences. It is intended to be read as a guide on what the project was about, how to set up the software artefacts developed during the project and how the development could be continued. The intended audience are students from the same semester as the authors or Software Engineering / Business Informatics students with a similar level of knowledge.
The handover document first of all describes the developed product itself, which is an implemented functionality of the mobile application *Connected.Football*. To do so, it is described what frameworks both the application itself and the backend communicating with it are based on, as well as how they communicate. It is furthermore detailed what functionality the project group worked on, with focus on why such a functionality was necessary to be implemented.
This is followed by a guide on how to setup the developed software artefacts. This includes directions as to where to retrieve said artefacts. Following that, it is detailed how the mobile application can be built from the retrieved source code and how the application can be tested on both an emulated *Android* phone and an actual *Android* phone, as it was done during development.
The next chapter details the work of the project group on a high level view. It is explained where most of the relevant source code can be found and how the developed artefacts interact with each other. This chapter is structured around one particular use case which is used as an example to detail the structure of the source code.
To finish off, the handover document closes with suggestions as to how to continue the development. In particular, it is explained what functionality is still missing from the implemented feature. Furthermore, depending on the type of missing functionality, it is suggested where to start or what technology or frameworks to make use of.

2 Mobile Application *Connected.Football*

This chapter explains the *Connected.Football* application in detail. The *Connected.Football* application is used by football coaches for planning and communication with the team.

2.1 Architecture

The *Connected.Football* application was created with *React Native*. *React Native* is based on the *JavaScript* library *React* which is maintained by Facebook. *React* can be used as a base for single-page websites and for mobile applications.

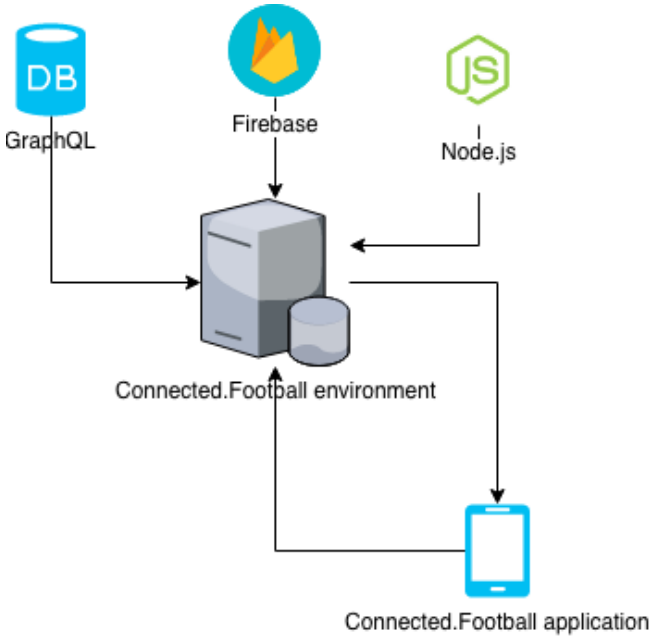

Figure 1: *Connected.Football* Architecture

Connected.Football also uses *MongoDB* as a database for storing the members, the training programs and other data. *GraphQL* is used to communicate between the *Connected.Football* application and the database.

2.1.1 *React Native*

React Native was announced by Facebook in 2015 and is using the *React* architecture to crate native *Android* and *iOS* applications.
The working principles of *React Native* are basically the same as *React* except that *React Native* uses native views. It runs in a background process (which interprets the *JavaScript* written by the developers) directly on the end-device.

2.1.2 *GraphQL*

GraphQL is a query language for APIs and a run time for fulfilling those queries with existing data. *GraphQL* provides a complete and understandable description of the data in your API,

gives clients the power to ask for exactly what they need and nothing more, makes it easier to evolve APIs over time, and enables powerful developer tools.

3 Setup Guide

This chapter deals with the technical details of how to setup the *Connected.Football* application for future development. To do so, it is explained how the source code artefacts can be retrieved, how the necessary modules can be installed with *Node.js*, how an Android device can be set up to be used during development, and eventually how to deploy and run the application on the device, as well as how to use it once it was deployed.

3.1 Retrieval of Sources

The source code of the application can be found on *GitHub*. For reasons of confidentiality, the link to the actual repository can not be disclosed in this document. Access to the repository can be requested from the owner of the *Connected.Football* application. Once access has been granted, the repository has to be checked out to a local machine using *Git*.
Further development artefacts, such as design mockups and state diagrams, can be found in an *SVN* repository of Fontys University of Applied Sciences. The repository can be found at `https://www.fontysvenlo.org/svn/2018/sofa2018/g2`. Access to the repository can be requested from the lecturers of SEBI Venlo. For the deployment of the application itself access to this repository is not a necessity.
Since it is unlikely that a future development team that is not enrolled at Fontys University of Applied Sciences would get access to the aforementioned repositories, the mentioned diagrams can be found in the appendix of this document, more specifically *Appendix A: Activity Diagrams*, *Appendix B: State Machine Diagrams* and *Appendix C: Mockups*.

3.2 Installing modules

Since the application is based on the *React Native* framework, it is necessary to install further modules using *Node.js*. It is therefore required to install *Node.js*, which comes bundled with *NPM*, the package manager of *Node.js*. All can be downloaded at `https://nodejs.org/`.
Once the repository has been downloaded to a local machine which has *Node.js* installed, it is necessary to install the modules as they are denoted by the file *package.json*. To do so, the following command needs to be executed in the same directory as the aforementioned file:

```
1 npm install
```

Listing 1: *Node.js* Installation

If a new *Node.js* module is to be installed during future development, it can be installed with the command below. In this case the file *package.json* needs to be committed to the repository, so all other team members can adapt to the change by running the command mentioned in Listing 1 again.

```
1 npm install --save [NAME OF THE TO-BE-INSTALLED MODULE]
```

Listing 2: *Node.js* Installation of a specific module

3.3 Mobile Platform Setup

Since *React Native* applications are created with a mobile platform as a target, it is necessary to have a mobile device ready for development and testing. During the project, *Android*

devices were made use of. To deploy *React Native* applications to any kind of *Android* device, it is required to install the *ADB* drivers on the development machine. A guide to installing and using *ADB* can be found in the *Android* developer documentation, specifically at `https://developer.android.com/studio/command-line/adb`.
During the course of the project, it was tried to run the application on an *iOS* device, both physical and emulated. Since there have been a great amount of problems trying to get the application to run, the deployment to the *iOS* platform has not played a part in this project and was not followed upon on further. Therefore, there is also no mention of *iOS* in the further setup.
React Native applications can be deployed to both physical devices as well as emulated ones. To deploy a development build of a *React Native* application to a physical device, the device needs to be enabled to handle USB debugging. Instructions to enable USB debugging can be found at `https://facebook.github.io/react-native/docs/running-on-device.html`.
If no physical *Android* device is available for development purposes, an *Android* device can be emulated on the development machine. Known as *AVDs*, such emulated devices can be created and started using the *Android* Development Studio software. Further details can again be found at the *Android* developer documentation, at `https://developer.android.com/studio/run/managing-avds`. The *AVD* has enabled USB debugging by default.
To test if a physical or emulated device is ready to be used for *React Native* development, the following command can be ran:

```
1 adb devices
```

Listing 3: ADB Connection Test

If the device is connected properly, the command will return its name and ID. If this is the case, the device is ready to be used for development.

3.4 Deploying and running

Once a device is ready to be used, the application can be built. To do so, the following commands need to be run:

```
1 gradlew clean            # in the /android subfolder
2 react-native run-android # in the main folder
```

Listing 4: Building and Deploying

This will clean and retrieve any dependencies needed and then install a development build of the application on the connected device. The development build can stay open on the device as long as it stays connected. If there is a change in the source code, the change can be reflected in the build without recompiling, since the development build connects to the development machine as a server. On a physical device, the device itself can be shaken to bring up the developer menu, which includes commands to reload once, reload every time there is a change and even to reload over a wireless connection. On emulated device, the letter *R* reloads the application. If the connection between server and device is separated, the application needs to be built and deployed again.
After following this guide, the application should be accessible on an *Android* device. Please note that an account is required to access all functionality of the application, including the functionality developed during the project in question. Such an account can be requested by the owner of the *Connected.Football* application.

4 Functionality *Vote4Fun*

This chapter describes functionality of the Vote4Fun extension. The basic idea of this extension is that a coach is able to create a poll for players that will be able to vote for exercises. To do so, a setup wizard was created that takes the user through the necessary steps to create such a poll. The wizard can be found during the creation of a new program, as it can be seen in *Figure 2*.

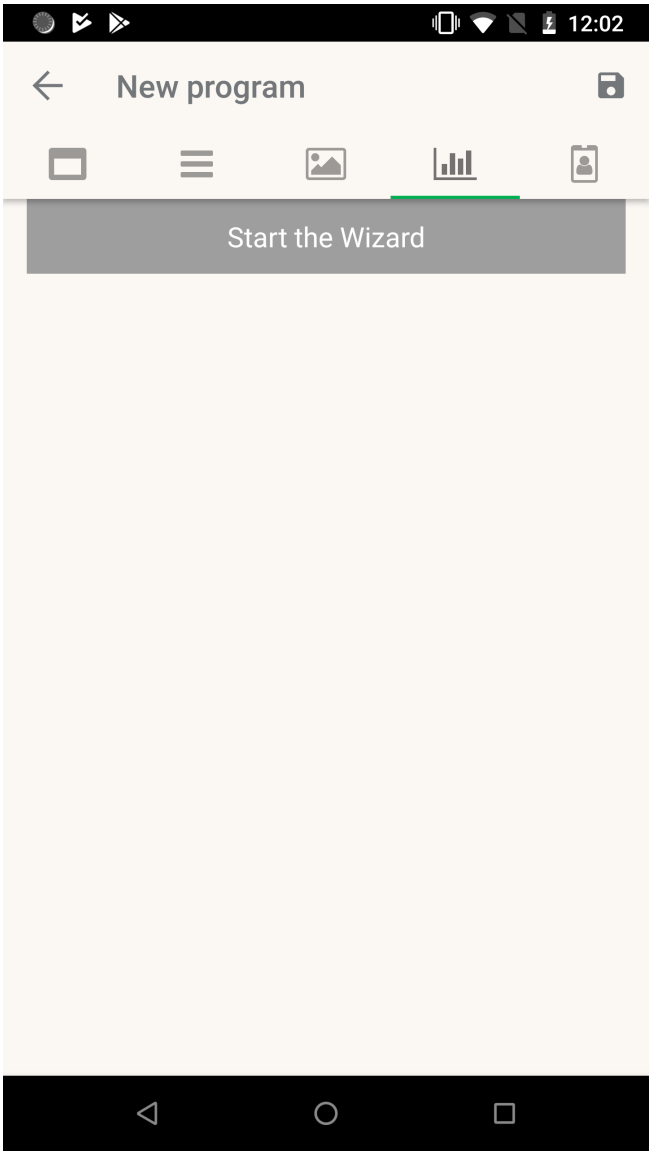

Figure 2: Starting the Wizard in *New program*

The wizard has four stages, first the coach will be able to give the poll a name, then he will be able to select a team that is allowed to vote for this poll. After this step, the coach is allowed to add guests to the poll, guests are users that are not in the previously selected team. This step is not mandatory and can be skipped. The coach can than set the training programs that should be voted upon as well as a deadline for the poll.
Now a confirmation page is prompted where all entered information are displayed and the coach is able to confirm the entered details or correct information, if needed. If everything is correct, the application sends a *GraphQL* mutation to a server, where a new poll object is

created using the information entered by the user. A schema of such an object can be found in *Listing 6*.
Almost all components written as a part of the project are functional components. They make use of a *Node.js* module called *recompose*. This means that the functional part of the component is strictly used for rendering purposes. No logic is implemented this way. Making use of *recompose*'s features such as *withHandlers*, *withState*, *onlyUpdateForKeys* and *mapToProps*, logic can be implemented and supplied to the functional component. This way, business logic can be reused rather easily. State can also be applied and managed as well.
The entire wizard is being used using a navigation push. This means the wizard is encapsulated, with only properties and callbacks being handed down through the navigation component inside the pushing operation.
The following snippet of code (see *Listing 5*) shows a simplified version of the Wizard setup as it is written in *React Native*. This snippet shows the applicable usage of *recompose*. The source code first defines as functional component called *WizardProcess*, which takes a variable and two functions as its parameters. It returns a rendering function that makes use of the supplied parameters to show certain stages of the setup depending on one variable. Depending on the variable *stage*, different components are shown, including the buttons that increment and decrement the stage. However, the buttons do not handle or implement this incrementation logic themselves. They too reference only what is supplied to the rendering function, which are the functions *incrementStage* and *decrementStage*. *recompose* is than made use of to first define the state of the component, including a default value and a function name to change said value. Furthermore, business logic is implemented, which makes use of the aforementioned state. All these definitions are mapped as properties of the component, so that the rendering function can make use of them. The component is also told to only trigger a new rendering when the state *stage* changes. As it can be seen, *recompose* divided the component into a visual and a logical part, with both being changeable without having to change the other.

```
1  import React from 'react'
2  import {View, TouchableOpacity, Icon} from 'react-native'
3  import {compose, mapProps, withHandlers, withState, onlyUpdateForKeys} from
       'recompose'
4
5  const WizardProcess = ({
6    stage,
7    incrementStage,
8    decrementStage,
9  }) => {
10   return (
11     <View>
12       {stage == 0 && <WizardStageTitle/>}
13       {stage == 1 && <WizardStageTeam />}
14       {stage == 2 && <WizardStageGuests />}
15       {stage == 3 && <WizardStageExercises />}
16       {stage == 4 && <WizardStageDueDate />}
17       {stage == 5 && <WizardStageCheck />}
18
19       <View>
20         {stage != 0 && (
21           <TouchableOpacity onPress={decrementStage}>
22             <Icon name='arrow-circle-left' size={60} />
23           </TouchableOpacity>
24         )}
25         {(stage >= 0 && stage <= 4) &&
26           <TouchableOpacity onPress={incrementStage}>
```

```
27              <Icon name='arrow-circle-right' size={60} />
28            </TouchableOpacity>
29          }
30          {stage == 5 && (
31            <TouchableOpacity onPress={buildVote4FunObject}>
32              <Icon name='check-circle' size={60} />
33            </TouchableOpacity>
34          )}
35        </View>
36      </View>
37    )
38  }
39  export default compose(
40    withState('stage', 'setStage', 0),
41    withHandlers({
42      incrementStage: props => () => {
43        props.setStage(props.stage + 1)
44      },
45      decrementStage: props => () => {
46        props.setStage(props.stage - 1)
47      },
48    mapProps(props => ({
49      stage: props.stage,
50      incrementStage: props.incrementStage,
51      decrementStage: props.decrementStage,
52    })),
53    onlyUpdateForKeys([
54      'stage',
55    ])
56  )(WizardProcess)
```

Listing 5: Simplified Wizard *React Native* Component using *recompose*

Other than the main logic of the setup wizard, the components that are displayed as part of the wizard as well as the pages that are iterated over during the setup are components themselves. Each has their own encapsulated logic and rendering presentation.

5 Future Development

This chapter describes where to pick up development and which features could be implemented next. For future development it is recommended to focus on the unfinished parts before aiming at new features.

5.1 General

We encapsulated the parts we developed from the main application so that they do not interfere with the main application. Nearly all code we produced can be found in the following path: `app/pages/private/sofa`. It contains two folders, one for the poll creation wizard and the other for the poll details page. The wizard itself is also split into two parts, the start page and the process where all sub pages are included.

5.2 Unfinished Parts

Because of the limited time and manpower there are some tasks we didn't manage to finish in time. The following components still need some love:

- During the development the data layout changed a little bit, when we started a training didn't have a team associated to it. With the new layout there now is, so the team picker needs to be removed and the team needs to be passed to the wizard.
- The exercise selector needs additional checks for verifying that a correct number of exercises is selected (between 2 and 4).
- The check and accept page of the wizard lacks crucial information: team and selected exercises.
- The styling of the wizard pages needs to be reworked.

5.3 Additional features

There are some basic features needed so the polling works conveniently:

- Deletion of a already created poll by the coach.
- Actual voting of the players.
- A overview screen of the active poll, for both player and coach.
- A option to change the preferences of a created poll.

5.4 Vote4Fun Object

The data model we worked out with the client looks like this:

```
1 {
2     id: {
3       type: SimpleSchema.RegEx.Id,
4       optional: true
5     },
6     title: {
```

```
      type: String,
      min: 1,
      max: 50
    },
    description: {
      type: String,
      max: 10000,
      optional: true
    },
    trainingTarget: {
      type: String,
      max: 200,
      optional: true
    },
    exercises: {
      type: Array
    },
    'exercises.$': {
      type: Object,
      optional: true
    },
    'exercises.$.referenceId': {
      type: String,
      optional: SimpleSchema.RegEx.Id
    },
    'exercises.$.duration': {
      type: Number,
      min: 0,
      max: 30
    },
    'exercises.$.comment': {
      type: String,
      optional: true,
      max: 1000
    },
    'exercises.$.trainingPhase': {
      type: String,
      optional: true,
      max: 50
    },
    fieldSize: {
      type: Number,
      allowedValues: [125,250,500,750, 1000]
    },
    nrOfPlayers: {
      type: Number,
      optional: true,
      min: 0,
      max: 20
    },
    trainingDate: {
```

```
58        type: String,
59        optional: true,
60        //
              https://stackoverflow.com/questions/3143070/javascript-regex-iso-datetime
61        regEx:
              /(\d{4}-[01]\d-[0-3]\dT[0-2]\d:[0-5]\d:[0-5]\d\.\d+([+-][0-2]\d:[0-5]\d|Z))|(\d
62      },
63      teamSharing: {
64        type: Object,
65        optional: true,
66      },
67      'teamSharing.teamId': {
68        type: String,
69        regEx: SimpleSchema.RegEx.id,
70      },
71      'teamSharing.coach': {
72        type: Boolean,
73        defaultValue: false,
74      },
75      'teamSharing.coordinator': {
76        type: Boolean,
77        defaultValue: false,
78      },
79      'teamSharing.player': {
80        type: Boolean,
81        defaultValue: false,
82      },
83      'teamSharing.vote4fun': {
84        type: Boolean,
85        defaultValue: false,
86      },
87      vote4fun: {
88        type: Object,
89        optional: true,
90      },
91      'vote4fun.type': {
92        type: String,
93        allowedValues: ['clubteam', 'open'],
94      },
95      'vote4fun.title': {
96        type: String,
97        min: 3,
98        max: 40,
99      },
100     'vote4fun.exercises': {
101       type: Array,
102       minCount: 2,
103       maxCount: 4,
104     },
105     'vote4fun.exercises.$': {
106       type: Object,
```

```
107     },
108     'vote4fun.exercises.$.exerciseId': {
109       type: String,
110       regEx: SimpleSchema.RegEx.Id,
111     },
112     'vote4fun.exercises.$.playerIds': {
113       type: Array,
114     },
115     'vote4fun.exercises.$.playerIds.$': {
116       type: String,
117       regEx: SimpleSchema.RegEx.Id,
118     },
119     'vote4fun.deadline': {
120       type: String,
121       //
              https://stackoverflow.com/questions/3143070/javascript-regex-iso-datetime
122       regEx:
              /(\d{4}-[01]\d-[0-3]\dT[0-2]\d:[0-5]\d:[0-5]\d\.\d+([+-][0-2]\d:[0-5]\d|Z))|(\d
123     },
124     'vote4fun.guestIDs': {
125       type: Array,
126       optional: true
127     },
128     'vote4fun.guestIDs.$': {
129       type: String,
130       regEx: SimpleSchema.RegEx.Id,
131     },
132     'vote4fun.showIntermediaResults': {
133       type: Boolean,
134       defaultValue: false,
135     },
136     'vote4fun.showFinalResults': {
137       type: Boolean,
138       defaultValue: false,
139     },
140     'vote4fun.notificationHoursBeforeDeadline': {
141       type: Number,
142       defaultValue: 1,
143       min: 1,
144       max: 24,
145     },
146     'vote4fun.notificationAfterPercentageVoted': {
147       type: Number,
148       min: 0,
149       max: 100,
150     }
151 }
```

Listing 6: *Vote4Fun*

Appendices

Appendix A: Activity Diagrams

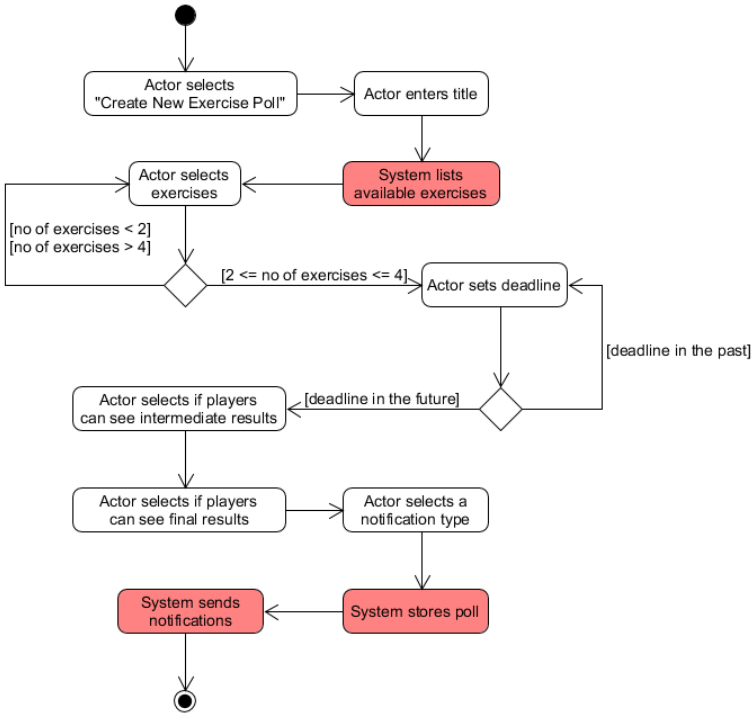

Figure 3: Activity Diagram Create Exercise Poll

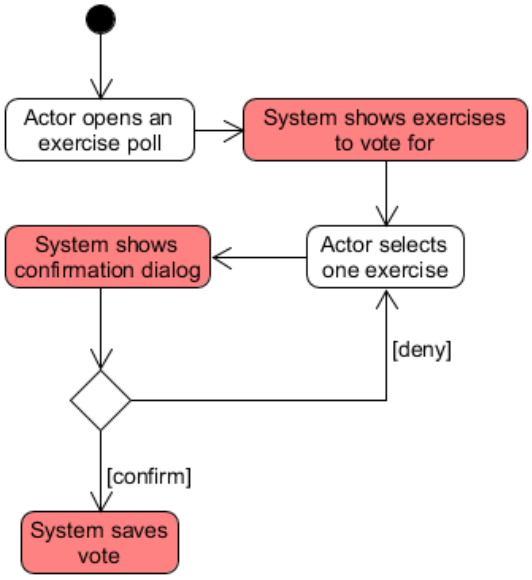

Figure 4: Activity Diagram Vote On Exercise Poll

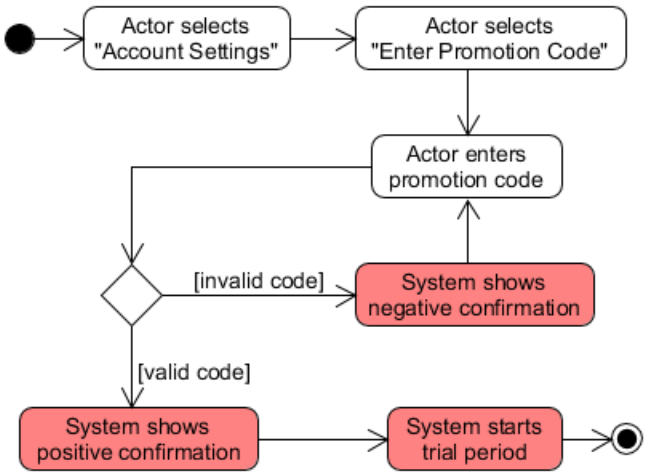

Figure 5: Activity Diagram Enter Promotion Code

Appendix B: State Machine Diagrams

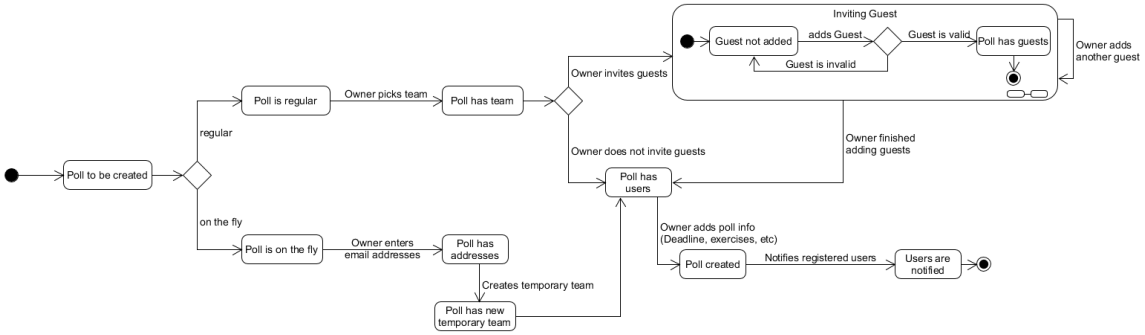

Figure 6: State Machine Diagram Create Poll

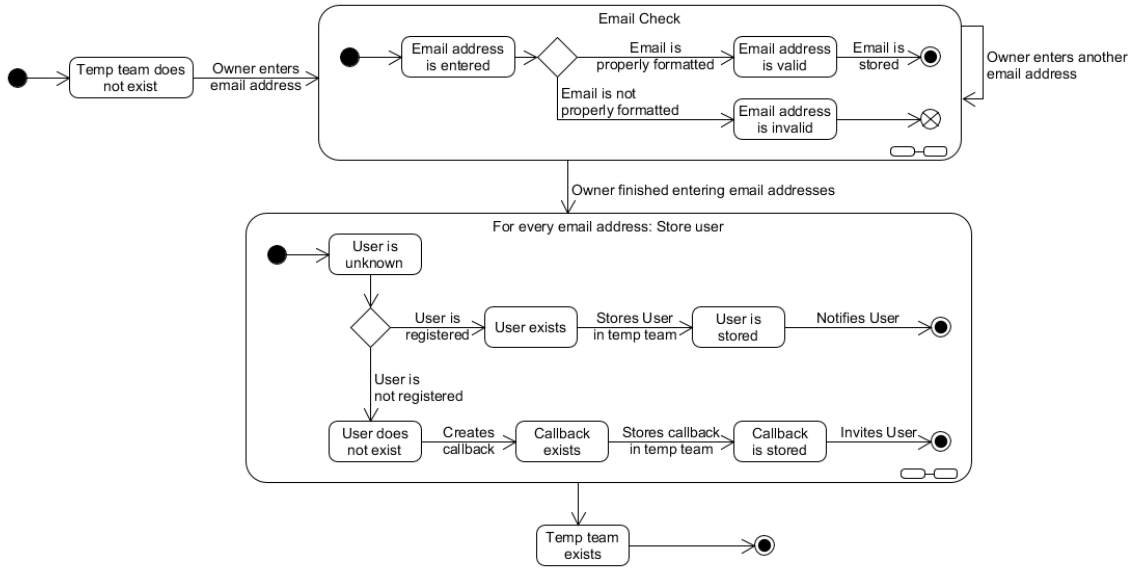

Figure 7: State Machine Diagram Create Temporary Team

Appendix C: Mockups

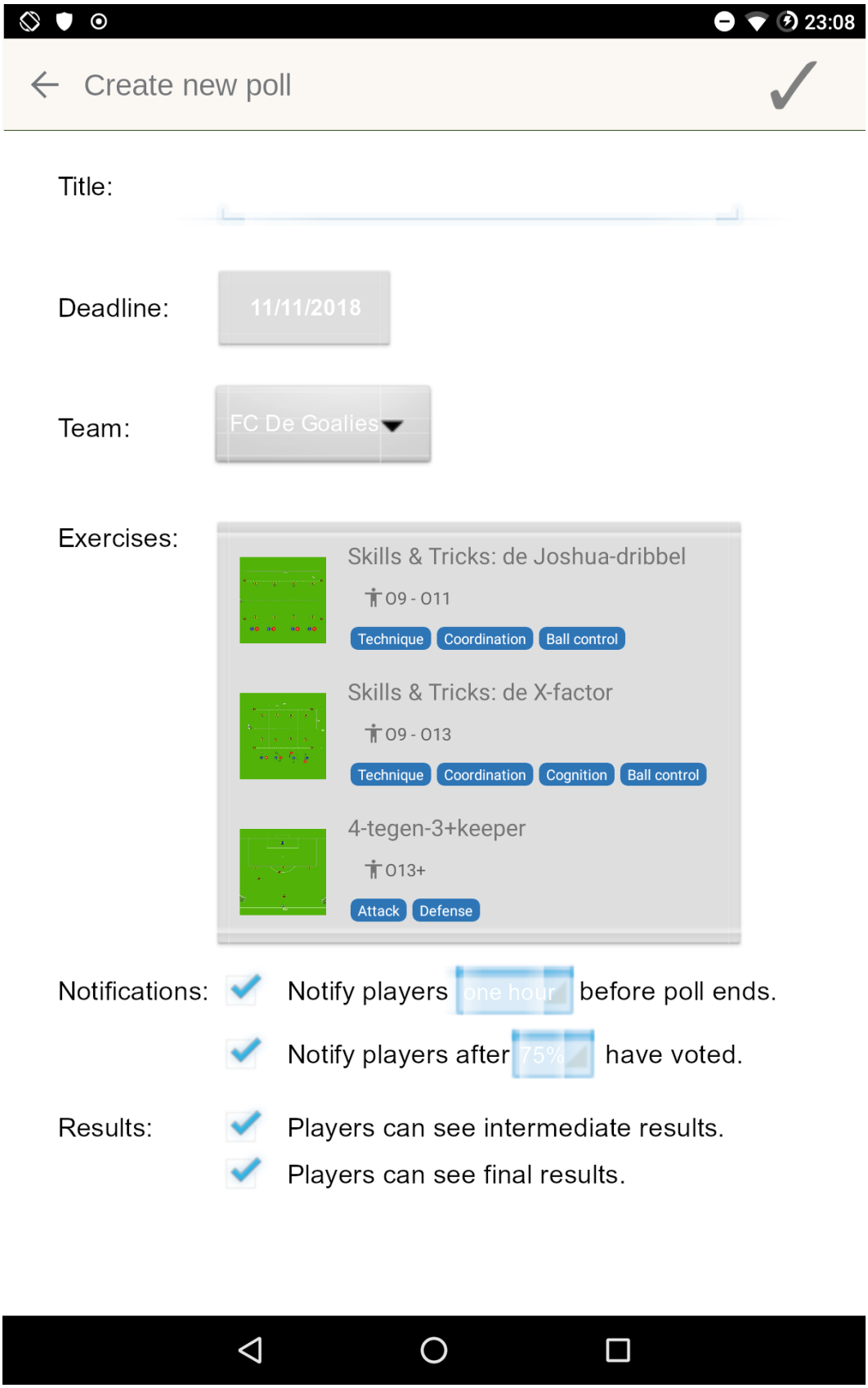

Figure 8: Mockup Create New Poll

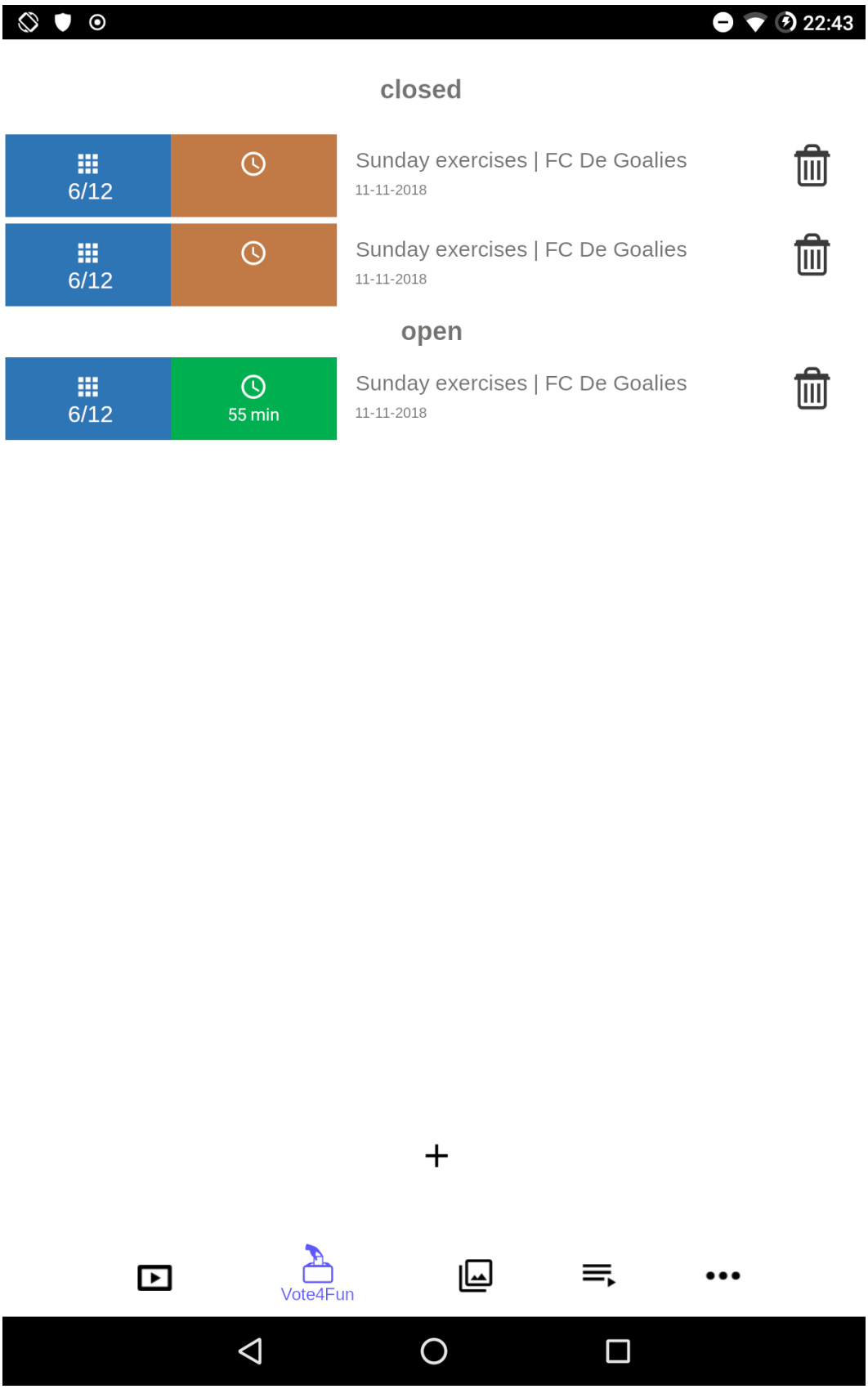

Figure 9: Mockup Poll Overview

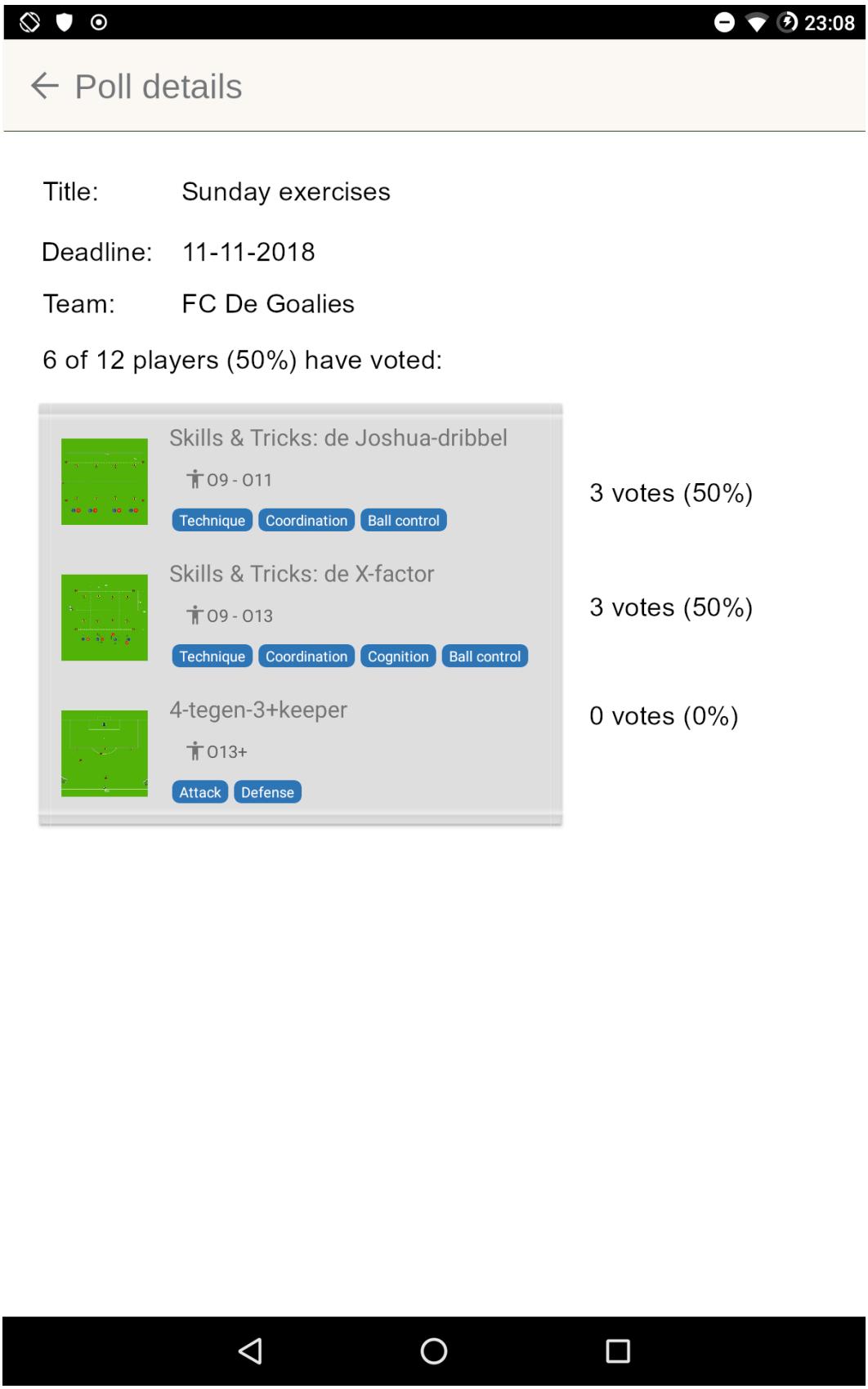

Figure 10: Mockup Poll Details

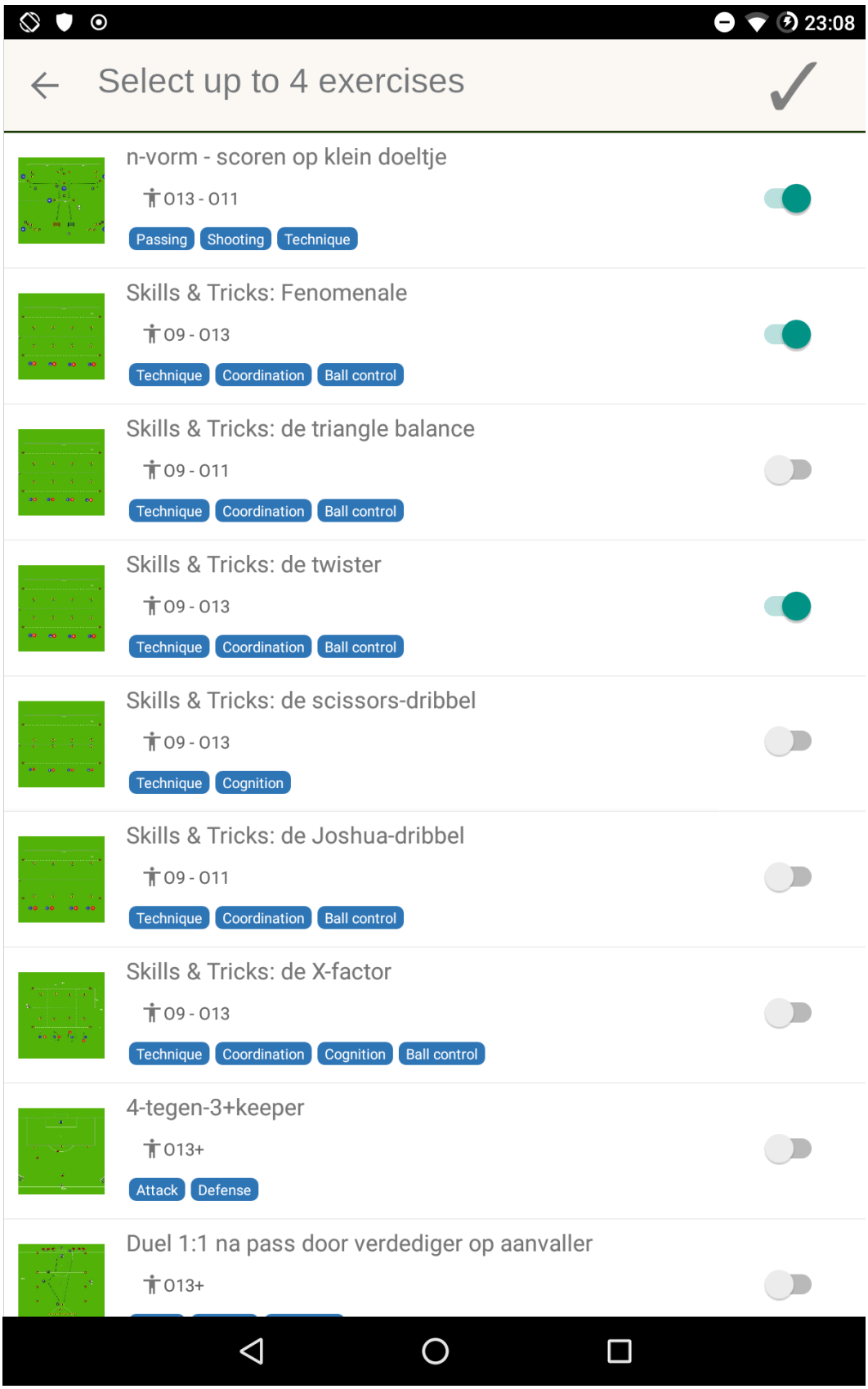

Figure 11: Mockup Select Exercises