

The thesis front page.

Contents

Contents	1
1 Introduction	6
1.1 Relevance	7
1.2 Boundary	7
1.3 External collaboration	8
1.4 Work process	8
1.5 Roadmap	10
2 Pre-study	11
2.1 Scrum tools	11
2.2 Cross-platform frameworks	12
2.3 Back-end frameworks	15
2.4 Distribution of used operating systems	16
2.5 Minimal project	16
3 User survey	18
4 Requirements	21
5 Market analysis	24
6 Architecture	25
6.1 Domain model	25
6.2 Framework architecture	26
6.3 Front-end architecture	27
6.4 Back-end architecture	28
6.5 N+1	29
6.6 Data persistence	30
7 Implementation	32
7.1 Algorithms	32
8 Security	33
8.1 Authentication	33
9 Testing	35
9.1 Front-end	35
9.2 Back-end	35
10 Back-end Analysis	37
10.1 Optics	37
11 Acceptance tests	41
12 Racket description	44
12.1 Terms	44
12.2 Sensors	45
12.3 Features	46
Bibliography	47
A Consultant interview	49
A.1 Hvordan vil et typisk salgsforløb til potentielle kunder forløbe?	49
A.2 Hvilke værktøjer er nødvendige i dette forløb?	49
A.3 Hvad skal programmet kunne udføre?	49
B Android version distribution	51

C Apple iOS version distribution	52
D Racket and sensors	53

Glossary

Angular 2 A JavaScript Single-Page application framework from Google. 13, 15, 17, 26, 27

AngularJS A JavaScript Single-Page application framework from Google. 13

API Interface between applications to enable well defined communication methods. 3

Apollo A JavaScript data transfer framework with reactive data sharing. 3, 16, 17, 28

Apollo Server A part of the Apollo framework managing the communication with the front-end application. 26

Apollo Client A part of the Apollo framework managing the communication with the back-end application. 26

app store A store for applications on a device i.e. *Google Play* on Android and *App Store* on iOS. 29

Bluetooth A wireless technology standard to transfer data over short ranges between devices. 6, 24

Bluetooth Low Energy A wireless technology standard based on Bluetooth but with lower power consumption. 12, 13, 22

Cordova An open source framework for developing cross-platform mobile applications with HTML, CSS and JavaScript. 13

Distributed Data Protocol A client-server protocol based on the publish-subscribe messaging pattern to synchronize changes to the backend database with clients. 13

document database A document database is a non-relational (NoSQL) databases designed to store semi-structured data as documents (rows). 30

Endomondo Mobile application acting as a personal trainer. It tracks movement, helps with training schedules and other training related tasks. 18

Express A minimalistic web framework for node.js. 37

GitHub An online platform for software collaboration through Git version control. 17

GoPro Small HD action camera build to withstand most environments. 18

GraphQL A graphical interface of the GraphQL Schema, useable in a browser both for viewing the schema, but also for querying. 33

GraphQL A query language specification defining the Application Programming Interface (API) between distributed software systems. 15, 26, 33, 37, 39

Hapi a rich web framework for node.js. 37

JavaScriptCore The JavaScript run-time on iOS devices. 26

Koa Next generation web framework for node.js made by the Express team. 37

- Meteor** A JavaScript platform for building web, desktop and mobile applications. 37
- MongoDB** Open Source document database based on a non-relational data schema. 13, 15, 26, 29, 30
- MVC** Software architecture where the application responsibilities are separated into Model, responsible for business logic, View defining the UI and Controller facilitating communications between the view and models.. 27
- MySQL** Open Source database based on a relational data schema. 15
- NativeScript** A TypeScript framework and build-tool set to develop cross-platform mobile applications and compile to native sources. 16, 17, 26, 27, 29, 35
- Node.js** A JavaScript runtime running on multiple platforms that enable developers to write applications in JavaScript without the need of a browser. 13, 26, 28, 29, 30, 32
- NoSQL** Database based on a non-relational data schema. 3, 15
- React** A JavaScript Single-Page application framework from Facebook. 13, 14, 15
- Redux** A JavaScript framework to enable a state container in an application. 15
- Ruby** Ruby is a programming language with focus on simplicity and productivity. 37
- Scrum** Agile software development framework. 8, 10, 12
- Scrum board** A board consisting of columns named after task states. Tasks move from left to right over the board and visualize the progress of the current sprint. 9, 11
- Scrum point** An arbitrary measure of the effort to complete a Scrum task. 8
- Travis CI** Travis CI is a continuous integration server for testing and deploying your application. 35
- type definition** TypeScript files translating JavaScript sources to TypeScript sources. 17
- TypeScript** TypeScript is a superset of JavaScript. It is a strongly typed language that compiles to JavaScript. 28
- user story** Sentence describing a given user demand in the form: As a <user>, I want to <action>. 21, 22, 23
- V8** The JavaScript run-time on Android devices. 26

Acronyms

API Application Programming Interface. *Glossary:* API, 3, 12, 13, 14, 15, 16, 17, 26

BLE Bluetooth Low Energy. *Glossary:* Bluetooth Low Energy, 14

CLI Command Line Interface. 13

CSS Cascading Style Sheets. 3, 13

DDP Distributed Data Protocol. *Glossary:* Distributed Data Protocol, 13

HTML Hyper Text Markup Language. 3

HTTP Hyper Text Transport Protocol. 33

JSON JavaScript Object Notation. 15

MVC Model View Controller. *Glossary:* MVC, 27

ORM Object-relational mapping. 28

OSS Open Source Software. 12

UI User Interface. 14, 27

XML Extensible Markup Language. 13, 27

Introduction

1

This chapter gives a brief introduction to the project and how it is expected to be solved. The main problem to solve is:

Enable athletes to monitor, visualize and track their performances, in an accessible way, when exercising

The overall system aims to enable collaboration between athletes, coaches, administration and other stakeholders in the badminton clubs to gain better achievements and progress for the athletes.

The main part is a "smart" racket that can provide the club, coach and individual player with a bunch of information about their badminton performances. This information can help the coaches find weaknesses and show them where to focus their training for a specific player. The second part is a management system where management staff can handle business such as economics, practice attendance and events such as parties and tournaments. The system is illustrated in figure 1.1.

The vision for this project is to revolutionize club management and progress monitoring. The progress monitoring will be made possible with a mobile application that can gather data over *Bluetooth* from sensors in the smart racket and transfer it to the management

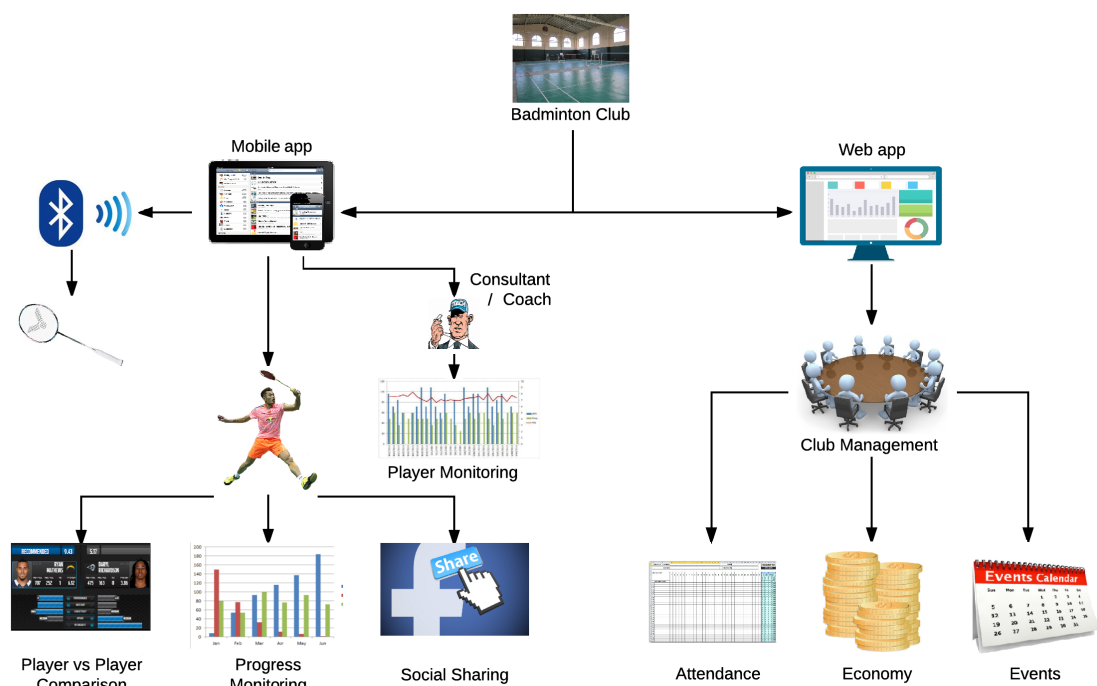


Figure 1.1. The different parts of the project

system. The data will be available on both the mobile device and in the web application for the player, coach and club to analyse.

The club can own smart rackets for their players to use on occasions where the coach gathers the data for both the player and club to see. The player should also have the option to buy its own smart racket to monitor progress at every single practice or match. Data will bind to the player's individual user and will be shareable with the coaches and club. The players can further more compare their progress and stats with their friends and other players from the club and share training sessions and progress on social media.

1.1 Relevance

As sports wearables are on the rise with new products launching every month, most large technology companies like *Apple, Inc.* and *Samsung* are investing heavily into the market and *Gartner, Inc.* predicts that "*From 2015 through 2017, smartwatch adoption will have 48 percent growth(...)*" (Woods and van der Meulen, 2016).

New products that enables users to understand their performances and activities are in high demand, but most of the wearables are centered around wristbands and smartphone applications. These have certain limitations and the precision of the data gathered can be of poor quality. The product of this project is different. It is in the center of the action and gives precise measurements of what forces act on the racket. With this in mind the product is different in the technological sense but fits the same user demands of existing wearables.

1.2 Boundary

Since the vision is more than two man can overcome during this project's duration, boundaries are set for the specific parts that we want to achieve, figure 1.2.

As mentioned in the relevance section, the part that makes this system different from a normal club management system, is the progress monitoring through the mobile application and smart racket. Since this is the unique feature, it is also going to be the focus point for the project.



Figure 1.2. The parts this project will focus on

1.3 External collaboration

As this is a software project we have allied with two external consultants. They are responsible for the development of the physical smart racket and the electronics inside. Further more they have great knowledge of the badminton community and a great personal network with club managers, coaches and players that can be used for testing, research of user demands and alike. They will be referred to as the "consultants" throughout this report. An interview with the consultants were conducted in the initial phase of the project as seen in appendix A.

1.4 Work process

The project process is managed with the agile Scrum development framework as this is the most popular methodology in software development (Johnson, 2015). The flow of events utilized in the process are as follows. A grooming session is held to discuss and define new tasks for the project. This is a returning event in every sprint and makes sure the *Backlog* is prioritized and up to date.

The *Backlog* is holding all tasks to be carried out. They are all estimated with Scrum points, an arbitrary value of how large a given task is. This is an estimate and the team is "learning by doing" what a single Scrum point means. The number of points to

burn down each sprint is set by comparing the former sprints total burndown and the expected available resources in the coming sprint. Each sprint is set for two weeks. This was evaluated in the initial weeks of the project where sprints with one and two weeks duration was tested out. The latter one was what fit us the most as this covered enough work to make progress and little enough to assess.

In the grooming sessions a non-formal *retrospective*, i.e. reflection of the sprint, is held. If any tasks were not finished in the current sprint, it is decided if they should be moved into the new sprint or the *Backlog* for latter completion.

Visualization of tasks and their states is done with an online Scrum board. The tool used was decided in a pre-study, see section 2.1. On figure 1.3 is a snapshot of the current sprint. On the left, column *Ready*, is the tasks ready to be completed in the sprint. Just left of this column is a collapsed column with the *Backlog*. Next to this is the *In Progress* column holding all tasks currently being worked upon. After a task is done it moves to the *Review* column where it is reviewed by a team member. When the review is done and there are no comments or they are resolved, the tasks is placed in the *Done* column.

To make the management of tasks as easy as possible, the used tool synchronizes with *Git* branches, i.e. in progress work in version control. When a new branch is created in *Git*, referencing a task number, it is moved to *In Progress*. When a *Pull Request*, i.e. the changes are ready to be merged into the *master* branch, the task is moved to *Review* and finally if the *Pull Request* is closed and merged, the task is put in *Done*.

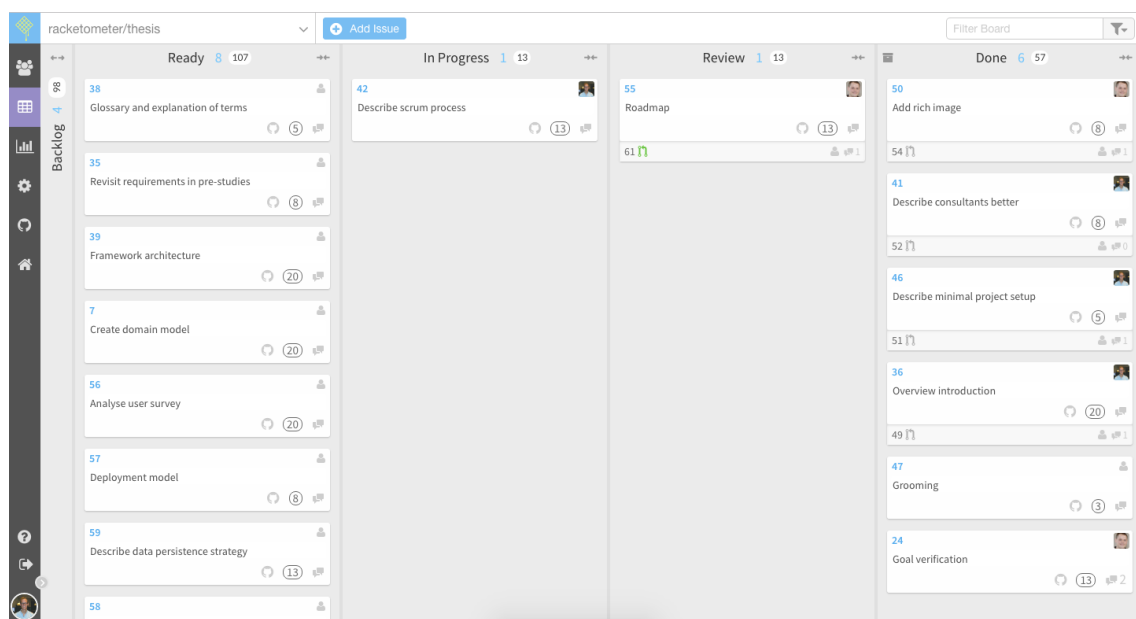


Figure 1.3. Snapshot of the Scrum board

1.5 Roadmap

In planning what we can achieve during this project, a roadmap was created with milestones and dates. The roadmap focuses on the progress of the application during the project. The roadmap will be used during Scrum planning and might change over the duration of the project. See the roadmap on figure 1.4.

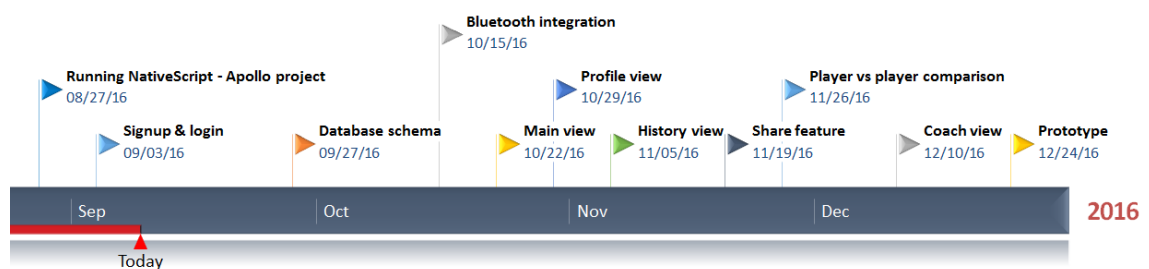


Figure 1.4. Roadmap

Pre-study

2

Here follows pre-studies on technologies and tools to be used throughout the project, conducted in the project's initial phase. See chapter 4 for more details of the requirements.

When researching technologies, focus will be on new cutting edge technologies. This comes with a risk as they are not as battle tested as older technologies. They often have more bugs and this can cause problems during development but they also often provide better performance and new features or advantages over their counterparts. The trade-off is assessed to be worth it as we as developers get experience with new technologies that might be relevant for years to come.

2.1 Scrum tools

This pre-study is aimed to finding a useful tool for managing the agile software development work flow carried out throughout the project.

Here follows a list of features that is, more or less, requirements. Further more it should preferably be free to use for Open Source Software.

Must haves

- A Scrum board to organize tasks in categories, e.g. *Ready*, *In progress* and *Done*.
- A way to manage a backlog of tasks to do.

Nice to haves

- Automatic integrated handling of task progress with source control tools, i.e. *GitHub*.
- A way to visualize the history of how tasks got processed.

In a general search of solutions the products *Trello* from Trello, Inc., *ZenHub* from Launch Labs, Inc. and *Waffle* from CA Technologies was spotted and further analyzed.

Here follows a brief description of each product.

Trello

A simple To Do list web application that enables users to arrange *cards*, i.e. tasks in columns of the users desires and as such fulfills the *must have* requirements. *Cards* can be assigned labels, check lists, members and has its own discussion board associated.

ZenHub

A fully featured project management tool to handle agile processes around software development. It is integrated with the *GitHub* web application, that is used for source control of the software, by use of a browser extension and enables the users to handle sprints, work flow visualization and integration with *GitHub* task management. This product meets the *must have* requirements as well and further more integrates with *GitHub* and enables visualization of the work done. It does however require the users to install a browser extension on each development environment.

Waffle

A web application powered by *GitHub* work flow actions. It visualizes the current tasks to do, tasks doing and tasks ready for review in a user specified column setup. It allows for creating new issues and more from the web application and picks up changes from *GitHub* as well. It meets both the *must have* and *nice to have* requirements.

Conclusion

All products meet the *must have* requirements but only *ZenHub* and *Waffle* integrates with *GitHub* and they are all free to use when working with Open Source Software (OSS). Between *Waffle* and *ZenHub* a main difference is that the latter one requires installation of browser extensions that currently is supported on the *Chrome* browser from Google, Inc. and *Firefox* browser from Mozilla, Inc. (CA Technologies, 2015). *Waffle* is instead accessible to everyone through the website and because of this *Waffle* is chosen as the Scrum tool.

2.2 Cross-platform frameworks

This section will evaluate different frameworks for cross-platform development which could be used in the project.

It is considered a requirement that the application works on multiple platforms as the initial interview with the consultants, appendix A, established the needs for this. Further more the existing racket hardware communicates with Bluetooth Low Energy. When working with new technologies the community is an indicator of the adoption of the technology and is considered important, as to avoid learning short-lived technologies.

The *nice to have* requirements are defined from a developers point of view. If the framework supports both web and mobile applications functionality can be reused and thus reduce development effort and costs Type safety also reduces development costs as less errors are introduced by the developer when the compiler detects mistakes. To follow modern standards in mobile, and web, development, data must be reactive, as this is expected by the users. If the code base compiles to native APIs as well, performance is increased and thus the user experience is improved.

The *must have* and *nice to have* capabilities are summed up below.

Must have

- Cross-platform capabilities.
- Wide community support.
- Bluetooth Low Energy capabilities.

Nice to have

- One framework both for web and mobile applications.
- Type safety in the programming language.
- Reactive data. Updated data in the database should reflect on the clients.
- Compiles to native code for increased performance.

Meteor

Language: JavaScript

Description:

Meteor is a framework for developing both web and mobile applications. It comes with Cordova built-in and is considered super fast for developing applications and prototyping. It comes with a lot of built-in tools and a package library for easy setup of things like user and login integrations of different kinds, e.g. *Google* and *Facebook*. It comes with hot code push which is a tool for uploading new versions of your application to the App-stores.

Meteor has its own Command Line Interface (CLI) which is very useful when developing and it allows one to write the back-end in plain JavaScript. Out of the box it integrates with *MongoDB* and it uses *Distributed Data Protocol (DDP)* for sending data between the client and the server. This data is reactive and will be updated at the clients when it changes.

Meteor comes with its own view layer named *Blaze* but this can be substituted with *React*, *AngularJS* or *Angular 2* if any of them are preferred.

NativeScript

Language: JavaScript

Description:

NativeScript uses a JavaScript toolchain to compile the JavaScript code into the native API. This is a huge advantage because it makes the application work exactly like it was written natively without ever writing any native code or knowing about the native APIs. The markup is written in an Extensible Markup Language (XML) like language and is styled with Cascading Style Sheets (CSS). It is developed by Telerik, a big software company where one can opt-in for support through a paid license.

It is supported by Google and therefore works well with *Angular 2* and TypeScript and with this the *nice to have* feature "type safety" is fulfilled. *Bluetooth Low Energy* capabilities can be achieved with the Node.js package *nativescript-bluetooth*, created by *npm* user

eddyverbruggen. *NativeScript* supports *iOS*, *Android*, *Universal Windows Platform* and if *Angular 2* is used it also supports building web applications (Stoychev, 2016).

ReactNative

Language: JavaScript, Objective-C and Java

Description:

ReactNative is a framework by Facebook, Inc. It extends the *React* framework for building native applications. *ReactNative* requires the developer to know Objective-C for *iOS* development and Java for *Android* development. This does not fulfill the requirement that it has to be cross-platform without the developer needing to know about the distinct platform languages and APIs.

Xamarin

Language: C#

Description:

Xamarin was recently purchased by Microsoft. It is a cross-platform framework for developing applications for *iOS*, *Android*, *Windows Phone* and *Windows Universal Platform*. There are two main ways of developing with *Xamarin*. *Xamarin forms* which is the method that has the highest amount of code-sharing between platforms. It is estimated around 80-95% code sharing depending on how specific the User Interface (UI) has to be. The other is *Xamarin native* which has around 50-60% code sharing with mainly business login. With this the UI implementations differ for each platform. When using *Xamarin native* it is a huge advantage to know about the native UI APIs.

Requirements summary

This table shows how each framework fulfills the specified requirements.

	Meteor	NativeScript	ReactNative	Xamarin
Cross-platform	✓	✓	✓	✓
Community	✓	✓	✓	✓
BLE capabilities	✓	✓	✓	✓
One framework	✓	✓	✗	✗
Type safety	✗	✓	✗	✓
Reactive data	✓	✗	✗	✗
Native performance	✗	✓	✓	✓

Conclusion

NativeScript fulfills most of our requirements out of the box. The only thing that is not support is reactive data, however this can still be achieved with other tools. Therefore the selection falls upon *NativeScript*.

2.3 Back-end frameworks

This section will evaluate different frameworks and services to support the project with data storage and possibly for running heavy jobs that are not preferred to run on the client applications.

As a minimum, the back-end must have the capabilities to store and retrieve data from some kind of database and as the data to store is personal records of the user's performances, access to the back-end must be limited with authentication.

In the "nice to have" part, an implementation independent interface for the clients is preferred, i.e. a generic API. This makes sure the clients do not become dependent on the back-end implementations. As with the cross-platform frameworks, the back-end must support some kind of reactive data structure to follow modern standards.

The *must have* and *nice to have* capabilities are summed up below.

Must have

- Be able to store and retrieve data.
- Support authentication.

Nice to have

- Generic API.
- Reactive data. Updated data in the database should reflect on the client.

Firebase

Function: Reactive Data storage

Database type: NoSQL

Description:

Firebase is a database service from Google, Inc. It has lots of features one do not normally see on a database, e.g. user authentication. It is a real-time database, i.e. the data is reactive, so whenever there is a change in the database, this is pushed to the clients. Pushing data into the database is done with JavaScript Object Notation (JSON) and running specific functionality is not supported. It is however possible by use of a commercial product from *Zapier*¹.

Apollo

Function: Reactive Data storage

Database type: None specific, free to choose. Client application interface is *GraphQL*.

Description:

Apollo is a modern data stack that offers reactive data. This means that your data will always be up-to-date if desired. It is a data stack provided by the *Meteor* team, but as opposed to the *Meteor* framework, *Apollo* is to be used in any application.

¹FiXme Fatal: Reference to this?

There are integration guides for *Angular 2*, *Redux*, *Meteor*, *React* and *React native*, but one should be able to integrate it into any JavaScript front-end. *Apollo* consists of two parts. A schema that describes the type of data available on the server and a query language, *GraphQL*, that enables the client to describe the data it needs. This decouples the client from the database type as *GraphQL* is only a query specification. *Apollo* can query different types of database, e.g. *MongoDB* and *MySQL*, without the client realising it. As the client specifies what properties are needed, no unused data is send over the wire.

Requirements summary

This table shows how each framework and service fulfills the specified requirements.

	Firebase	Apollo
Be able to store and retrieve data	✓	✓
Support authentication	✓	✓
Generic API	✗	✓
Reactive data	✓	✓

Conclusion

Apollo fulfills all of the requirements and provides an excellent separation from the clients and server. With *Firebase* the client application is bound to it. *Apollo* lets the developer pick what database to be used and can support several, and different types, of databases. This is at the expense of more work by the developer but a considerable better flexibility why *Apollo* is the selected back-end.

2.4 Distribution of used operating systems

The mobile and tablet operating system market is dominated by two products, i.e. *Android* (65.01%) and *iOS* (28.00%), running on 93.01% of all devices as of October 2016. Other systems include *Windows Phone* at 2.72% and *Java ME* at 1.79% (Net Applications, 2016). The latter ones are omitted in the following, as the market share is below 10%.

Several versions of each system are in broad use. *iOS* is dominated by versions 9 and 10 covering 92% of usage (Apple, Inc., 2016). *Android* versions are more scattered. Most used is version 4.4 *KitKat* covering 25.2% followed by 6.0 *Marshmallow* covering 24.0% (Android, 2016).

The references for these numbers are updated regularly why the statistics are replicated in appendices B and C.

2.5 Minimal project

In the pre-studies each framework and technology was researched and the theoretical cooperation between them was taken into account. As the used technologies are all very new and the combination of them are not practically known to work, a minimal project was created to verify this.

Most frameworks supply simple *Getting started* projects to showcase the framework in action. This was also the case with the selected frameworks *NativeScript* (NativeScript, 2016) and *Apollo* (Apollo Stack, 2016).

As *NativeScript* is the main framework to build the application with, their sample project was used as a base. From this the *Apollo* sample was studied and merged into the project to verify the cooperation. This proved to have some complications as *Angular 2* was still in release candidate versions, with major API changes between each version, and the samples were not supporting the same version.

By studying the commit history, in the *GitHub* version control repositories, of both the *NativeScript* and *Apollo* sample, a compatible version set was found. Some technical complications regarding the type definition was causing some compile time complications and some manual type definitions was written to overcome this. By use of the *Apollo* server sample, a simple back-end server was set up locally to allow the mobile applications to communicate with a server instance. When this was set up the frameworks were able to perform a simple query to the back-end and receive data from the database.

As of this we were sure the frameworks could cooperate and confident enough to proceed the development process with them.

User survey

3

In the initial phase of the project a user survey was distributed over social media to get an understanding of the product user group. In the period August 29th 2016 to September 20th 2016, the survey received 529 responses of which 425 was users playing badminton.

Racket price

When comparing the price ranges to user groups it shows that most users have a racket in the price range kr. 1000 - 1499 as seen on figure 3.1. It is interesting that in the age of 19 to 24 years users spend the most money on their rackets. The price ranges can be used to find a proper cost for the product and/or services offered.

Gadget knowledge

When asking what sport-gadgets the users know, the results show a clear pattern as seen on figure 3.2. *Endomondo* is the most known gadget closely followed by *GoPro* with 87% and 73% respectively. A key difference between these are clear when looking at how many users owning it. Only 17% of the users knowing GoPro owns one whereas it is 52% with Endomondo. An explanation on this could be that Endomondo has a free version of the application with limited features. *GoPro* offers no such thing as it is a physical product.

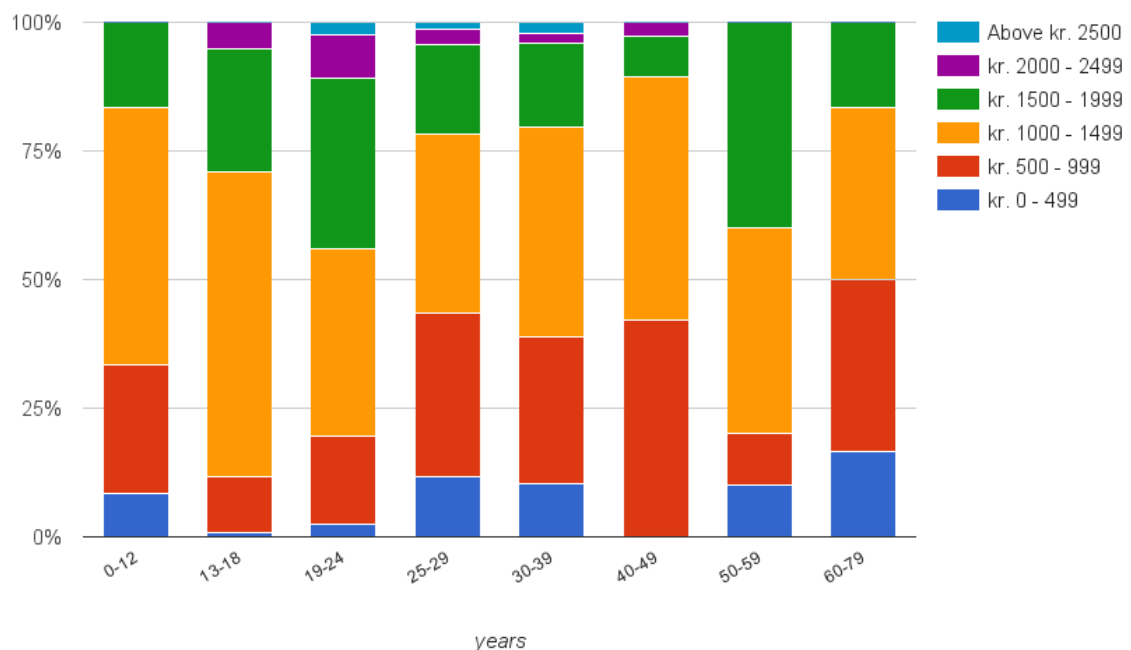


Figure 3.1. Price distribution versus age groups

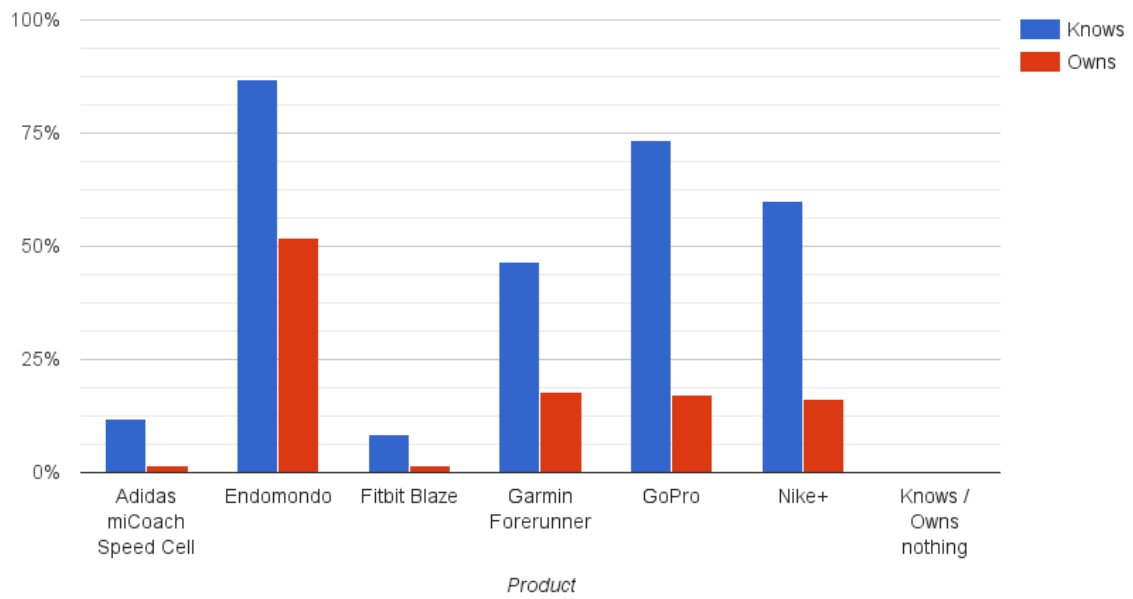


Figure 3.2. Knowledge and ownership of sport-gadgets

Information sharing

The survey asked the users if they were willing to share their recorded information to improve the algorithms behind the calculated results. 81% approved this, figure 3.3. Further more the users was asked if they would share their recordings to match other players for ranking and alike and only 62% approved this. As the difference between these two options are quite subtle, i.e. the anonymous part of it, it is clear that this must be taken into account when creating the product.

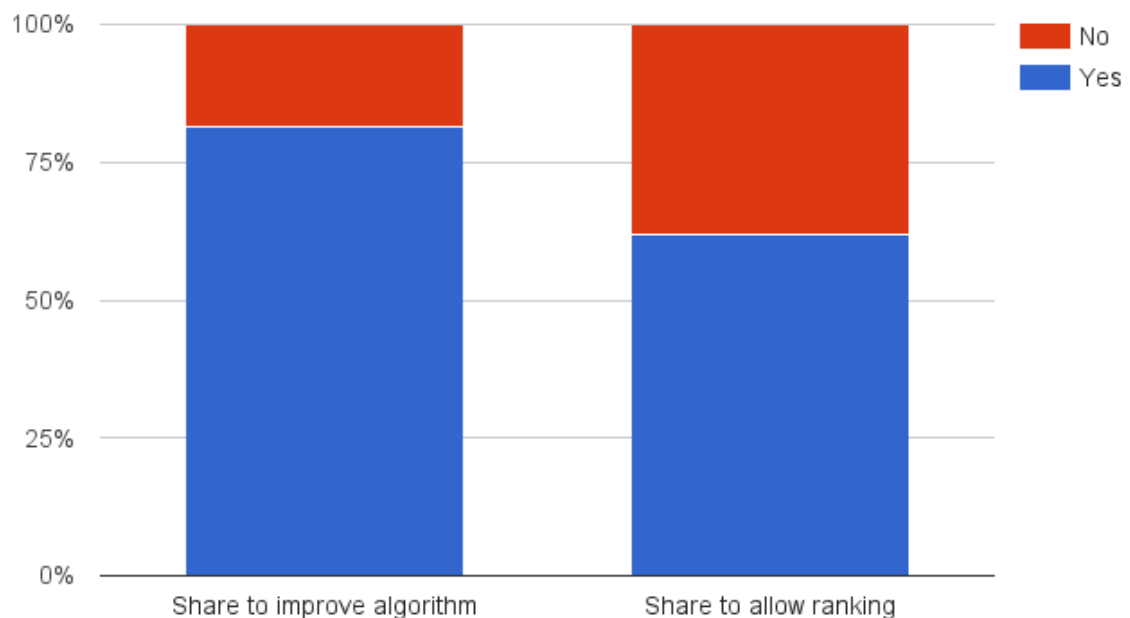


Figure 3.3. Connection of anonymous data to improve algorithm and user data to allow ranking

Implementation options

When asking for possible sensor implementation solutions for mounting the sensors to the racket, the most popular option with 57% of the answers, was mounting a sensor on an existing racket, as seen on figure 3.4. This is the least intrusive solution as well, why this was expected. The initial approach from the consultants was to mount the sensor inside a racket. Of the users in the survey, 36% were willing to buy a new racket with the sensor built in, why this is a viable solution for the prototype.

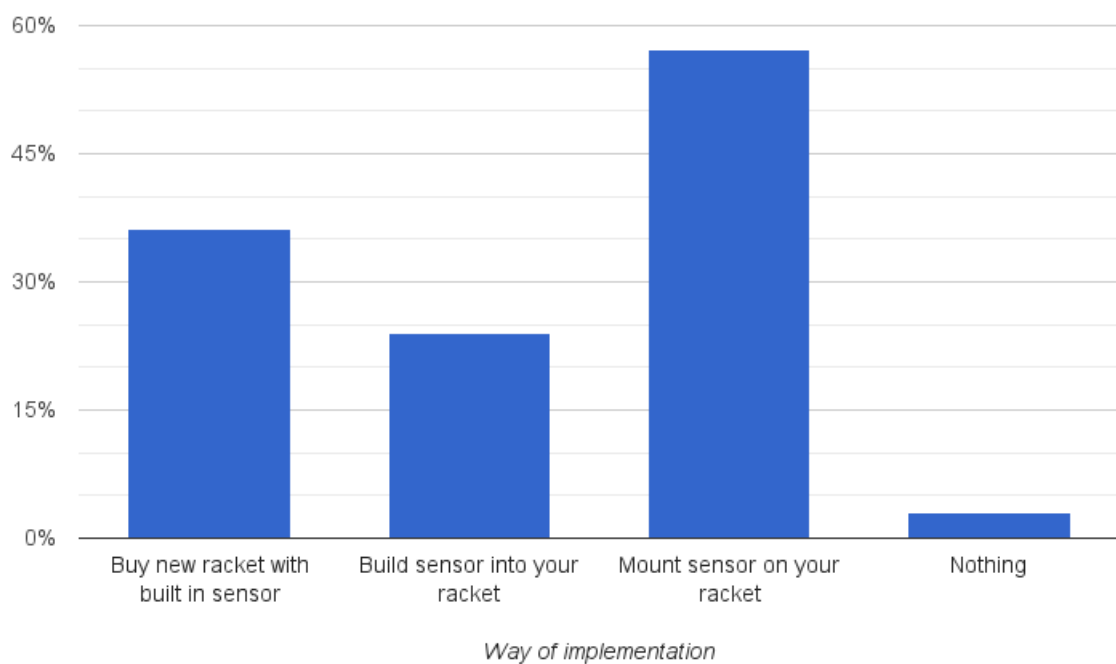


Figure 3.4. Sensor implementation possibilities on racket

Requirements

4

This chapter defines the requirements for the project. The functional requirements are described with *user stories* with some additional conditions of satisfaction to increase the testable details and add non-functional requirements.

User stories

As a consultant / user, I want to use a phone or tablet to access the system

This *user story* is specified in the consultant interview appendix A.2.

Device operating systems to support includes:

Apple iOS 9 and 10

Android 4.3, 4.4, 5.0, 5.1, 6.0 and 7.0

This covers 92.4% and 92% of *Android* and *iOS* devices respectively, according to the distribution tables in appendices B and C.

As a consultant, I want to input user information when beginning a session and automatically add them to the user database

This *user story* is specified in the consultant interview appendix A.1.

If internet is present, the user information must be submitted immediately to the database.

If internet is not present, the user information must be stored on the device for later automatic submission.

Information to input must include the following fields:

name **string** The full name of the user

email **string**

birthday **Date**

experience **int** Years of experience

racketType **RacketType** Type of racket from predefined list

isComparisonAllowed **boolean** Allow comparison to other users

As a consultant, I want to start a session with a user and record data from the racket

This *user story* is specified in the consultant interview appendix A.1.

Information is transmitted from racket to the system with *Bluetooth Low Energy*.

Information saved for a session must include:

consultant **Consultant**
date **Date** Date of the recording session
location **Location**
racketDate **RacketData** Recorded racket data
user **User**

As a consultant, I want started sessions to be persisted between application restarts

This is an expected behavior and are therefore added as a requirement.

As a consultant, I want to find a previously stored user

This *user story* is specified in the consultant interview appendix A.1.

If internet is present, the list of stored users are updated to reflect the users stored.

If no internet is present, the cached list of users are listed.

As a consultant, I want to show details of recorded data to specific users

This *user story* is specified in the consultant interview appendix A.1 and A.3.

Measured data to be presented as numbers or graphs:

strokes **int** Number of strokes
types **Array<StrokeType>** Most used types of strokes from predefined types
maxRacketSpeed **double** Maximum racket speed in m/s
maxShuttleCockSpeed **double** Maximum shuttlecock speed in m/s

As a user, I want to receive an email with a generated password for my account, when a consultant have created an account on my behalf

This *user story* is specified in the consultant interview appendix A.3.

When a consultant creates an account on behalf of a user, the user should be able to access that account with an auto generated password send to his account after creation.

If internet is present, the email must be submitted immediately.

If internet is not present, the email must be stored on the device for later automatic submission.

As a user, I want to be able to change my password from within the application

If internet is not present, this operation is not possible.

As a user, I want to compare my performance data to other users

This *user story* is specified in the consultant interview appendix ¹

Users that allows comparison must be able to be matched on ranking lists.

Ranking lists must be available on the following metrics:

Number of total strokes

Number of strokes on specific predefined stroke types

Racket speed

Shuttlecock speed

As a user, I want to share my user profile on social media

This *user story* is specified in the consultant interview appendix ²

Social media includes *Facebook* and *Twitter*.

The shared information must be a summary of the performance, possibly edited by the user, and a link to the complete user profile on the web page.

As a user, I want to see my recorded data compared to historical data

This *user story* is specified in the consultant interview appendix ³

If a user has multiple recorded sessions, metrics must be presented in comparison with the previously recorded metrics.

¹FiXme Fatal: Add reference to user survey appendix

²FiXme Fatal: Add reference to user survey appendix

³FiXme Fatal: Add reference to user survey appendix

Market analysis

5

This chapter describes the findings of a market analysis for similar products. Here follows a short description of each product found.

SOTX Smart Badminton Racquet

SOTX have made the product series *Smart Badminton Racquet*. These rackets are labeled as having sensors and a microprocessor built into the handle and can communicate with *Android* or *iOS* smartphones via *Bluetooth* (SOTX Sports Equipment Company, Ltd., 2016). It supports both live streaming as well as transferring data at the end of a match. It seems to come with a racket charger you can plug your racket into (Hongzuo, 2016), but it advertises wireless charging, how those two things work together is not documented.

From the look of the app in the feature picture on an article on *LinkedIn*, it seems to focus mostly on what kind of swing you perform (Wong, 2016), but can also keep track on how long you have been working out, calories burned and maximum speed.

Holy Pie Smart Racket

In 2014 a *Kickstarter* campaign called *Holy Pie Smart Racket* launched (Dian, 2014). It successfully raised \$2010 with four backers. They made a system for both badminton and tennis. It is not easy to tell what this product can do since their website seems unfinished as most of the links do not work and there is still dummy text present (Shenzhen Earth Electronics, Ltd., 2016).

USENSE

USENSE is a sensor that is mounted at the bottom of the racket shaft. There is an *Android* and *iOS* application so that it can connect to your smartphone through *Bluetooth* 4.0+ or 2.0. Both specifications are mentioned (Voices of Spring, 2016).

This one is significantly different from the other two because you do not need to buy a new badminton racket, you only buy a sensor and mount it on your own racket. A downside might be that it is not integrated into the racket and changes the balance of the racket.

Reflection

All of these products seem very undocumented and their websites are either unfinished or cumbersome to understand, especially the English language ones. They are only available through smartphone applications and not on desktops or web which may be useful for coaches.

Architecture

6

This chapter describes the architecture used in the project.

6.1 Domain model

Given the above introduction a domain model was derived, figure 6.1, to make the domain entity relations visible. This is done to make sure the understanding of the relations are clear before doing any further development. It is important to note that this is a tentative model and as long as the project is under development, it is subject to change.

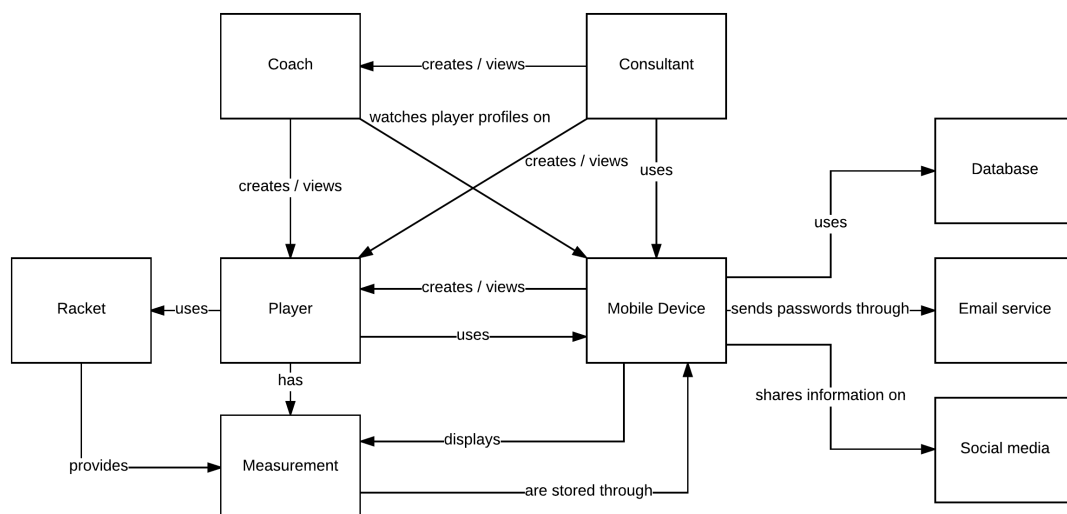


Figure 6.1. Domain model

6.2 Framework architecture

Multiple frameworks are used in the project. This is a short description of the tasks each framework is responsible for. An overview is shown on figure 6.2.

At the application layer of the front-end Angular 2 is responsible for the view rendering and business logic. It communicates with the Apollo Client, that is responsible for communicating with the back-end, as well as holding application state. Between the front-end and back-end the API is defined with GraphQL.

The client application layer is run with the NativeScript run-time. At compile time, the Angular 2 views and styling is rendered to native UI elements and styling. The application logic is held in JavaScript and run on a platform specific JavaScript engine, i.e. V8 for Android and JavaScriptCore for iOS.

On the back-end the application layer is an Apollo Server instance. This is the counter part to the Apollo Client and provides the reactive data system. It further defines the API for collecting data from databases like MongoDB, as this project utilizes. Both of these frameworks run on the Node.js run-time, which in turn runs on the V8 JavaScript engine.

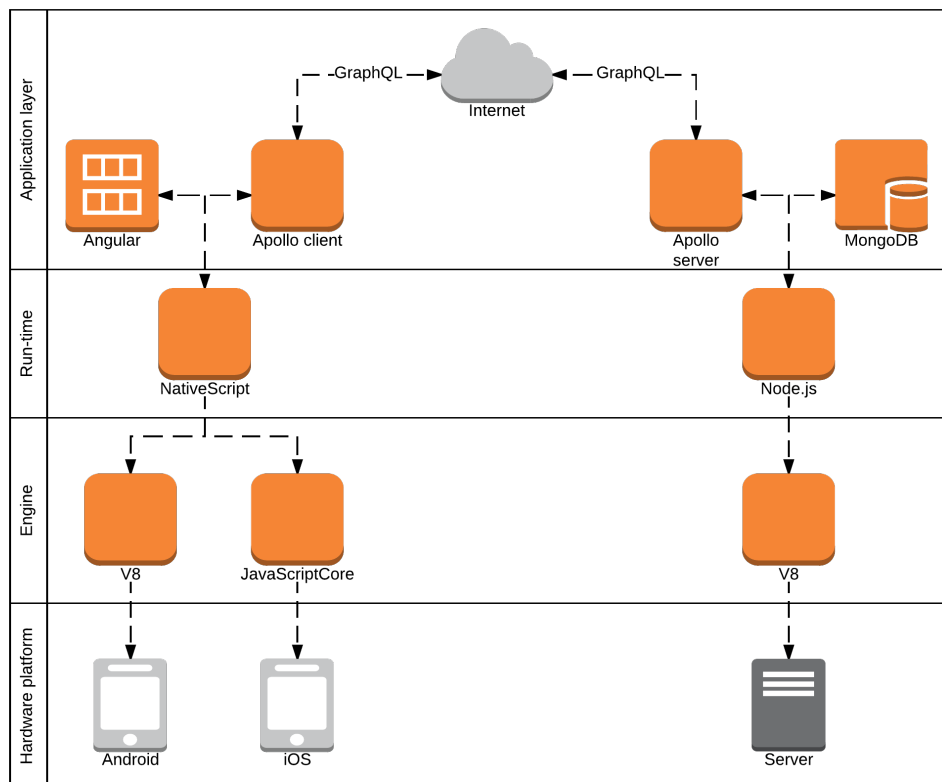


Figure 6.2. Framework architecture overview

6.3 Front-end architecture

The front-end application is written in Angular 2. The framework utilizes a component structured Model View Controller (MVC) architecture where each component represents the VC part and services cover M (Angular, 2016), as shown on figure 6.3.

A component is split into two parts. The **template**, specifically in NativeScript, is written in a language similar to XML and represents how a component should be rendered in the UI. It communicates with a **controller**, the component class, with events and get information with property bindings. It can communicate with services that are constructor injected by the Angular Dependency Injection system.

A **service** is responsible for the business logic and can communicate with a back-end or perform calculations etc. It is possible to do this in the controllers as well, but it makes them harder to unit test and further more adds logic that is not view specific.

The view is build from individual components, each responsible of a clear part of the whole UI. As of this, the complete view can be illustrated as a component tree where each component can propagate events up and down the tree and communicate with services.

On figure 6.4 the component tree of a simple login view is illustrated. It contains an input component that uses some validation component and a button. The root component **Login** subscribes to click events on the button and change events on the input field. When the button is clicked, it initiates the authentication routine in the **Authentication** service.

As the application grows more complex branches of the tree can be modularized. This makes the structure easy to mangle with and small parts, i.e. components, of the view can easily be reused.

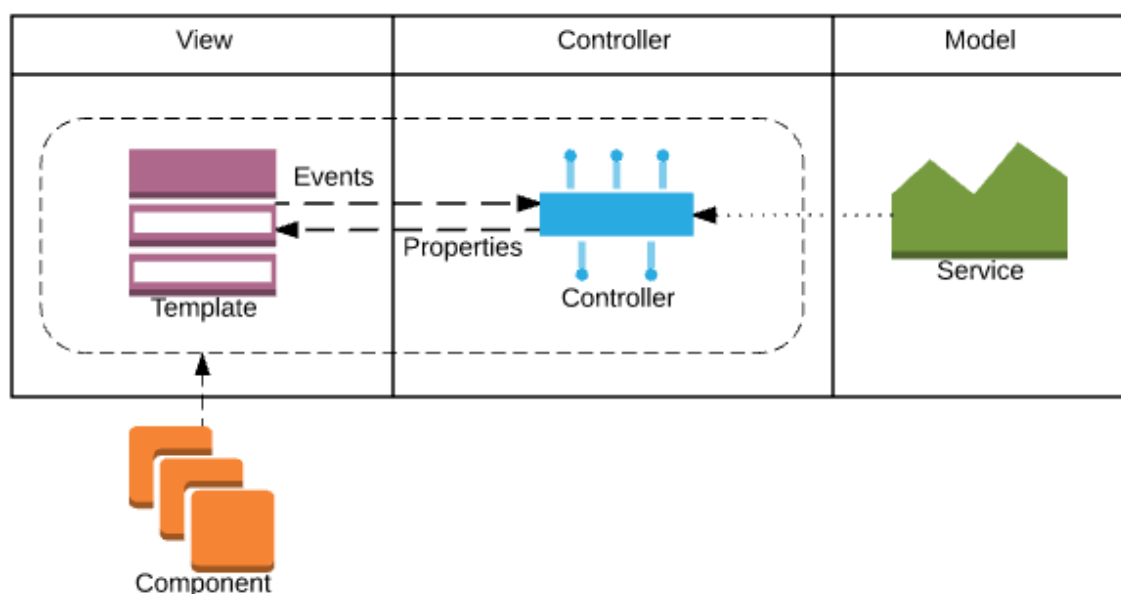


Figure 6.3. MVC component with Angular 2 terminology

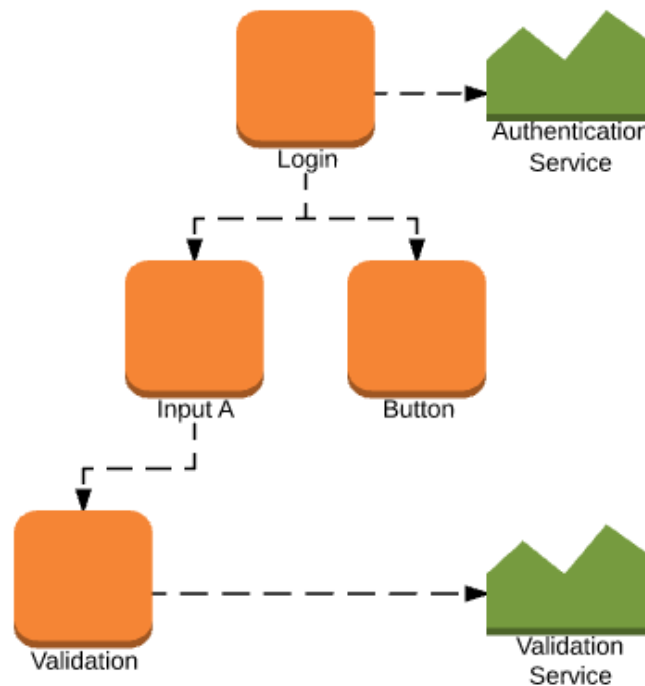


Figure 6.4. Component tree of a simple login view

6.4 Back-end architecture

The back-end application is written in TypeScript and build with a framework called Apollo.

The back-end is spun up using Node.js and Express which is a framework to spin up a server in a Node.js environment.

The back-end contains 3 layers of abstraction which are illustrated on 6.5.

The top layer is the one that gets exposed to the users of the back-end. The schema consists of models and functions that the user can query for. The Schema is written in the GraphQL Schema Language.

When the user queries, Apollo looks for a resolve function to resolve the query. This layer is called the Resolvers layer. Here, functions are defined to resolve specific queries related to the schema.

The data access layer contains 2 abstractions. The Connectors are where connections to databases are made and exposed to the above models layer. These databases are used in different ways. In the models layer, these are abstracted away so that the resolvers don't need to change, if the database or the Object-relational mapping (ORM) changes in the connectors layer. The models layer then exposes a generic API for finding, modifying, creating and removing persisted data

Besides the 3 layers that make up the Apollo server, we also have 2 services. One for racket algorithms used to calculate features based on session data and an email service used to send out emails to users when they are signed up or when they change passwords.

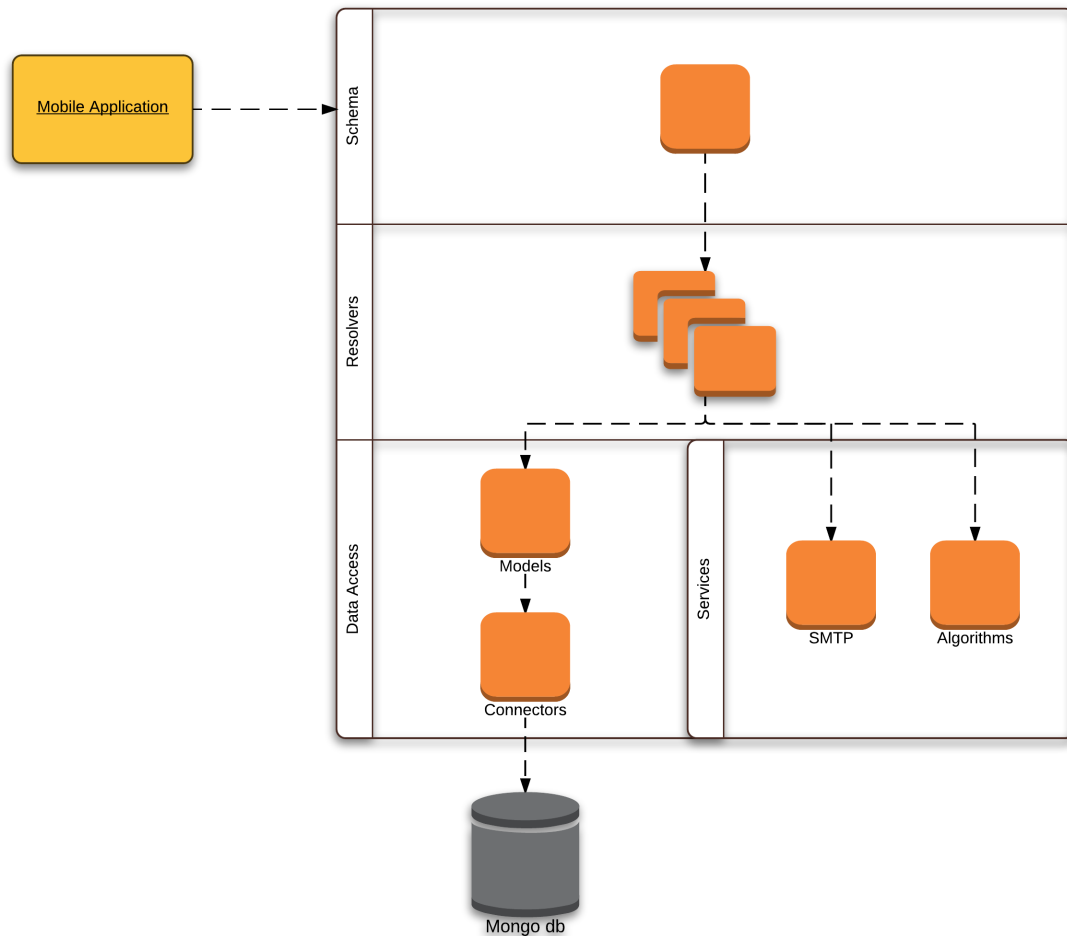


Figure 6.5. The back-end architecture layers

6.5 N+1

6.5.1 Deployment model

The deployment model is meant to illustrate where the different software packages are deployed, as shown on figure 6.6.

The *NativeScript* codebase is compiled into 2 different file extensions i.e. `.jar` for *Android* and `.app` for *iOS*. The compiled files are then deployed directly to their respective devices or uploaded to each platform's individual app store.

The back-end codebase is deployed on a web server with a Node.js environment as well as a *MongoDB*.

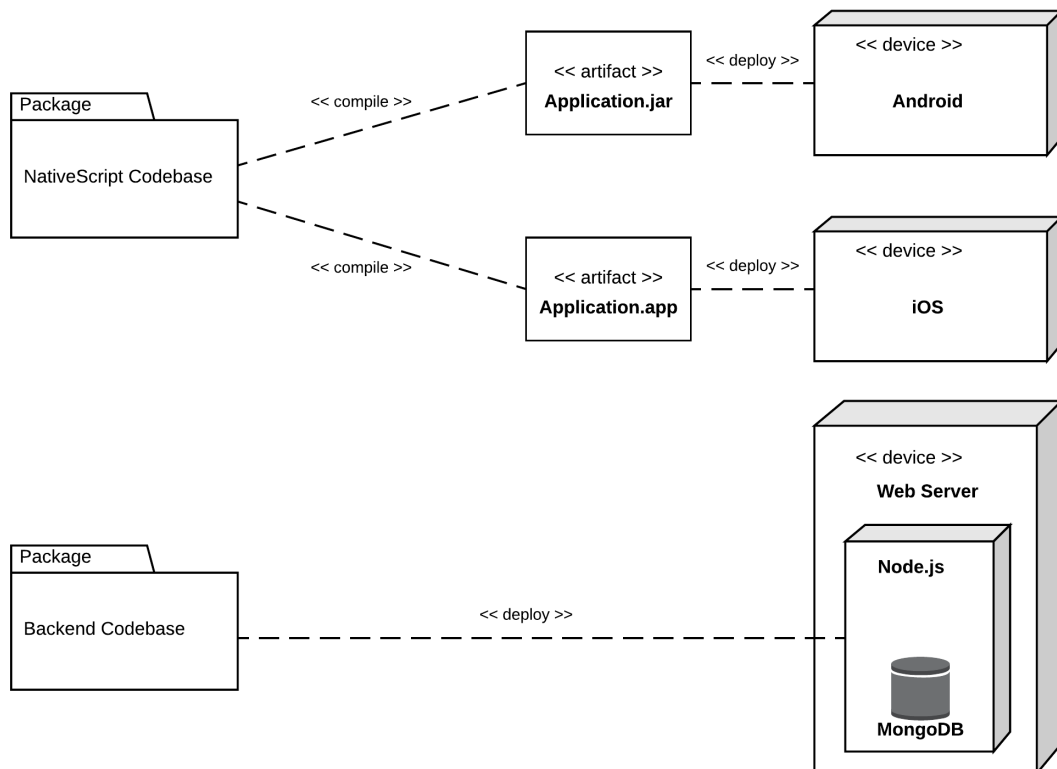


Figure 6.6. Deployment Model

6.6 Data persistence

This section describes how data persistence is obtained and also give insight into the database schema shown on figure 6.7.

To persist data this project uses *MongoDB* which is a document database. When working with document databases you often get a more flat and embedded data model than when working with relational databases. This is also reflected in our database schema on figure 6.7 which only has 2 entities which will translate into 2 collections in the *MongoDB*.

A Node.js environment is used on the server to communicate with a standalone MongoDB.

6.6.1 Database schema

After analyzing the domain and the requirements the following database schema was created to fulfill our needs.

It contains 2 collections, i.e. *Measurement* and *User*. *Users* represent all possible users of the system, both players, consultants and coaches. They are then distinguished with flags on the document (*isConsultant* and *isCoach*).

Users have a zero-to-many relation to itself because users represent every possible users i.e. friends, consultants and coaches.

Measurement represents both raw data as well as calculated data.

Measurements have a zero-to-many relationship to users because a user can have 0 measurements, but a measurement will always have at least 1 user and possible more if the measurement was uploaded by a coach or consultant.

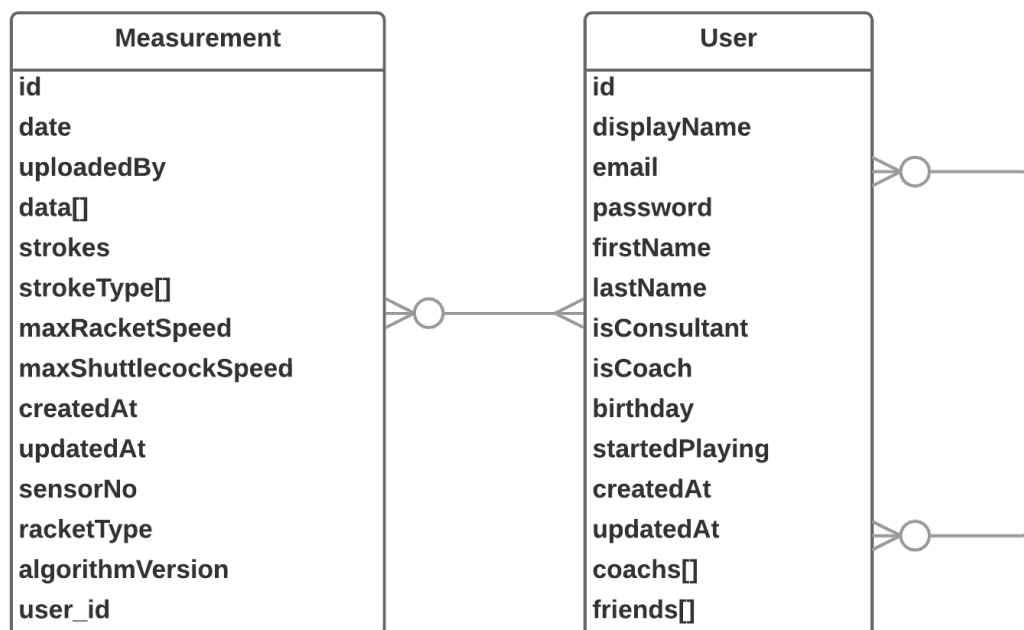


Figure 6.7. Database schema with *crow foot* notation

Implementation

7

This chapter describes some parts of the software implementation. For further details, see the documentation.

7.1 Algorithms

As mentioned in the introduction, section 1.3, the consultants deliver the data algorithms to analyse the measurements from the racket. These algorithms are written in Python and by this not directly available to the Node.js environment on the back-end and JavaScript run-time on the front-end.

On the back-end it is possible to execute Python scripts with Node.js if the Python compiler is installed on the server. It is, however, not possible on the native devices. As of this, the algorithms is ported to TypeScript to make it available on both the back-end and front-end.

The algorithms are not optimized or re-factored, but it still increases performance considerable when running on Node.js as compared to running the scripts in Python. Early measurements indicate a 10 times performance increase.

Security

8

This chapter describes the considerations and decisions taken to secure the back-end application.

8.1 Authentication

To make sure that malicious users can not abuse the back-end application, the decision to secure it through authentication was made. There was 2 options for how this could be handled.

8.1.1 The common approach

This approach is to add a token of sorts to the header of the Hyper Text Transport Protocol (HTTP) request. The downside of this option when working with GraphQL is that you can not add a HTTP header to your request in GraphiQL when you query from the browser, unless a browser extension for that specific thing is used. It would also not be very transparent or obvious when browsing GraphiQL that an authentication token is required for certain resources.

8.1.2 The Facebook approach

Facebook uses a root type called a viewer (Hurrell, Greg, 2015). The **viewer** represent the user making the requests. This can be used for several things. What we did was add a parameter on the root type **viewer** so that the **viewer** takes a token. This approach makes it very obvious that the **viewer** 'area' is restricted and requires a authentication token to access. Every resource that requires an authenticated user will then be underneath the **viewer** type and every other endpoint are exposed outside the **viewer** object. On figure 8.1 this is illustrated.

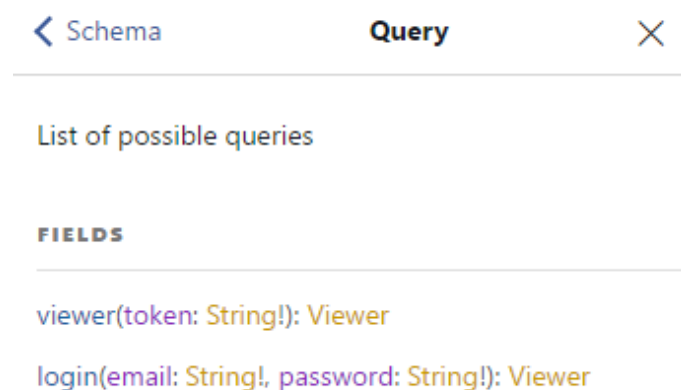


Figure 8.1. Query documentation showing the viewer and login

Notice that both of them returns a **viewer** type. When logging in, the same **viewer** as you can get when asking for the **viewer** with your token, is returned. This is to save a round-trip after logging. Another approach is to **login** and only receive a token which can then be use to query for the **viewer**. But that approach is a lot slower since it will require 2 round-trips.

The most important queries to get behind authentication is the mutations since they are the queries that manipulate data. So all the mutations are hidden behind a **MutationViewer**. This is shown on figure 8.2. When passing the token to the viewer, it is checked in the database and the user owning the token, if any, is then passed down the graph so that the queries below the viewer know exactly who the user is.

8.1.3 Token generation

The token is generated on login and removed on logout. An **npm** package named **node-uuid** is used to generate an RFC4122 v4 UUID, which is then stored with the user.

The odds of creating duplicate tokens when using RFC4122 v4 is extremely small. On average it would take a few trillions of UUIDs to generate 2 duplicate tokens (Wikipedia, 2016).

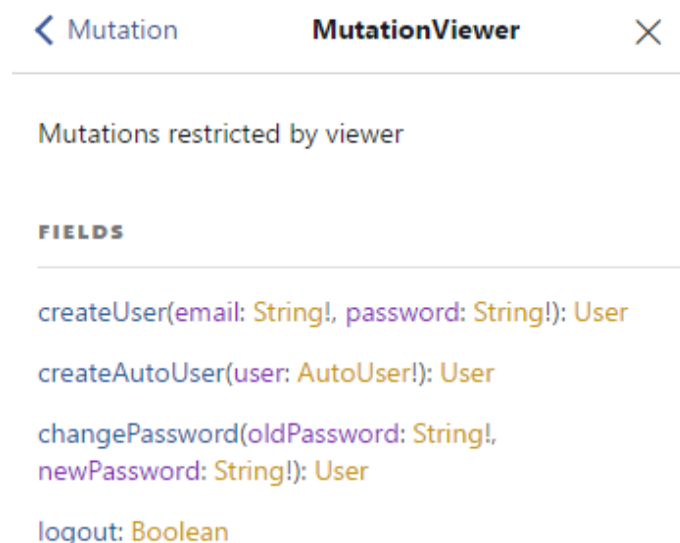


Figure 8.2. Mutation documentation showing what is behind the viewer

Testing

9

This chapter describes how testing is setup and managed across the applications

9.1 Front-end

Challenges were met when attempting to test the front-end application. With the Travis CI we met problems with testing using the NativeScript testing framework. The result of this is that we only test the business logic of the application and also tests that the application is able to build on the Travis CI.

The business logic of the application is tested with unit tests that isolate specific areas of the code.

The Travis CI runs the tests, build the application and checks the testing coverage of the code.

9.2 Back-end

The back-end application is tested on two levels. The first level is unit tests. These are run on isolated functions where dependencies are mocked away. The second level is integration tests. These test the application on every level, with nothing mocked away.

First level

Unit tests, test isolated parts of the application. They test functions at the different layers, where the below layer will be mocked away to make the tests isolated. This is the simplest form of tests but are still a valuable tool to create a durable and well tested application.

Second level

Integration tests work differently than the unit tests. They do not test the code itself, but the application as a whole. This means that these tests only go well, if no errors occur in any layer of the application. A top down approach is used. The integration tests start up the node server, and then starts querying against its endpoints. When tests are run, a test database is used. This test database is seeded with known data every time the server starts in testing mode.

Continuous Integration

When commits are made and pushed to the git repository. A Travis CI is setup to run different tasks and verify the integrity of the commit. Tasks that are run include linting, building, running tests and coverage. Should any of these tasks fail, the user that committed the code, will receive an email notification. This helps both the developer

who made the changed, but also the developer that have to review the changes. When you review a pull request, you get information about how the tests went and what the coverage is. If coverage went down in the pull request, compared to the master branch, then the developer probably didn't test his code well enough.

Back-end Analysis

10

This chapter gives insight into the performance of the back-end application.

10.1 Optics

Optics is an analysis tool build by the Meteor developer team for use on GraphQL servers. They currently support Express, Hapi, Koa and Ruby servers. It is a commercial product but also contains a free plan for developers.

Optics was added to the project to get insights into the performance of the different queries the user can perform. The data can then be used to pinpoint performance bottlenecks and where you need to optimize your application.

10.1.1 Performance analysis

Figure 10.1 is a screenshot from Optics that shows the analysis for querying after the **viewer**, including the **user** but excluding the users **measurements**.

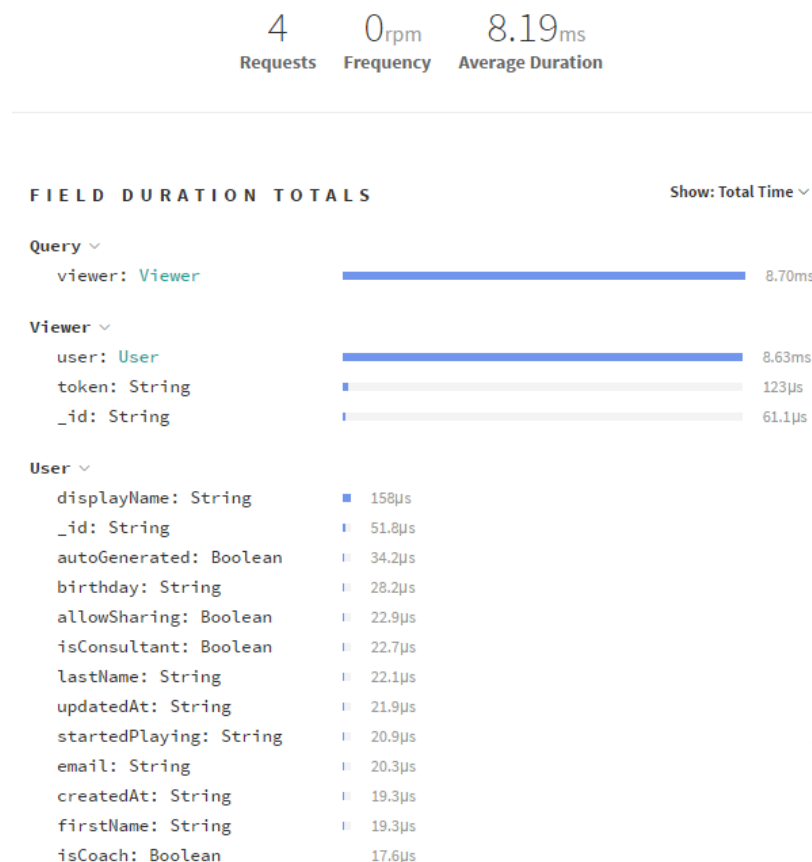


Figure 10.1. Optics analysis for viewer, including the user but excluding measurements

Figure 10.1 shows that the average duration for a `viewer` with a `user` only takes 8.19 millisecond to execute. That is pretty fast. But when asking for the `users measurements` the duration increases tenfolds, as illustrated on figure 10.2



Figure 10.2. Optics analysis for viewer, including the user and measurements

With `measurements`, the query takes on average 147 milliseconds to resolve. Those statistics are on a `user` with only a few measurements. From this we can easily see that when `measurements` grow we need to change how it is handled. Adding "pagination" could be one way to solve it. When analyzing closer it becomes clear that the `data` field on `measurements` are where it hurts. When we query without it we get the result on figure 10.3

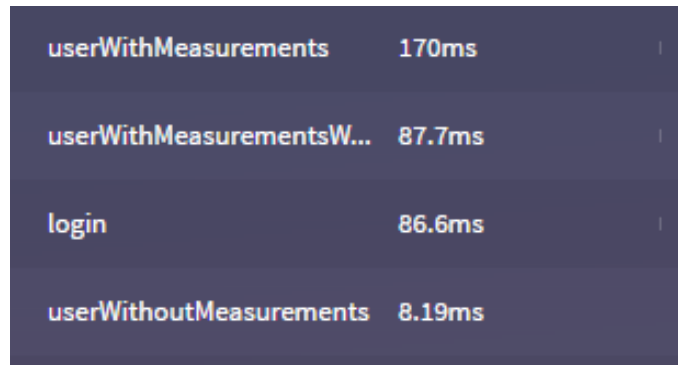
Without the `data` field on `measurements` the query was resolved in half the time. This shows the power of GraphQL and proves that asking for data you don't need can be a big performance penalty.



Figure 10.3. Optics analysis for viewer, including the user and measurements but excluding data

10.1.2 Comparison

When comparing a few queries side by side it becomes very clear that the amount of data that is queried have a big impact on performance. Why login is slower than querying after a user can be tracked down to how the database is setup. Users are only indexed by id. So when login tries to find a user matching a specific email and password, it takes a lot longer than when looking for a user with a specific id. By adding indexes to the email, we could see a performance increase.



userWithMeasurements	170ms	
userWithMeasurementsW...	87.7ms	
login	86.6ms	
userWithoutMeasurements	8.19ms	

Figure 10.4. Comparison of 4 different queries

Acceptance tests

11

This chapter will contain user stories from chapter 4 followed by acceptance tests to test whether the requirements are fulfilled or not.

As a consultant / user, I want to use a phone or tablet to access the system

Device operating systems to support includes:

Apple iOS 9 and 10

Android 4.3, 4.4, 5.0, 5.1, 6.0 and 7.0

To limit the scope of the project, only Android version 5.0 is tested on hardware and version 6.0 on an Android emulator.

	Yes	No
As a consultant / user, I can download the app from iOS app store using a phone or tablet with either one of the supporting operation systems		
As a consultant / user, I can download the app from Android app store using a phone or tablet with either one of the supporting operation systems		
As a consultant / user I can log in and access the system in the downloaded application on iOS		
As a consultant / user I can log in and access the system in the downloaded application on Android		

As a consultant, I want to input user information when beginning a session and automatically add them to the user database

If internet is present, the user information must be submitted immediately to the database.

If internet is not present, the user information must be stored on the device for later automatic submission.

	Yes	No
As a consultant, I can input user information on their behalf		
If internet is present, the user will be added to the database immediately		
If internet is unavailable, the user will be stored on the device and added to the database when internet connection is restored		

As a consultant, I want to start a session with a user and record data from the racket

	Yes	No
As a consultant, I can start a session with a user and record data from the racket		

As a consultant, I want started sessions to be persisted between application restarts

	Yes	No
As a consultant, I can restart the application and still have the same started sessions as before the restart		

As a consultant, I can find a previously stored user

	Yes	No
As a consultant, I can search for and find a previously stored user		

As a consultant, I want to show details of recorded data to specific users

	Yes	No
As a consultant, I can show details of recorded data		

As a user, I want to receive an email with a generated password for my account, when a consultant have created an account on my behalf

	Yes	No
As a user I receive an email with a generated password to my account, when a consultant have created one on my behalf		

As a user, I want to be able to change my password from within the application

If internet is not present, this operation is not possible.

	Yes	No
As a user, I can change my password from within the application, if i have internet access		

As a user, I want to compare my performance data to other users

	Yes	No
As a user I can compare my performance data with other user		

As a user, I want to share my user profile on social media

	Yes	No
As a user I can share my user profile on Facebook		
As a user I can share my user profile on Twitter		

As a user, I want to see my recorded data compared to historical data

	Yes	No
As a user, I can see my recorded data compared to historical data if i have multiple recording sessions		

Racket description

12

This chapter will describe the available features of the racket and explain some basic terms that are relevant for the descriptions. The chapter is mainly a short introduction to the terms based on the document in appendix D.

12.1 Terms

There are four terms generally used to describe badminton rackets.

Point of balance

Generally this is a measurement in millimetres from the bottom of the handle and up the shaft. Rackets are grouped into three categories according to this length. There is no official definition of the limits of these, but the general understanding is as shown in table 12.1.

Weight

The weight of the racket is normally in the range of 75-90 grams without strings and grip and 80-100 grams including strings and grip. Generally the Yonex *U-system* is used to group rackets as seen in table 12.2

Moment of inertia

This is a measure of the racket's ability to rotate. Different rackets require different energy amounts to rotate and typically this is in the range of 85-112 $kg \cdot cm^2$. This is a key measure to calculating the features of the racket.

Category	Length [mm]
Head light	< 285
Balanced	285-295
Head heavy	>295

Table 12.1. Categorization of a racket's point of balance

Category	Weight [grams]	Description
U1	95-100	Very heavy
U2	90-94	Heavy
U3	85-89	Normal
U4	80-84	Light
U5	75-79	Very light

Table 12.2. Categorization of a racket's weight

Flexibility

As rackets are constructed of different materials they differ in their ability to bend. This is the measure of flexibility. According to the player's style more or less flexibility can be preferred.

12.2 Sensors

The sensors build into the racket are an accelerometer and a gyroscope measuring on all three axes as illustrated on figure 12.1.

Each sensor has a conversion factor to translate the measured levels into SI units. The factors from the sensor producers are, per the consultants, not accurate to the measurements made with the racket. Because of this, it is important to keep track of what sensors recorded the data to be able to recalculate features when new versions of the racket might ship.

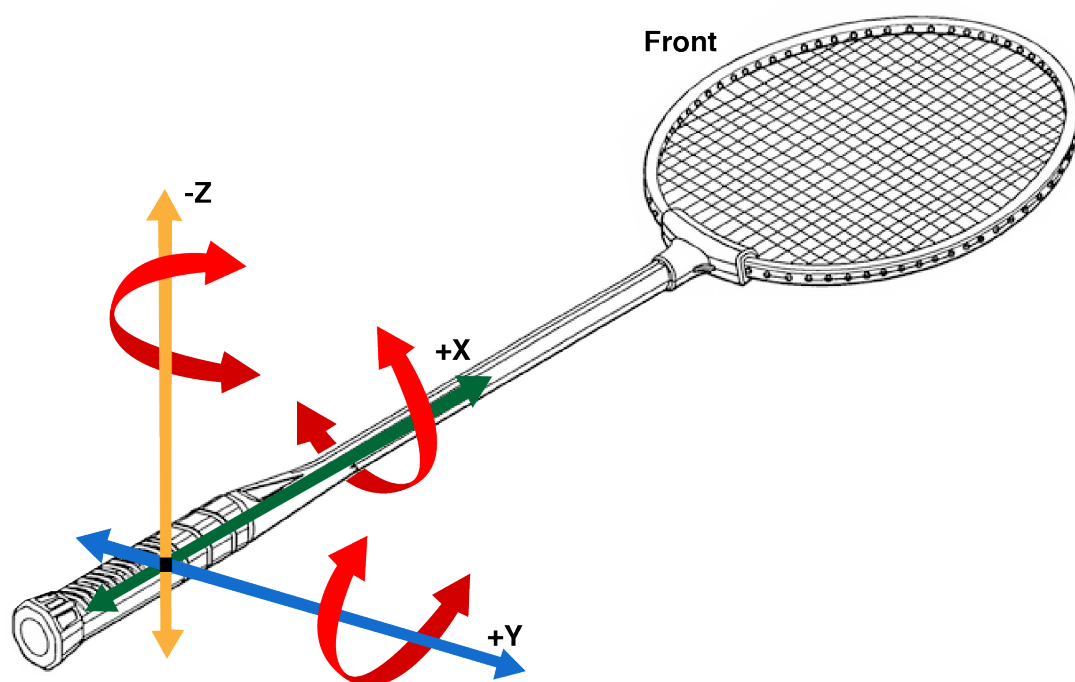


Figure 12.1. Axes of measurement for the racket sensors

12.3 Features

Several features are possible to identify from the collected data. Some requires only the data it self and others correlate to existing data, e.g. the users own data.

Kinematics

This is a feature that analyses the kinematics during a stroke. Four measures are available from the sensors.

- Angle acceleration
- Angle speed
- Linear acceleration
- Linear speed

Each of these can give a measure independently or be combined as a measure of the athletes technique as described below.

Theoretically all the measures combined can describe the racket's position in space. This is however not yet possible, as the racket construction introduce some drift to the data.

Power

The measure of power is really a calculation of the theoretical initial speed of the shuttlecock.

Technique

The kinematic measures can be used to calculate the stroke details and can describe how "good", "accurate" or "hard" a given stroke was. By correlation analysis with existing records of performances these measures can be used to rate strokes against fellow athletes or *a golden standard*.

Choice of racket

To help the athlete select the perfect racket, a combination of the above mentioned measures can be used to select the better racket type for the individual. Athletes on different levels have different demands of their racket and by analyzing the kinematics and power of a stroke with different rackets, the optimum racket can be found.

In-game time

This is a measure of the effective game time of the athlete, i.e. excluding breaks and time outs.

Bibliography

Android. Dashboards. <https://developer.android.com/about/dashboards/index.html#Platform>, 2016. [16.08.2016].

Angular. Architecture Overview. <https://angular.io/docs/ts/latest/guide/architecture.html>, 2016. [25.10.2016].

Apollo Stack. angular2-apollo. <https://github.com/apollostack/angular2-apollo>, 2016. [06.09.2016].

Apple, Inc. App Store Support. <https://developer.apple.com/support/app-store/>, 2016. [16.08.2016].

CA Technologies. Frequently Asked Questions. <https://www.zenhub.com/support>, 2015. [25.06.2016].

Zhi Dian. Holy Pie Smart Racket: For Badminton and Tennis. <https://www.kickstarter.com/projects/107521314/holy-pie-smart-racket-for-badminton-and-tennis/description>, 2014. [09.06.2016].

Liu Hongzuo. SOTX's badminton tracking Smart Racquet now available in Singapore. <http://www.hardwarezone.com.sg/tech-news-sotx-s-badminton-tracking-smart-racquet-now-available-singapore>, 2016. [09.06.2016].

Hurrell, Greg. Unofficial Relay FAQ. <https://gist.github.com/wincent/598fa75e22bd44cf47#how-does-graphql-know-what-type-of-data-is-being-requested-by-a>, 2015. [22.11.2016].

Eva Johnson. Why is Scrum so Popular? Why is Scrum So Successful? <https://intland.com/blog/agile/scrum/why-is-scrum-so-popular-why-is-scrum-so-successful/>, 2015. [11.09.2016].

NativeScript. Groceries. <https://github.com/NativeScript/sample-Groceries>, 2016. [06.09.2016].

Net Applications. Mobile/Tablet Operating System Market Share. <https://www.netmarketshare.com/operating-system-market-share.aspx?qprid=8&qpcustomd=1&qpdp=2016&qpnp=1&qptimeframe=Y&qpct=3>, 2016. [27.09.2016].

Shenzhen Earth Electronics, Ltd. Holy Pi. <http://www.szearth.com/en/index.html>, 2016. [09.06.2016].

SOTX Sports Equipment Company, Ltd. SOTX Smart Badminton Racquet. <http://www.sotxsports.com/en/html/news/list-257.html>, 2016. [09.06.2016].

Valio Stoychev. NativeScript 2.0 - the best way to build cross-platform native mobile apps. <https://www.nativescript.org/blog/details/nativescript-2.0---the-best-way-to-build-cross-platform-native-mobile-apps>, 2016. [29.06.2016].

Voices of Spring. USENSE. <http://goo.gl/vvoBov>, 2016. [09.06.2016].

Wikipedia. Universally unique identifier. https://en.wikipedia.org/wiki/Universally_unique_identifier#Version_4_.28random.29, 2016. [22.11.2016].

Perry Wong. "Smart Racquet" advance technology in badminton sport. <https://www.linkedin.com/pulse/smart-racquet-advance-technology-badminton-sport-perry-wong-1>, 2016. [09.06.2016].

Viveca Woods and Rob van der Meulen. Gartner says worldwide wearable devices sales to grow 18.4 percent in 2016. <http://www.gartner.com/newsroom/id/3198018>, 2016. [07.07.2016].

Consultant interview



This is a transcription of a semi structured interview with two representative consultants. The interview was conducted in danish as both consultants are danish citizens.

Interviewer:

Bjørn Sørensen

Jesper O. Christensen

Interviewees:

Mark Boesen, marcboesen@hotmail.com

Kaare Moss, kaaremoss@gmail.com

Date: 19:45 - 22:00 03/08/2016

Location: Aarhus, Denmark

A.1 Hvordan vil et typisk salgsforløb til potentielle kunder forløbe?

Konsulenten indsamler lidt metadata om en bruger, som navn, alder, dato, erfaring, ketchertype, mv., inden der spilles i en kort periode med systemet. Derefter præsenteres den optagede data på en PC. Der bør være en umiddelbar spiselig grafisk præsentation med nogle pop-termer, som hvor mange slag, typer og grafer. Hvis det er interessant for brugeren, så kan der skiftes til nogle mere detaljerede oplysninger.

Efterfølgende vil det være interessant at kunne oplyse brugeren om hvilke muligheder og perspektiver der er for at forbedre sin træning med træningsprogrammer og lignende.

Opfølgning på forløbet kan ske i form af nyhedsbreve og direkte og indirekte henvendelser til klubber og spillere. Evt. ved fremsendelse af målte data på e-mail og papir.

A.2 Hvilke værktøjer er nødvendige i dette forløb?

En PC, tablet eller lignende hvor der er muligt hurtigt og nemt at præsentere data samt indtaste brugeroplysninger. Det bør virke uden internetadgang, men skal senere kunne sende data ud til videre behandling.

A.3 Hvad skal programmet kunne udføre?

Need to have

Præsentation af seneste optagede data, sammenholdt mod evt. historisk data. Helst en grafisk præsentation af historikken.

Fremsendelse af målt data (spillerprofil) på e-mail.

Top-lister og rangeringer mod venner og professionelle.

Skal kunne dele data på sociale medier. Gerne direkte fra resultat-e-mailen der er fremsendt.

Bruger oprettelse

Nice to have

Nyhedsbrevsfremsendelse ud fra en e-mailliste.

Brugere skal kunne indhente achievements når nogle fastsatte måltal er overgåede, f.eks. over 2000 slag.

Sammenligning med andre spillere, hvis de følger dem.

3D illustration af de enkelte slag med kvantitative detaljer undervejs i svingsløjfen.

Træningsplaner til træner for specielle scenarier.

Push-beskeder der kan notificere brugeren om seneste hændelser.

Trænerprofil med overblik over sine spillere og deres fremgang.

Klubsystem med mange administrative funktioner. Detaljer kan høres i interviewet fra omkring 48:00. ¹

¹FiXme Fatal: Add directions to get interview here eventually

Android version distribution B

The following statistics are replicated as reference based on Android (2016).

The distribution of *Android* operating system versions as of November 7, 2016.

Version	Name	Distribution
2.2	Froyo	0.1%
2.3.3 - 2.3.7	Gingerbread	1.3%
4.0.3 - 4.0.4	Ice Cream Sandwich	1.3%
4.1.x	Jelly Bean	4.9%
4.2.x	Jelly Bean	6.8%
4.3	Jelly Bean	2.0%
4.4	KitKat	25.2%
5.0	Lollipop	11.3%
5.1	Lollipop	22.8%
6.0	Marshmallow	24.0%
7.0	Nougat	0.3%

Apple iOS version distribution C

The following statistics are replicated as reference based on Apple, Inc. (2016).

The distribution of *Apple iOS* operating system versions as of October 25, 2016.

Version	Distribution
10	60%
9	32%
Earlier	8%

Racket and sensors

D

This is a description written by the consultants.

Nyttige begreber

Badmintonketsjere, til voksne, har nogle forholdsvist enkle målbare parametre, som hver menes at have indflydelse på badmintonspilleres spil og spillestil.

Balancepunkt

Ketsjere med samme vægt kan have forskelligt balancepunkt. Balancepunktet er målt i millimeter fra bunden af håndtaget og op på skaftet. Der er tre klassifikationer på, hvorledes ketsjere inddeles efter balancepunkt; hovedlet, balanceret og hovedtung. Der er ikke nogen præcis konsensus om, hvornår ketsjere klassificeres som det ene eller det andet, men inddelingerne betragtes normalt vis til at være <285 mm, 285-295 mm og >295 mm.

Vægt

Vægten af badmintonketsjere, uden strenge og greb, varierer almindeligt vis mellem 75-90 gram og 80-100 gram med strenge og greb. Ketsjere klassificeres ofte efter Yonex's U-system som er givet i tabellen her under.

U1	95-100 gram	Meget tung	
U2	90-94 gram	Tung	
U3	85-89 gram	Almindelig/Normal	
U4	80-84 gram	Let	
U5	75-79 gram	Meget Let	

Inertimoment

Inertimomentet skal intuitivt forstås som ketsjerens træghed mod rotation. For at bestemme inertimomentet skal en ketsjers vægtfordeling og omdrejningsakse kendes. Ketsjere med samme vægt men forskellige vægtfordeling, som f.eks. hovedtung, balanceret, og hovedlet, vil have forskellige inertimomenter. For at svinge en ketsjer med et højt inertimoment skal der, teoretisk set, bruges mere energi, end ved en ketsjer med et lavere inertimoment ved samme rotationshastighed. Inertimomentet ligger normalt vis inden for $85 - 112 \text{ kg} \cdot \text{cm}^2$.

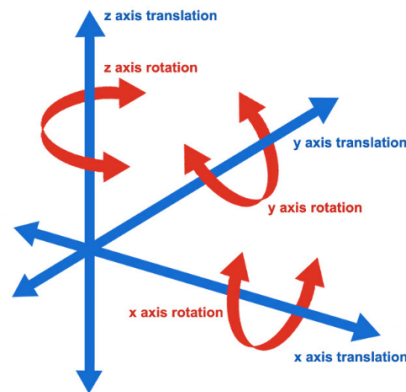
Fleksibilitet

Fleksibilitet giver en indikation af ketsjerens modstand imod deformation (bøjning). I modsætning til vægt og balancepunkt findes der ikke umiddelbart ikke nogen SI-enhed, som beskriver ketsjers fleksibilitet, hvilket gør det vanskeligt at sammenligne ketsjers fleksibilitet med hinanden. En ketsjers fleksibilitet afhænger af det anvendte materiale og måden producenten fremstiller ketsjeren. Ketsjere fremstilles enten i stål, aluminium eller kulfiber, afhængig af pris og kvalitet.

I et studie fandt man, at ketsjeres fleksibilitet og den elastiske deformation af skaftet i et smashslag bidrager med 4-6 % af fjerboldens udgangshastighed. I studiet bemærkede man, at fleksible ketsjere ikke nødvendigvis er at foretrække, da der som følge af øget fleksibilitet er en højere varians mellem slagene, end ved stivere ketsjere.

Lidt om bevægelsessensoren

Bevægelsessensoren (IMU'en) (ROM'en) består af en accelerometer- og gyroskopenhed, som registrerer translation i x, y og z retning og rotation omkring x, y og z akserne.



Accelerometeret kan har en målvidde på $\pm 200 \text{ g}$ og gyroskopet har en målvidde på $\pm 6000 \frac{\text{grad}}{\text{s}}$. ROM'en har en samplingsfrekvens på 500Hz og er en 10-bits enhede – dvs. $2^{10} = 1024$ svarmuligheder, hvilket vises i form af heltal mellem 0 og 1023. hvor ADC-niveau 511 svarer til 0 grad/s, ADC-niveau 0 svarer til -6000 grader/s og 1023 svarer til +6000 grader/s. Det samme er gældende for accelerometeret (bare -200 og +200). For at finde ud af hvor meget hvert ADC-niveau (Analog/Digital Conversion) er, kræver dette en omregningsfaktor. I starten beregnede vi omregningsfaktorerne på følgende måde:

$$\text{gyro: } \frac{12000}{1024} = 11,7 \frac{\text{grader}}{\text{s}}$$

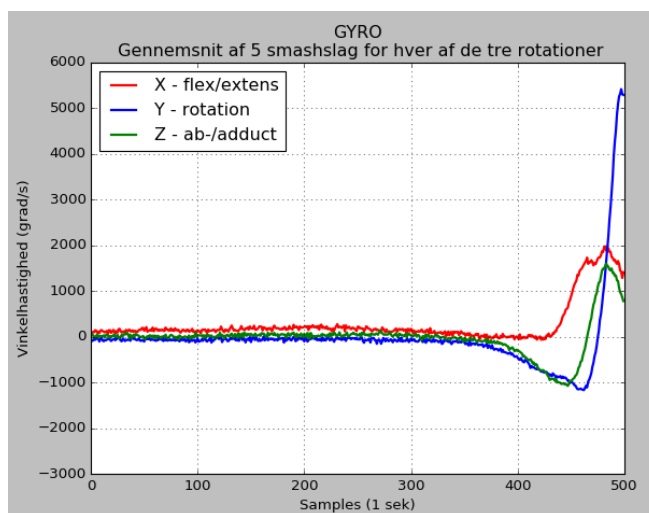
$$\text{accel: } \frac{400}{1024} = 0,39 \frac{\text{m}}{\text{s}^2}$$

Efter vores valideringsforsøg fandt vi dog ud af, at gyroskopets omregningsfaktor var ca. $18,5 \frac{\text{grad}}{\text{s}}$, hvilket betyder at det rent faktisk kan måle $\pm 9472 \frac{\text{grader}}{\text{s}}$ (producenten lover dog kun linearitet inden for $\pm 6000 \frac{\text{grader}}{\text{s}}$). Vi har endnu ikke valideret accelerometeret, men det er helt klar forventeligt, at omregningsfaktoren bliver en anden (højere) end 0,39.

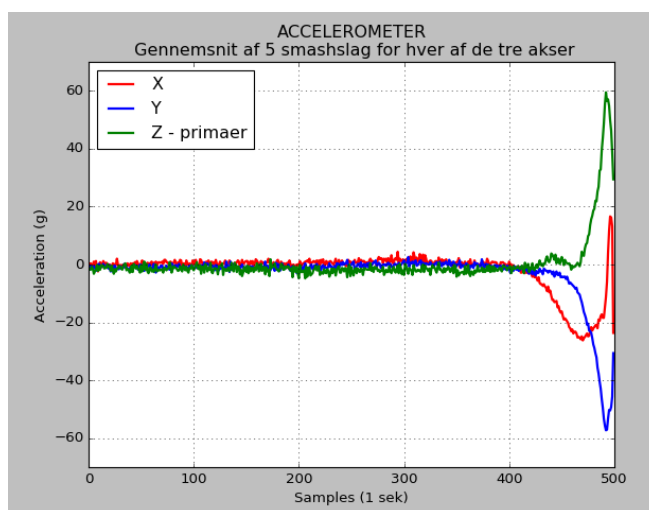
Det vi kan læse ud fra data

Nedeværende Figur 1 og 2 viser, hvordan data ser ud for hhv. gyroskopet og accelerometeret når rådata (ADC-niveauerne) er omregnet til de respektive SI-enheder, som er vinkelhastighed og acceleration.

Kurveforløbene på Figur 1 og 2 er gennemsnittet af 5 smashslag, som så er delt op i gyro data (rotation) og accelerometer data (translation) – havde det været andre typer slag, havde kurveforløbet set anderledes.



Figur 1 kurveforløb for rotation omkring X, Y og Z-akserne



Figur 2 kurveforløb for translation langs X, Y og Z-akserne

Nu hvor data er omregnet til brugbare SI-enheder, er det sådan set kun fantasien, som sætter grænser for, hvad man gerne vil vide eller finde ud af. Inddeles i overordnede kategorier, kunne det eksempelvis være interessant at se på 1) ketsjeføring, 2) power og 3) teknik, da disse giver en indikation af spillerniveau og træningsfremgang/tilbagegang. Yderligere vil man kunne se på indflydelse af 4) ketsjervalg, 5) effektiv spilletid og 6) andet spændende.

1) Ketsjeføring

Ketsjeføring er et begreb som giver en indikation af, hvordan svingsløjfen i forskellige typer af slag udføres. Her kan man kigge på:

- Vinkelacceleration ($\frac{grad}{s^2}$)
- Vinkelhastighed ($\frac{grad}{s}$)
- Lineær acceleration ($\frac{m}{s^2}$)
- Lineær hastighed ($\frac{m}{s}$)
- Position i rummet (når det ene gang bliver muligt at integrere data uden, at det driver)

2) Power

Hvis man kan finde ud af det, er det altid sjovt at slå hårdt. Derfor er fjerboldens hastighed en sjov men også nyttig spillerindikator.

- Fjerboldens (teoretiske) udgangshastighed.

3) Teknik

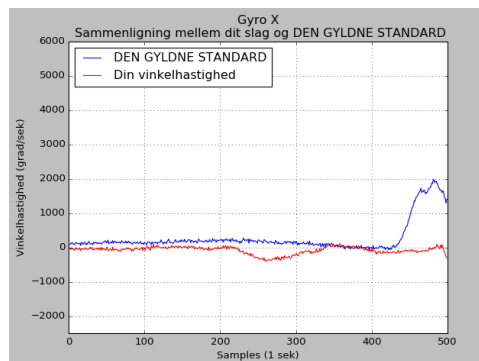
På baggrund af 1) ketsjeføring og 2) power i svingsløjfen er det muligt at lave en korrelationsanalyse (Figur 3 og 4) af sine slag, hvor man enten sammenligner med sig selv, sit sociale netværk eller med en "pro"/gylden standard, for at se på træningsfremgang/tilbagegang. Vil man sammenligne med sig selv eller sit sociale netværk, kræver dette en database med andres (tidligere) målinger. Vil man sammenlignes med en pro/gylden standard, kræver dette oprettelse af et slagkatalog, hvor alle relevante svingsløjfer for slagtyper findes, se Figur 5 og 6.

```
-----DIN SVINGSLØJFE ANALYSE-----
Antal fundne slag: 4
Gns vinkelhastighed (grad/s) 2083.7 , std 159.4
Gns vinkelacceleration (grad/s^2) 18431 , std 188
Gns impulsmoment ((kg*m^2)/s) 0.31 , std 0.02
Gns lineær hastighed (m/s) 18.2 , std 1.4
Gns lineær impuls (kg*m^2) 0.6 , std 0.6
Gns udgangshastighed (ms/s) 28.4 , std 2.1 , (km/t) 102.4
-----DIN GYLDNE STANDARD-----
Antal fundne slag: 5
Gns vinkelhastighed (grad/s) 5535.5 , std 223.6
Gns vinkelacceleration (grad/s^2) 110358 , std 6291
Gns impulsmoment ((kg*m^2)/s) 0.82 , std 0.03
Gns lineær hastighed (m/s) 48.5 , std 2.0
Gns lineær impuls (kg*m^2) 1.6 , std 1.6
Gns udgangshastighed (ms/s) 73.3 , std 2.9 , (km/t) 263.7
```

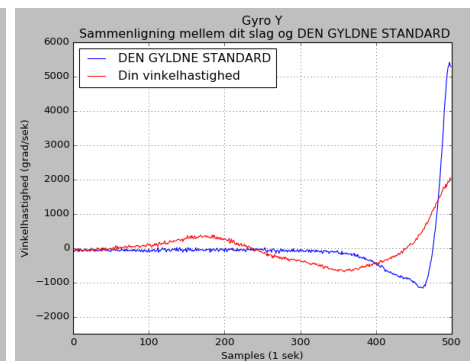
Figur 3 forskelligt data for GYLDEN STARDARD og EGEN ANALYSE.

```
-----KORRELATIONS ANALYSE-----
X [[ 0.03252902]
Y [[ 0.64023066]
Z [[ 0.23232469]
```

Figur 4 korrelation mellem GYLDEN STADARD og EGEN ANALYSE – jo tættere på 1 desto bedre.



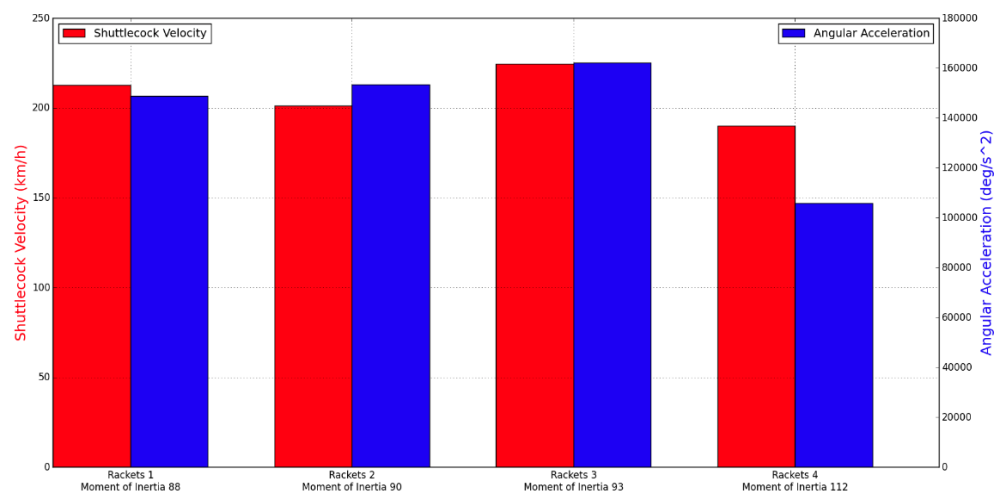
Figur 5 kurveforløb for rotation om X-aksen



Figur 6 kurveforløb for rotation om Y-aksen

4) Optimeret ketsjervalg

Afhængig af hvilken spillertype man er og spillespil man har, kan valget mellem 1) ketsjeføring og 2) power være afgørende for sin træningsplanlægning, udvikling og målsætning. Måske er man heldig at finde en ketsjer hvor *både* ketsjeføring og power er mest optimal (højest værdier) – ligesom Racket/Ketsjer 3 på nedeværende Figur 7.



Figur 7

5) Effektiv spilletid

- Giver vel sig selv?

6) Andet spændende

- Evt. andet? (kun fantasien sætter grænser)
-

To do

Udarbejdelse af et dynamisk script som på en sofistikeret måde kan definere og kende forskel på forskellige slagtyper ud fra én lang datastreng, så slagene ikke behøver at optages/trænes isoleret.