Saxion University of Applied Sciences

Graduation report

Open source “Load monitor” application

Student:
Dovydas Valiulis
436254@student.saxion.nl
HBO-IT SE-track

Company supervisor:
Etto Salomons
e.l.salomons@saxion.nl
Senior lecturer

Graduation supervisor:
Fleur Oudenampsen
f.oudenampsen@saxion.nl
Docent

2021-06-13

Table of Contents

1. Abstract

Over the last couple of years, an application that helps football club trainers monitor the performance of the athletes and predict optimal training difficulty was being developed. The main goal of this project was to combine all of the progress that was done by previous teams, add missing functionality, and make it open source so that other organizations could build on it further.

Over the period of this project, a lot of time was spent on refactoring and combining the code of different teams that have worked on it, mostly changing REST API to adhere to the best web development practices.

Another big part of the project was developing an import system where club trainers can import athlete's training data. Two prototypes were created and their performance compared in five different scenarios that should represent real-life scenarios. Based on the performance and other features a single prototype was selected. Later on, data that is imported is used for training machine learning models and predicting the difficulty of an athlete's training to reach optimal performance.

The basic functionality of this application could be used in the professional environment already. Nevertheless, the expectation for this application is to be further improved on by other developers to give even more insight into athlete's performance.

2. Definitions

API - Application Programming Interface. Collection of methods that allow two systems to communicate with each other. Usually in the form of HTTP requests.
Algorithm - Sequence of instruction of how to accomplish the desired goal.
Application architecture - Describes the patterns and techniques used to design and build an application (Redhat, n.d.)
Augmented interactions - Methods that incorporate technology into everyday lives and extend the capabilities of the user.
Augmented reality - Area of augmented interactions that focuses on displaying additional objects/information on the user's device.
Back-end - Describes part of the software system that is not seen by the users. It includes processing logic, database access, and communication with other systems.
Big data - Describes large volumes of data and solutions for analyzing it.
Bottleneck - Part of the system that limits all other components when loaded.
Data mining - Process that gathers useful data to the business from the raw collected data (Investopedia, n.d.)
Database - Collection of data saved/used in a software system.
Developer - Person that designs/implements software systems
Embedded systems - A computer system that is functioning in a larger computer.mechanical system.
End-user - intended user of the application.
Front-end - Describes part of the software system that users see. Also known as graphical user interface
Functional documentation - Document that describes functions of the application.
GDPR - General Data Protection Regulation. EU regulation that protects user data. (EU law, 2019)
Jupyter notebooks - Integrated development environment (IDE) intended for Python development.
Machine learning - a data processing technique that creates models from existing data or environments.
Mixed reality - Area of augmented interactions that focuses on combining virtual reality and real-world to provide more information and/or interactiveness to the user
Node.js - Javascript runtime environment intended for creating back-end servers
Open-source - Application or code that is open to anyone to access, improve.
Scalability - Feature of the software system to increase processing capabilities and/or functionality.
Source code - application code that is written by the developer to implement desired functions of the system.
Technical documentation - Document that describes technical details of the application. It includes system architecture models, class descriptions, API endpoints, etc.
UI - User Interface, part of the application that the user interacts with.
Virtual reality - fully simulated experience of the environment.
Vue.js - Javascript framework used to develop UI for the web application.
Web application - Application that runs on the webserver and is reachable remotely.

3. Introduction

In this day and age, almost all industries encounter a problem where they have too much data to process traditionally. No exception is the sports industry. Data collected from athletes allows trainers and staff to accurately measure the performance and health of the athlete. We reached a point where we gather more data than we can process using traditional methods. So a relatively new method must be used to process an abundance of data. One of those methods is Machine learning. By letting machines analyze and create models of the data we can achieve much better results than by processing data in traditional methods.

This project strives to do just that. Provide trainers with additional knowledge by using machine learning. This helps trainers adjust training intensity and time to achieve the best performance out of the athletes.

4. Client

Ambient intelligence (AMI) is a research group that specializes in enabling IT for the smart world. AMI is a research group that works within Saxion University of Applied Sciences in the academy of creative technology (ACT) department. Ami is curated by professor Wouter Teeuw and three associate professors. Also, many of the researchers of AMI work as lecturers for Saxion. The main focus of the AMI is safety, sports, and the smart industry. It specializes in developing sensors for various applications, analyzing data from them, and providing software solutions based on that data. Based on these specializations there are three separate research lines at AMI: connected embedded systems, applied data science, and augmented interaction. (Ambient technologies, n.d.)

The connected embedded systems branch focuses on developing sensors for various uses. Furthermore, it works with sensor fusion, data distribution and filtering, communication protocols, and embedded software development. This wide field of activities allows AMI to provide products for many different areas. Projects in this research field include a range from energy sensors in microgrids to localization solutions for firefighters. (Ambient technologies, n.d.)

Applied data science's main focus is to process and create models out of the collected data. This field provides solutions for data storage and management, data mining, and machine learning model development. With these tools, they can provide intelligent solutions in many Big data fields. This project is part of this branch of research. (Ambient technologies, n.d.)

The main focus of the Augmented interaction branch is to incorporate technology into everyday life. It is focused on virtual technologies such as augmented, mixed, and virtual reality development and on physical representation like robots. These technologies have various application possibilities. One of the biggest achievements of this branch was developing a robot that can extinguish fires where it is too dangerous to enter for firefighters. (Ambient technologies, n.d.)

AMI collaborates with other universities, companies on various projects. AMI provides opportunities for students to collaborate on these projects. Some of the projects that multiple organizations have been collaborating on were:

- Mosis project - a collaboration with firefighters on creating a fireman tracking system inside the burning building.
- Load monitoring project - is a collaboration with some of the football clubs on creating a machine learning model for predicting various metrics of the players based on their performance during the training and other sources of data. One of the most used predictions is Rated Perceived Exertion (RPE).

5. Context

This assignment is a continuation of the “Big Data Technologies for Load Monitoring” project. During the last iteration of the project, the prototype was created for tracking and predicting an athlete’s load during physical training. To predict various metrics that are valuable for coaches, data is collected from sensors and questionnaires. According to that data, a machine learning model was developed to predict how the following training should be adjusted to reach optimal performance.

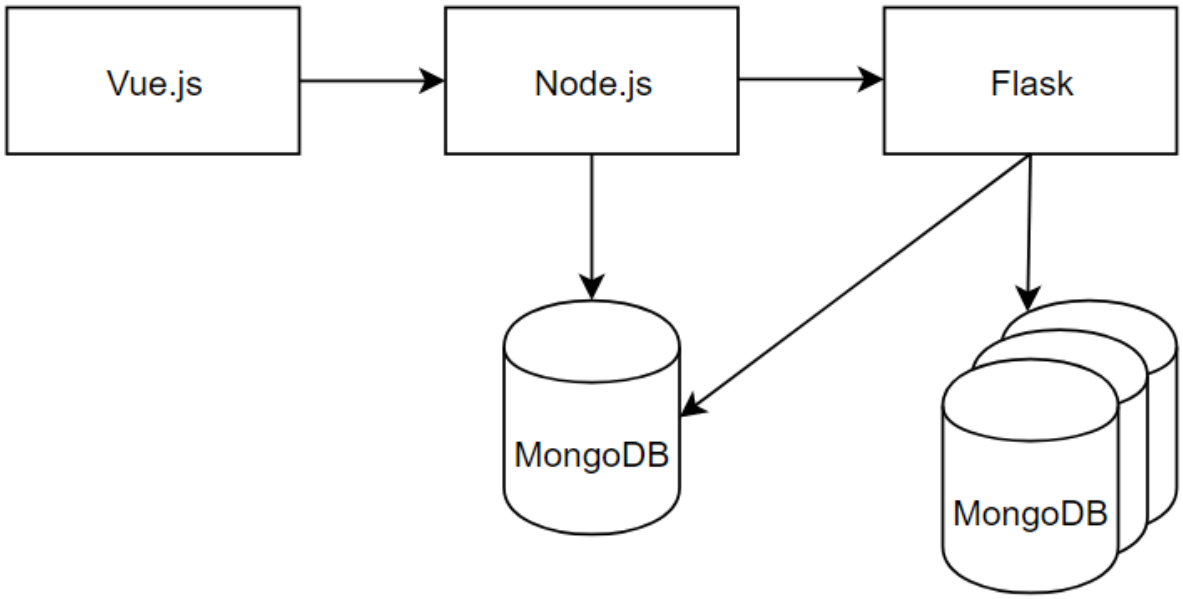
One of the main goals of this project is to make the whole system open source so that other organizations could use it freely, improve it, and help sports clubs get a better understanding of the data they are collecting. Unfortunately, some design mistakes were made during previous projects that prevent further development of this product.

5.1 Current system

The current version of the application is scattered among many different parts. The main parts of the application are:

1. Front-end (Vue.js) - Responsible for displaying information about clubs, teams, and players to the end-user.
2. Node.js server - Responsible for providing data to the front-end application and managing the majority of the entities in the system like clubs, teams, and trainers.
3. Flask server (Python) - Responsible for managing athlete’s training data, managing machine learning functionality, and mapping player data among databases.
4. MongoDB - Responsible for storing data. In this application, there are many database instances. There is one general-purpose database that is mostly used by the Node.js server. Also, many smaller databases are created for each club to store their athlete’s training data and machine learning predictions.

Figure 1
System model diagram of the original system



Besides all these parts there are some separate scripts for various functions that can not be executed by the end-user, like athlete's training data import that can only be imported by the developer.

Because this application is so scattered it is not possible to open source it. Therefore, it is desirable to have at least the main functionality working for the end-user. Also, have a more coherent and better-documented structure so it is easier to continue development for others.

6. Research

6.1 Research questions

How should data input/processing methods be designed to provide football organizations with easy, open-source, data import solutions using any format they like?

1. How should the application be refactored to improve stability, performance, security, and cohesion?
2. Which features are needed to collaborate with “Sport Data Valley”?
3. What features should be implemented to improve communication with the machine learning model?
4. What features and design changes are required for implementing data sharing functionality?

These sub-questions should provide a reasonable impact in answering the main research question. The first subquestion directly impacts open-source requirements as it is important that open source applications should be stable and clear for whoever continues the work.

The second subquestion is mostly concerned with integration with another company's product. It is also a good indicator if the software can be open-sourced if integration is successful.

The third subquestion is connected with implementing the missing functionality of the application that is one of the most crucial parts.

The final question describes additional functionality of this application that would provide more accurate data processing in a machine learning model by having the opportunity to gather more data.

6.2 Research phases

Due to different activities being required to answer all the research questions this project is divided into several different research and development phases. These phases provide a good amount of activities to answer research questions.

The first phase is Refactoring. Refactoring is the first phase because it allows us to explore the system and find troublesome parts. In this phase, the previous application was refactored to have a more coherent structure and to meet best programming practices. This phase answers the first and the third research sub-questions.

The second phase is developing/refactoring a generic import system that can be used by any football club that wants to. In this phase, research will be conducted on what import methods are available and how to implement them. This phase answers the main research question.

The third phase is research on data sharing solutions so that smaller clubs can contribute and collaborate to improve the system and use results to improve their training effectiveness. Also, it includes collaboration with the "Sport Data Valley". This phase answers the second and the fourth research sub-question

6.3 Refactoring

In this phase, several parts of the application were refactored to make it possible to open source it. With the previous solution, it was not possible to open source it because the structure of the project was scattered and several systems were connected to achieve a somewhat working product (more details in the following chapter). Also, several crucial parts weren't implemented at all into the final web application. For instance, the data import functionality was never implemented and only the placeholder component was created to show where the functionality was supposed to be. To import data to the system "Jupyter Notebook" was used. The main reason why this project is so scattered is that several teams have already worked on different parts of the application and probably did not communicate properly. Therefore each team has designed different parts of the application and used different technologies to implement them.

Table 1
Refactoring phase requirements (Sorted by priority)

1.	The system must be refactored to optimize resources used, simplify underlying architecture, and conform to the Open API standards
2.	Load the data in separate parts, only when the data is needed. Instead of one big request
3.	Update the REST API request to adhere to best web development practices
4.	Functionality that is already developed must be preserved.

In this phase, the main objective was to make a single backend server with all the functionality that was required for the front-end application.

6.3.1 Database layer refactoring

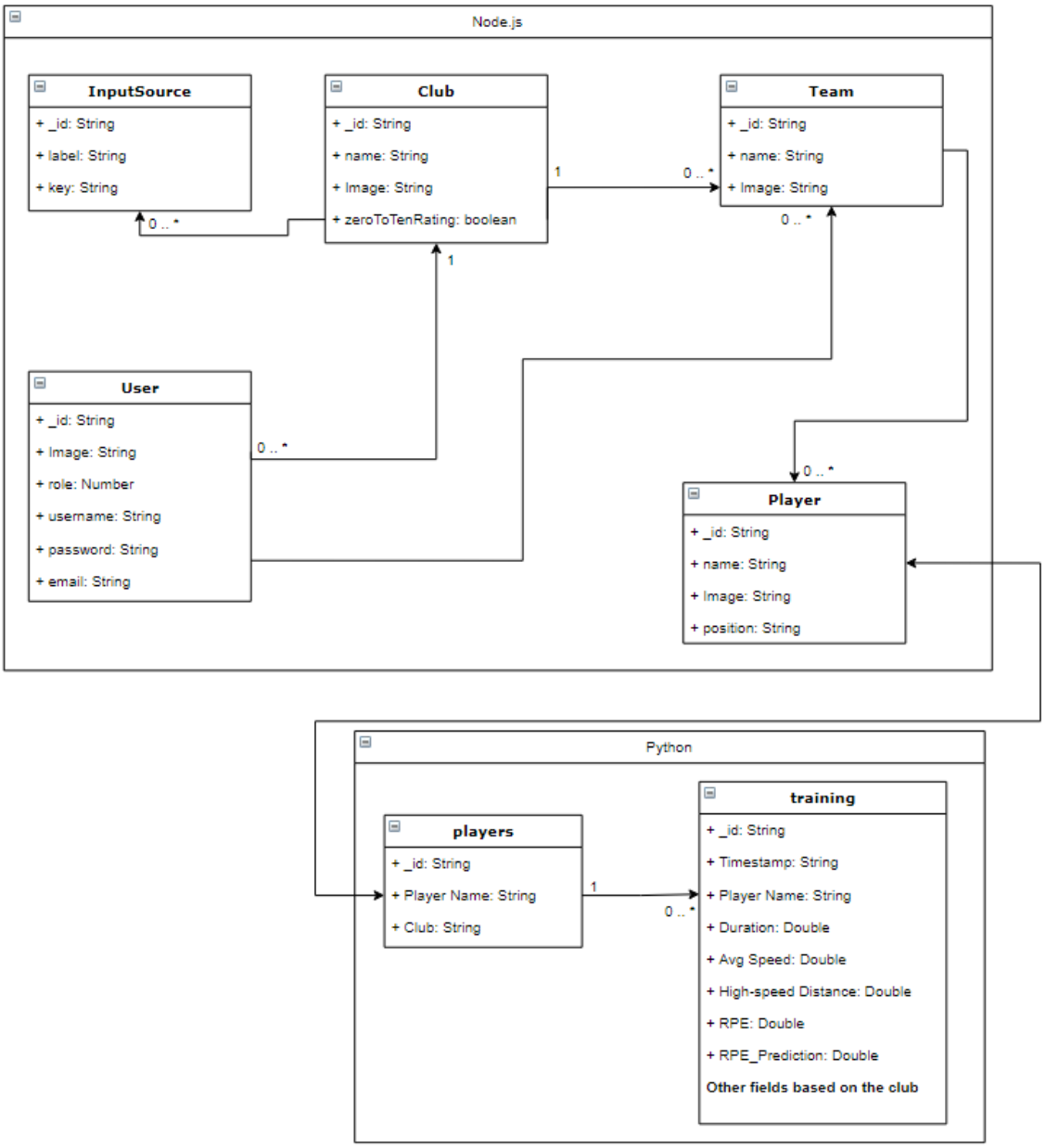
As it was previously mentioned, the whole application was scattered into separate services that have different responsibilities. One problem that occurs using multiple systems like that is that, at some point, entities have to be mapped from different systems.

Furthermore, with the previous version of the database model, some entities were described twice in separate databases and had different IDs but had almost the same information. For instance player representation in the system was being saved in multiple databases. In the first database general information about the player was saved and in the other database information about physical training performance and machine learning predictions. This makes managing these entities a lot more difficult because they are recognizable only by their name and not by immutable property like entity identifier (ID).

Finally, there was some inefficient way of connecting entities with each other. Mostly the name of the object was used to represent connection and not the id. This made having objects with the same name impossible. For instance, the system could not handle a scenario where two players have the same name. In the real world, this of course is possible and should be expected on the system level.

Figure 2

Class diagram of the original system

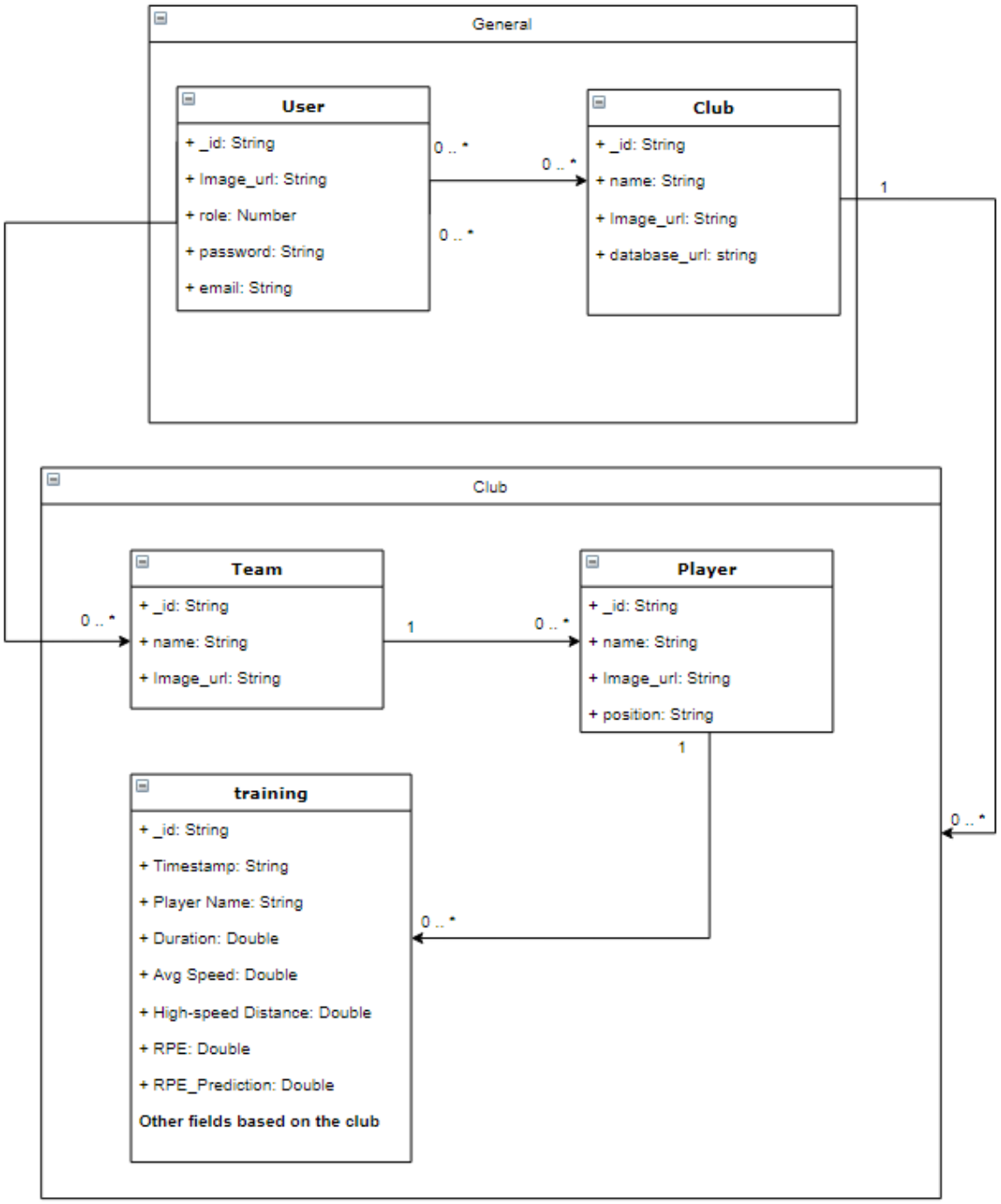


Therefore the first step in refactoring is to create a new database model that can provide all the possibilities of the previous systems and fix some of the issues from the previous architecture.

For this reason, two possible solutions have been created that solve some of the problems with the previous system.

Model 1

Figure 3
Class diagram of model 1



The first model is split into two parts. The first part is a general database that is used for all clubs. In this database, user information and basic club information are saved. Also, the “Club” entity has a connection that refers to the other database.

Another part of the model is a separate database where more detailed information about the club is stored. Each club has its own information stored in a separate database.

Advantages:

1. Club data is stored separately for each club. Therefore there is less chance that clubs could access sensitive data of another club.
2. Because each club database only contains records of a single club, the database itself is smaller in size and easier to manage.
3. Because the club database is separated from the general database, clubs can choose where they want to store data itself. So the club can store their data on an entirely separate machine from the general database.
4. It also provides better security because clubs can be responsible for their own data. Furthermore, it helps to comply with GDPR.
5. Each club can have different entity fields. Therefore separating club data helps to keep specific data of the club more organized and less probable that some of the unrelated data from other clubs will leak into the system.

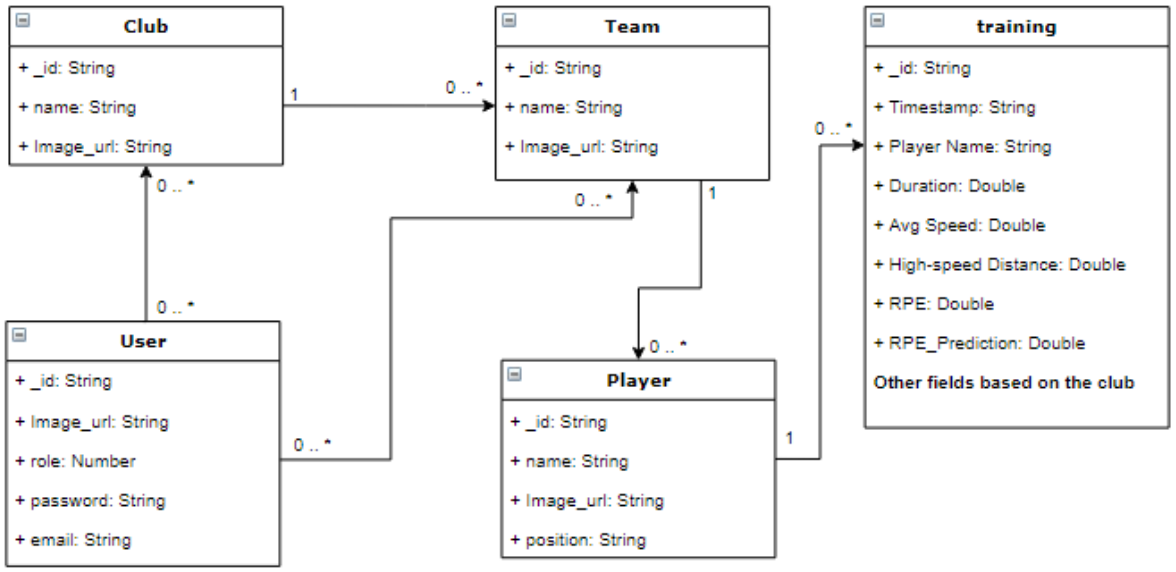
Disadvantages:

1. Due to the multiple databases that the system has, it might be difficult to manage all the connections.
2. Before each request that the user is making, the correct database needs to be resolved. This introduces some overhead on the back-end server itself to have a method to resolve these connections effectively.
3. Club databases need to be dynamically added or deleted when a new club is created or an old club is deleted. A new database has to be created because it is not known what clubs will be registering and data of each club should be stored in separate databases. The database is also deleted when the club itself is deleted to comply with GDPR and ensure the safety of sensitive data.

This model provides a couple of benefits over the previous solution. First of all, club data (other than basic information) is stored in a different database from the general information. Therefore each club can manage their own database and not be concerned that their data is accessible by the other clubs. Another advantage of this model is that there is no need of mapping the same entity with each other there for saving computation power, also it provides better mapping. It uses IDs to map entities with each other (the previous model was using entity names). Therefore this solution provides a better and more stable solution to the same problem.

Model 2

Figure 4
Class diagram of model 2



The second model which is considered had a single database to store all the information of the system. This provides an easier way to manage the database layer because the application only needs to manage a single connection to the single database. Some benefits that it has over the previous version of the application are: no mapping is needed between the same entities of the system, all mapping between entities are using IDs instead of the entity names.

Advantages:

1. The system requires only one database.
2. Easier database connection management
3. Does not need to dynamically add or delete databases.
4. Easier to manage entities in the program itself. For instance, ODM/ORM solutions could be used to provide easier entity management.

Disadvantages:

1. Club data does not have a strict separation from the model perspective.
2. Some club-specific fields can be mixed up and added to the wrong club.
3. In a case of a database breach, all data of all clubs can be accessed.
4. Clubs can be concerned that their data is saved in a general database.
5. Over time database size can increase too much and require an unreasonable amount of processing power.

Model comparison

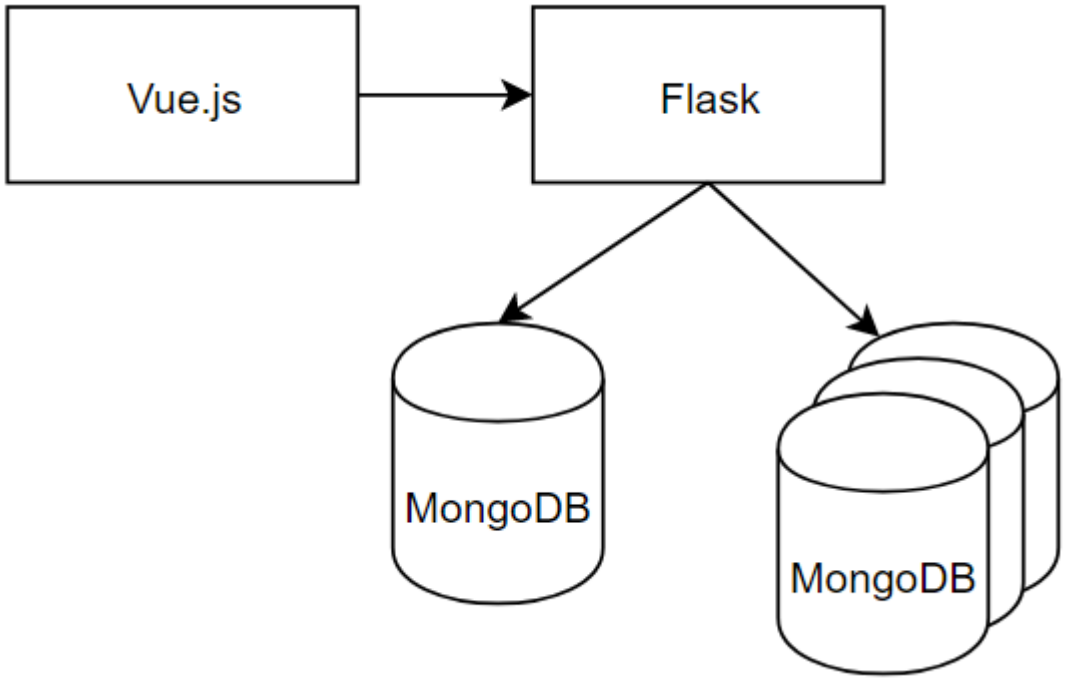
Overall both models are suitable candidates to replace the current implementation. To select the most applicable model of this system, advantages and disadvantages have to be evaluated.

After considering the advantages and disadvantages of each model, a decision was made to use the first model. The main reason for this choice is the separation of data that model 1 provides. But advantages provide better security and reassurance to clubs that their data will be safer than using another solution. Furthermore, it fulfills the requirements that are described in the first research subquestion.

New general system model

After implementing all of these changes the new system model looks like the figure below. It has fewer moving parts and is more consistent from the entity mapping point of view. This model does not require mapping algorithms for matching the same entities in the system (like the “player” entity). Everything is tracked in a single, coherent, system. Multiple databases remain, but data saved in them has changed. A single database is responsible for saving data about user logins and basic information about clubs (also club database connection address). Other databases are used to save specific information about the club like trainers, teams, and information about players. It also is responsible for saving specific physical training performance data that is imported and machine learning predictions.

Figure 5
System model diagram of the new system



6.3.2 API refactoring

As already discussed, the previous system was split up into several different parts. To make this application open-source, it had to be combined into a simpler version. Because one team was developing front-end applications and another was developing machine learning and import systems. They used different technologies but their purpose was basically the same; to analyze and provide data to the front-end application. The front-end team was using Vue.js and Node.js to implement their part and the machine learning team was using python framework Flask to implement API calls. Because both parts are responsible for the same thing, after discussions with the supervisor, it was decided to have a single codebase.

During this phase of the refactoring, Node.js and flask servers were combined into a single server using Flask. Flask was chosen because the machine learning part is written in python as well, therefore it is easier to have a majority of the source code in a single programming language.

Furthermore, some API calls needed to be updated because they did not adhere to the professional practices of web development. Some minor changes were made to the API call definition. For instance:

Table 2
REST API endpoint example changes

Old API URL	Old API method	New API URL	New API method
/api/club/add	POST	/api/club	POST
/api/club/delete	POST	/api/club	DELETE
/api/club/edit	POST	/api/club	PUT

There were a lot of similar changes to the API calls. The main problem with calls like that is that they don't adhere to best programming practices in web development and do not use proper HTTP request methods. (M-Way Solutions, 2018)

Another thing that was refactored was getting and updating the data of the front-end application. In the previous version, all data that was needed by the application was provided by one API call ("/api/loading"). It basically contained information about all clubs, trainers, and players with very little regard on which user sent the request. This approach had very large problems. First of all, every time an application updated any data all data was requested from the server. A small amount of data is not a big problem (from the performance standpoint) but with larger datasets the amount of data that is sent using this API call becomes infeasible. Also, this approach introduces another problem, all data filtering or sorting is being done on the front-end application which introduces larger loading times compared to handling all these actions on the back-end server or even forming specific queries that return data in the correct format. Also, it is a big security risk because data is not filtered on what the user can see. Due to all these issues, I have created API calls for each data entity that needs to be represented in the front-end and removed the single API call for

getting all the data. Overall twenty-two previous calls were changed or deleted and around fifteen new API calls were added to replace bigger calls.

6.3.3 Front-end application refactoring

Because most endpoints were changed in one way or another front-end application was refactored as well. The first data loading mechanism was changed, from loading data with a single API call to having multiple specific API calls for getting various forms of data required at the specific page of the application when the data is indeed required. Secondly, other API calls were changed to match with the refactored flask REST server.

Some additional functionality was added to the application itself to provide better security and more flexibility. Most changes were done to the new user creation and club administrator management. For instance, users in this application are created by the system administrators or club administrators. When a new user is created a password is displayed for the administrator that created the new user. This approach was refactored to the process that sends a password to the newly created users' email and the administrator has no information about what password was created for the user. The previous approach was very insecure because if the administrator account was leaked or hacked, by resetting the passwords of the users, the third party would have complete control over the whole system. And could easily reach sensitive club data. With the new approach, there is no way that the leaked system administrator user could access any data of the clubs.

6.3.4 Machine learning connection.

During previous iterations of the application development, the machine learning model and corresponding classes were developed. This machine learning model uses data that is imported into the system (described in the "6.4 Import system" chapter) and makes predictions on desired fields like RPE, average heart rate, sprint distance, etc.

Unfortunately, it was not connected to the front-end application and could only be used through separate scripts by the developers. Therefore it was needed to implement a basic user interface to give a user the ability to train new models and update predictions of the existing athlete's training.

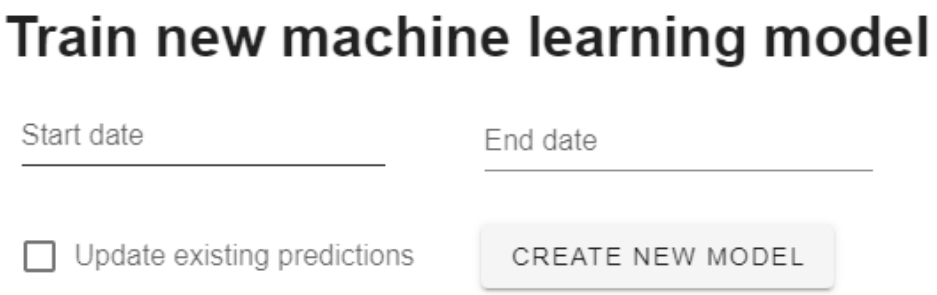
Table 3

Machine learning connection requirements (Sorted by priority)

1.	It must be able to create a machine learning model using GUI
2.	The user must be able to specify which period of time should be used to train the new model
3.	Users should be able to choose to update previous predictions based on the new model.
4.	Machine learning entry point should be configurable so that it is able to train models for more features.

In the new version of the application, the user has an option to create a new machine learning model based on the athlete's data. Also, users can specify which date interval should be used in creating a new model. Also, the user has an option to select if predictions based on the previous model should be updated automatically after a new model is created.

Figure 6
Machine learning model creation entry point in GUI.



This implementation directly corresponds to the third research subquestion. The new implementation provides the user a basic way of interacting with the machine learning module. This part of the system could be improved by adding additional ways of managing machine learning model creation.

6.3.5 Testing

For testing refactored features tests of previous teams were used to conduct verification tests. These test cases are described in the test report of one of the previous teams (Berendhaus et al., 2019). To verify that previous functionality is still working, regression testing using all test cases was executed. Overall most of the tests passed. Some tests did not pass due to changes made during this project to improve old features. For functionality that was refactored and no longer passes described test cases, new test cases were created. More details on test cases and results can be found in Appendix 11.1. In total 20 test cases were used to test previous functionality and all of them passed.

Because most of the requirements in the refactoring phase were concerned about architectural changes, functional testing is not an option. Unfortunately, due to time constraints, this part of the application was not thoroughly tested. The main indicator that the new design provides the same capabilities as the previous system and provides more functions were done in functional testing while executing the aforementioned test scenarios. Nevertheless, unit tests and integration testing would be a great way to ensure that the back-end system is working properly and find more ways where it can be improved.

For a completely new functionality like a machine learning connection, new test cases were created. Detailed descriptions and results can be found in Appendix 11.2. The main objective of these tests is to ensure that requirements have been fulfilled. Overall four new test cases were created to test machine learning functionality and all of them passed.

6.4 Import system

For an organization to use this application effectively, an import system for athlete's training data had to be developed. In the previous version of the application, the only way to import data was by using python notebook scripts. These scripts could only be run by the developers and were specialized for each organization. Therefore it was not a viable solution and was a major hurdle for making this project open source.

Table 4
Import system requirements (Sorted by priority)

1.	Import system must be configurable so that new organizations could import their data without changing the underlying logic of the system.
2.	The import system must be able to import at least excel (.xlsx) and CSV files. Other formats are optional.
3.	Import system must be able to map the same player's data from different files.
4.	Import system should have a way to map players from different files where their names have minor differences.

Data can be imported in two formats ".csv" and ".xlsx". Values in data sources can be one of these types: numeric, alphanumeric, boolean, and date text representation.

Data can be imported using a few different files or a single file. Few files are usually used when the source of data is different for each file. For instance, the club collects sensor data of each training and collects data from daily surveys about health and /or other details that could be useful in the machine learning model.

Therefore two new import prototypes were developed to compare them and find the best solution.

6.4.1 Methodology

To compare the performance of different prototypes file import time was observed. Also to get a better grasp of how the prototype performs, different scenarios were observed:

1. Scenario 1 - a single file import that maps to a single player in the database. The database table where imported files are stored is empty so there is no mapping among different training records. In the real world, it represents a scenario when a single player's training data is imported into a new system.
2. Scenario 2 - a single file that maps to a single player is used. The database table where imported files are stored is filled with available data so that the file would map to existing data. In the real world, it represents a scenario when a single player's training data is imported into an already usable system.

3. Scenario 3 - 10 files of different players are used. The database table where imported files are stored is empty so there is no mapping among different training records. In the real world, it represents a scenario when ten player training data is imported into a new system.
4. Scenario 4 - 10 files of different players are used, but only two of them are available for mapping with the specific player. The database is filled with required data for mapping two out of ten players. In the real world, it represents a scenario when some players already have pre-existing data and some players are newly added.
5. Scenario 5 - 10 files of different players are used. The database is full of available data for mapping and all ten players exist in an internal system. In the real world, it represents a scenario when ten player training data is imported into an already usable system.

In a single file test, a file of 58kb is used filled with preprocessed sensor data of the real player. In multiple file scenarios, 10 files of various players are used. Sizes of these files range from 18kb to 92kb. The total size of 10 files used was 556.5kB.

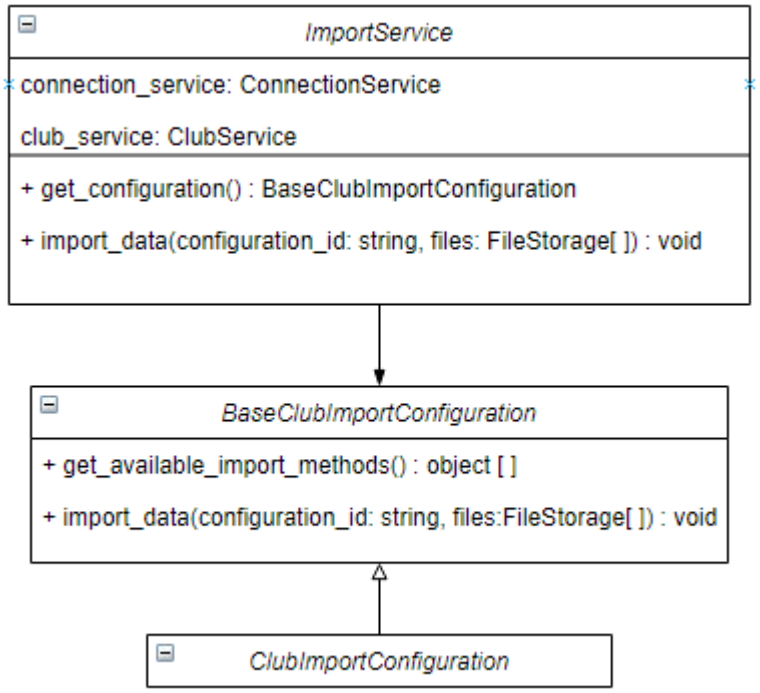
These scenarios provide an overview of how a system should be expected to run in the production environment.

Each scenario was run ten times to observe if import time is constant and does not depend on unforeseen factors. The same files were used in testing different models to ensure that we compare the same scenarios.

6.4.2 Import prototype 1

The first import prototype is a more static approach to the problem. Each organization needs a specified data reader to be implemented in the system. All readers have a configuration description of what files could be imported. This description is used in the user interface to provide users with the ability to import their data. Based on the configuration, the reader resolves what file is expected and how to read it. The file is first loaded into pandas dataframe (table-like data structure frequently used in data science) and then all necessary values are stored in a separate result dataframe. Then the mapping process begins. First records, if possible, are mapped with players. If it is unsuccessful then the record is stored in the unmapped training table and it will be mapped later on if a new player is created. Otherwise, it goes to the second stage of mapping where a record is mapped to pre-existing records based on player id and timestamp.

Figure 7
Class diagram of import prototype 1

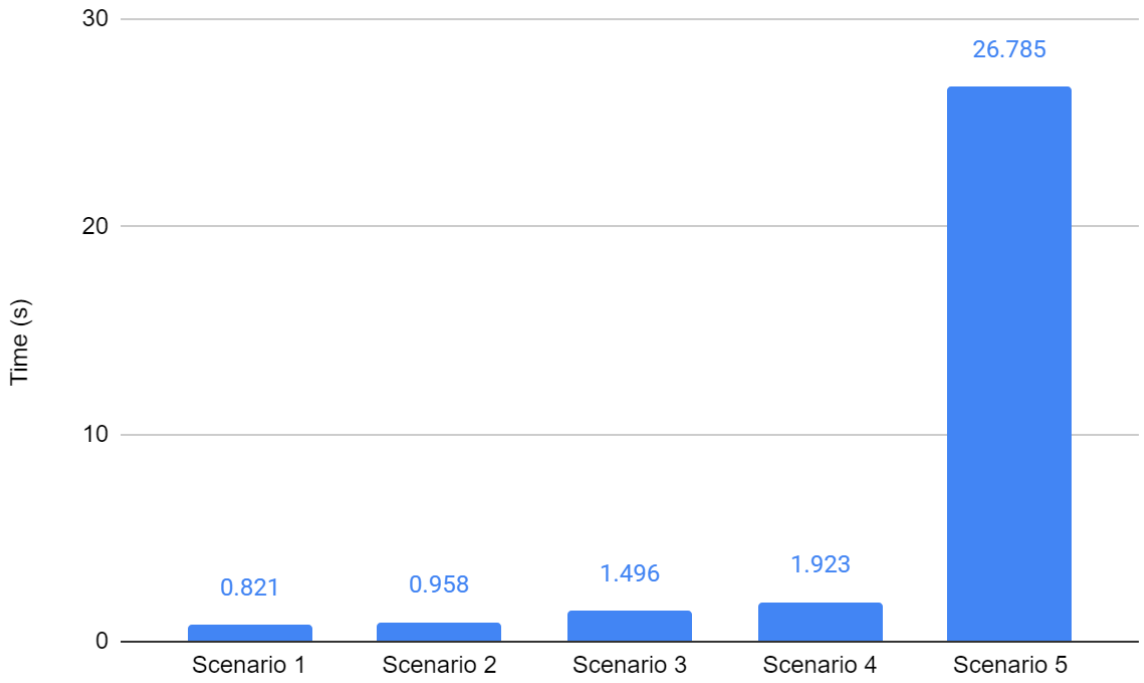


Results of performance testing:

Table 5
Import prototype 1 performance

	Scenario 1 (s)	Scenario 2 (s)	Scenario 3 (s)	Scenario 4 (s)	Scenario 5 (s)
run 1	0.92s	0.92s	1.59s	1.91s	25.3s
run 2	0.79s	0.98s	1.5s	1.78s	27.26s
run 3	0.87s	0.94s	1.39s	1.79s	26.27s
run 4	0.78s	0.97s	1.48s	1.91s	27.11s
run 5	0.73s	0.95s	1.49s	1.91s	26.61s
run 6	0.74s	0.95s	1.46s	2s	27.65s
run 7	0.94s	0.94s	1.49s	2.01s	26.28s
run 8	0.79s	0.9s	1.52s	1.98s	27.65s
run 9	0.78s	1.03s	1.54s	2s	26.43s
run 10	0.87s	1s	1.5s	1.94s	27.29s

Figure 8
Import prototype 1 average performance chart of each scenario



Overall this model performs adequately in all scenarios except 10 files with full 10 file mapping in the database. Scenario 5 is an outlier but it is partially expected because it imports the largest data sample and all records are mapped to pre-existing athlete's training. Nevertheless, the observed difference between scenario 5 and all other scenarios is larger than initially expected. The reason for this kind of difference is due to mapping algorithm insufficiency. In scenarios where data can be fully mapped, especially with larger pre-existing data in the database, it takes longer to find pre-existing records that can be mapped to newly imported data.

Advantages:

1. Takes less time to import a file
2. Import rules can be very specific therefore it is possible to further optimize the performance.
3. Easier to implement. Because it is very specific it is easy to implement field mapping as all file fields are known immediately.

Disadvantages:

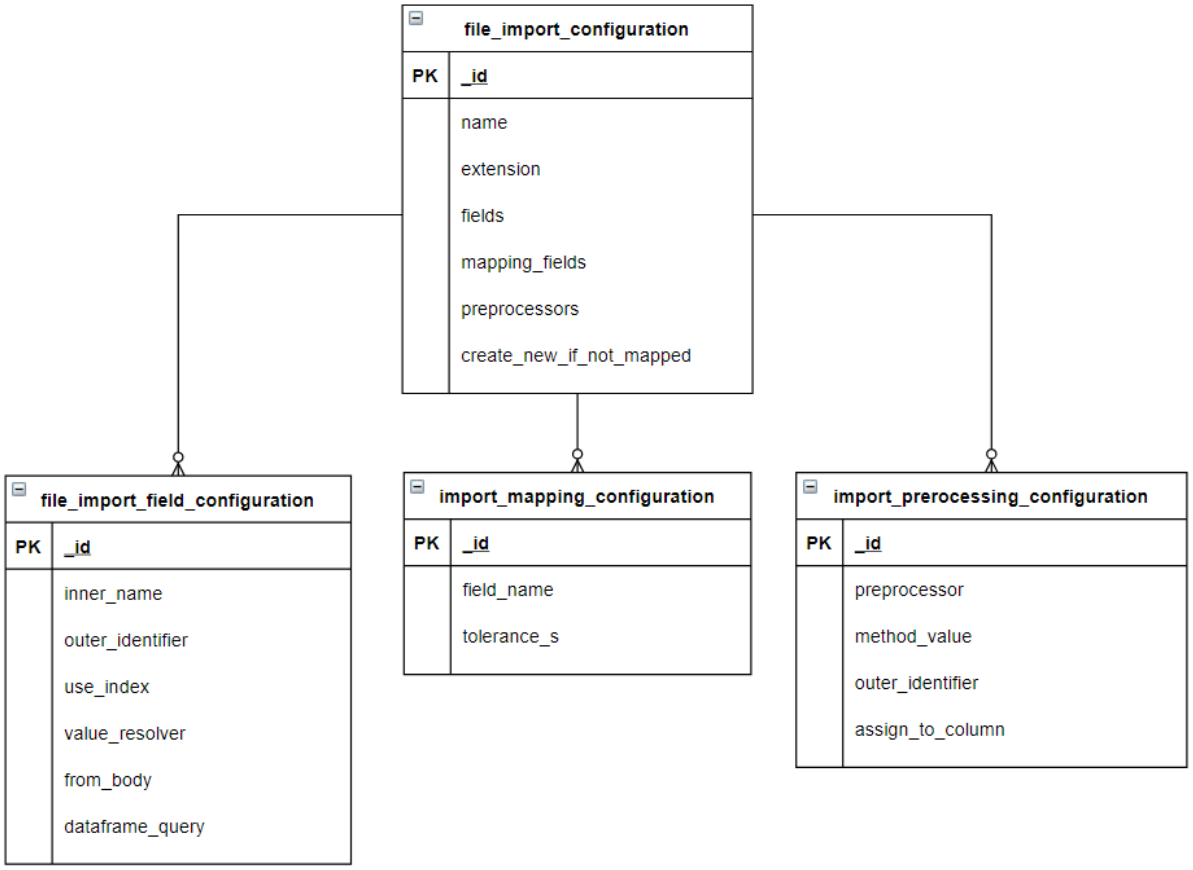
1. Each organization/club needs a separate implementation of the file reader interface.
2. Not future proof, each change to the import system must be done by changing the implementation.

6.4.3 Import prototype 2

The second import system is designed to be more configurable. This prototype constructs import rules based on the configuration data saved in the club's database. Full configuration description consists out of four database entities:

1. file_import_configuration – connects all other configuration parts and have default configuration parameters describes such as acceptable file extensions, a title for indication in the front-end
2. file_import_field_configuration – describes how specific values should be resolved from the provided document.
3. import_mapping_configuration - describes how constructed value should be mapped with other pieces of training in the system.
4. import_preprocessor – describes how the initial file should be pre-processed to efficiently resolve values later on.

Figure 9
Entity Relationship Diagram of import prototype 2



Workflow

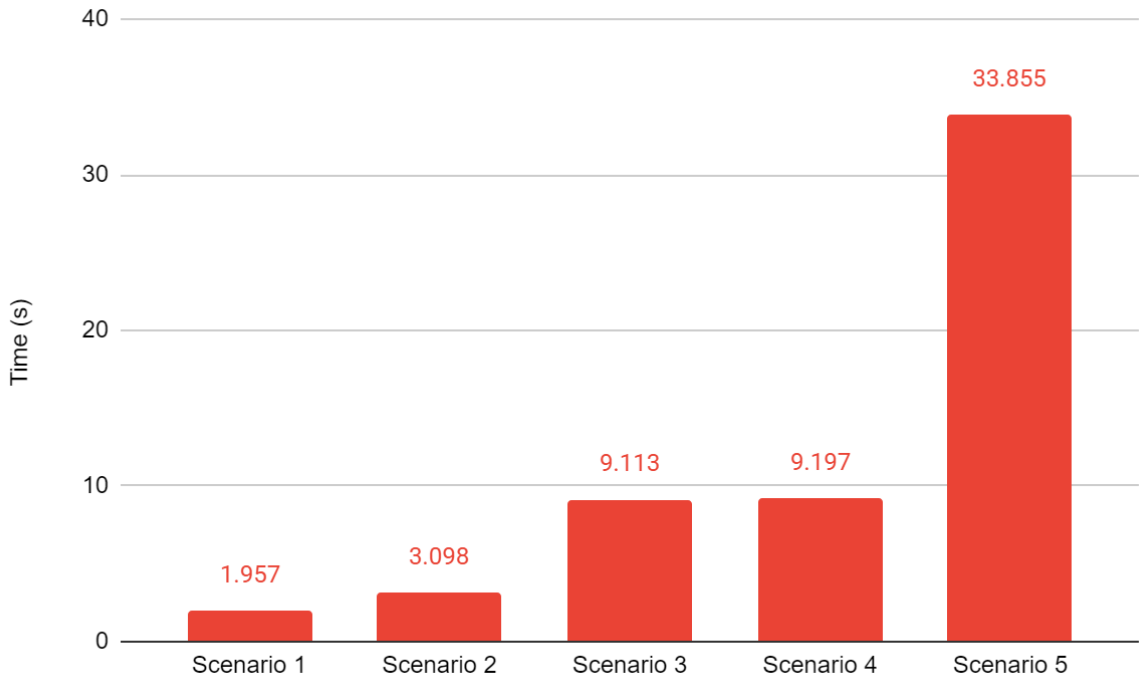
There are several different stages in importing data into the system. The first file is submitted to the back-end system with configuration id. Then “file_import_configuration” is resolved and the file is loaded into pandas Dataframe. After that “file_import_field_configuration” is retrieved from the database and one by one column values are calculated and stored in the new Dataframe. After the result dataframe is completed (all field configurations were executed), the mapping process is started. First of all, records are matched with the players in the system. If a record is successfully matched with the player it goes into another matching phase, otherwise, it is stored in the unmatched training table in the database. The next phase in record mapping is matching with pre-existing training records. This is necessary because some training records consist of several different files therefore it is necessary to have mapping rules. Several records are mapped according to “import_mapping_configuration”. Usually, records are mapped using player id and timestamp. Some sources have different time tolerances so the mapping configuration needs to specify values that have to be used. For instance, wellbeing survey data can be mapped with any timestamp within a twelve-hour period. If a record is not mapped with any other existing record, a new record is created (if not specified otherwise in the file_import_configuration).

Table 6

Import prototype 2 performance

	Scenario 1 (s)	Scenario 2 (s)	Scenario 3 (s)	Scenario 4 (s)	Scenario 5 (s)
run 1	1.85s	3.07s	8.67s	9.09s	32.83s
run 2	1.84s	3.04s	9.03s	9.26s	33.88s
run 3	1.92s	3.23s	8.91s	9.21s	34.54s
run 4	2.03s	3.09s	9.38s	9.36s	33.63s
run 5	2s	3.06s	8.95s	8.96s	33.68s
run 6	1.98s	3.12s	8.92s	9.38s	34.04s
run 7	1.97s	3.13s	8.78s	9.16s	34s
run 8	2.02s	3.11s	10.36s	9.14s	33.67s
run 9	2s	3.11s	9.17s	9.38s	34.27s
run 10	1.96s	3.02s	8.96s	9.03s	34.01s

Figure 10
Import prototype 2 average performance chart of each scenario



Overall performance of this model over different scenarios is worse than the first model. This behavior is partly because there is more information that needs to be loaded from the database (configuration parameters), partially due to inefficient code problems (discussed more in the “6.4.5 Optimization” chapter). Also, scenario 5 in this prototype has the same behavior as prototype 1.

Advantages:

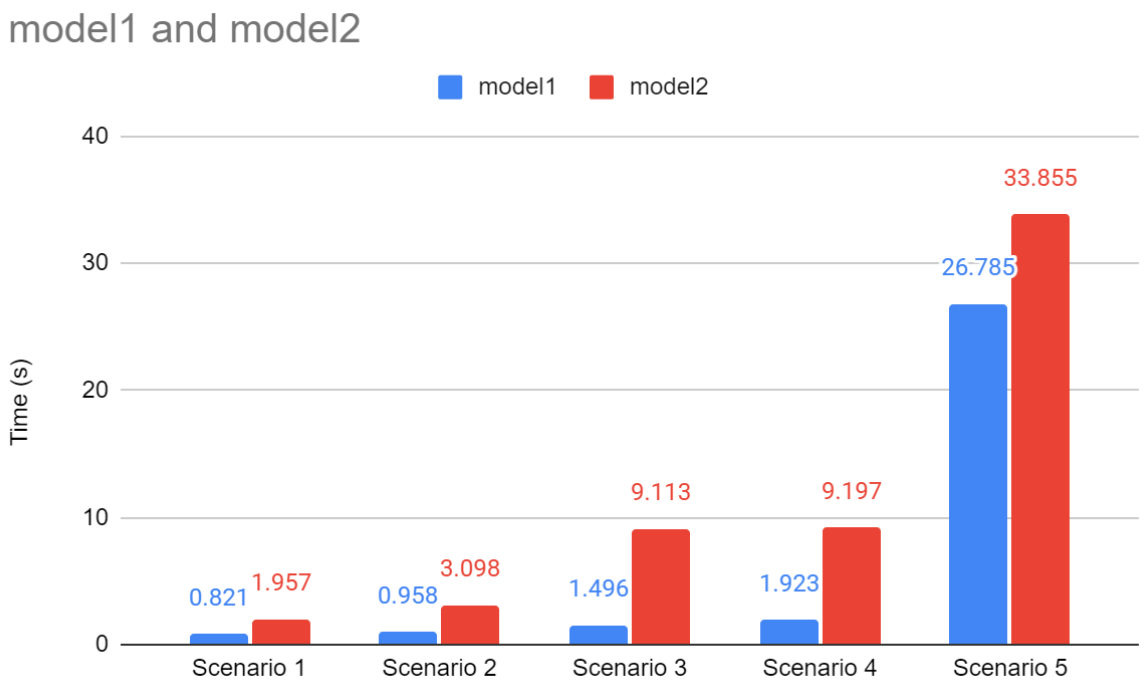
1. Very configurable. Possible to change specific import mapping methods without changing underlying code.
2. Can be further developed by adding more specific value resolvers or making configurations accessible in the front end so the user can configure it as they see fit.
3. Used a lot of reusable code therefore it can be used in other organizations.

Disadvantages:

1. Slower compared to another solution.
2. Could be hard to introduce a very specific processing method.

6.4.4 Prototype comparison

Figure 11
Import prototype performance comparison



The main difference between these two prototypes is import speed and configurability. The biggest difference between these two prototypes appears in the two scenarios. When trying to import ten files into the system but not all of them could be mapped internally therefore no mapping queries were executed in both cases. In these two scenarios, the difference in speed is about 6 times. The main reason for this difference in speed is the loading of the configuration into the server from the database and resolving column values one by one based on the configured value resolvers. Because both of these solutions are prototypes it is possible to optimize them to make them run quicker.

6.4.5 Optimization

Both of the solutions are prototypes therefore it is possible to optimize them to make them run faster. It was chosen to only optimize a single prototype due to time constraints. Mode 2 was chosen because it is more configurable and it is possible to optimize it more because as our testing showed loading times can be much quicker. Another reason for choosing the second model is the ability to further improve this solution and the ability for any club to create specific configurations that can work for their club with minimal knowledge of the system and programming.

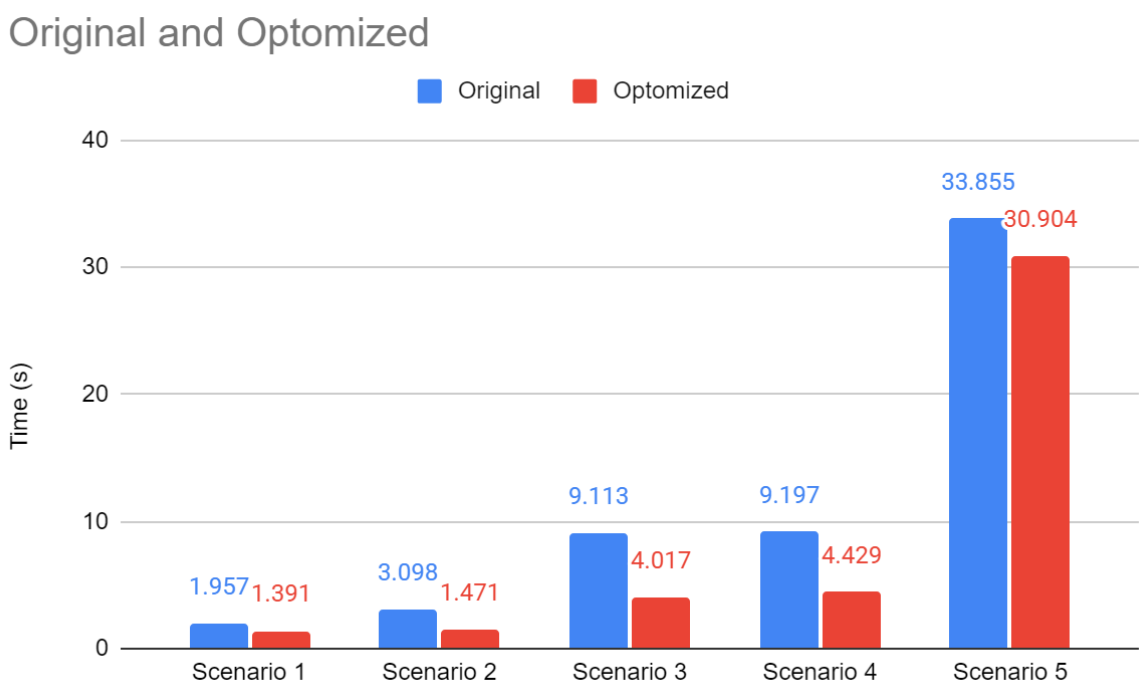
Several parts of the system were optimized. First of all, the mapping to players was changed. Each record was mapped with a player by a separate query to the database in the prototype version. In a new version based on the list of all records, a map of all possible player names is created, and then with a single query, all required player ids with player names are returned and converted to a map. This way when a record needs to be mapped with a player it can request a player id from a beforehand created map and get the id immediately without calling the database.

The second part of the optimization was done to the import field configuration loading. In the prototype version, each import field configuration was loaded separately by iterating through object ids that are stored in the file configuration object. In the optimized version, all configuration objects are retrieved with a single database call based on the list of provided ids.

The third part that was optimized is training mapping to already existing records in the database. In the prototype, version mapping was done in two parts. The first part was responsible for finding needed records for the update and the second part was assigning additional values to those records and saving them. In an optimized version, all of this is done using a single query to the database, therefore, decreasing import time.

Unfortunately, all of these optimizations did not achieve a significant improvement in speed that was expected initially.

Figure 12
Import prototype comparison (Original and Optimized versions)



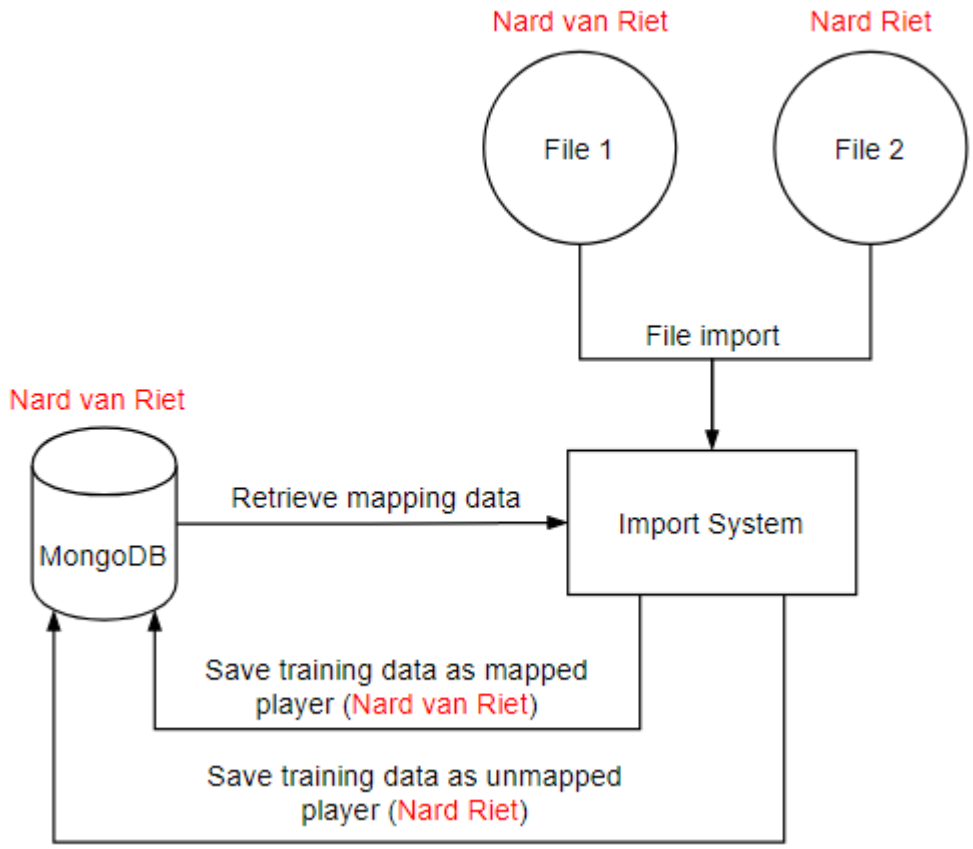
Overall all scenarios were improved. The biggest performance difference occurred in scenarios three and four. In these scenarios, ~54% increase in speed was observed. Unfortunately, the smallest performance gain was observed in scenario 5 with the longest import times.

6.4.6 Unmapped player data mapping

During the development of the import system, additional functionality was implemented to solve possible problems with player data mapping. Due to the nature of the data that is imported, it is possible that the same player can have different names in these files. For instance, the same player by the name “Nard van Riet” appears in two files that will be imported (Figure 13). The first file contains the correct name, while the second file does not contain his full name (“Nard Riet”). In this case, data only from the first file would be mapped.

Figure 13

Player mapping diagram (name in red is the same player's name in different files)



To resolve this issue additional functionality was needed.

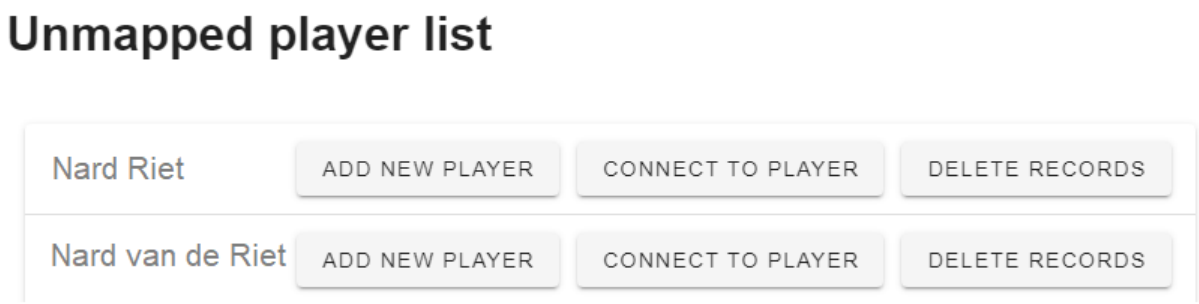
Table 7

Unmapped player data mapping requirements (Sorted by priority)

1.	Users must have the ability to map different names of the same player in order to correctly map their data.
2.	Users should be able to create new players in the system from the unmapped data.
3.	Users should be able to delete imported data if a player of that name does not exist in the system.

A new entry point in the web application was created to accommodate this functionality. On this page, users are able to decide what should be done with the unmapped data (create a new player with the unmapped data, add name variation to the existing player and re-import data, delete data). This functionality ensures that all data is mapped in the correct manner and resolves an issue of different names of the same player existing in different import files.

Figure 14
Unmapped player data mapping entry point in GUI



6.4.7 Import system testing

First of all performance testing was done with different scenarios to know how well the import system performs. This testing was done with two prototypes and the results are described in the previous chapters.

After performance testing functionality of this module was tested using functional testing. Several test scenarios were written to test import functionality. The main parts that were tested were mapping of the players from different files, data reading and manipulation, and unmapped player mapping.

These tests provide high-level assurance that the system works properly. Unfortunately, there was no time to implement unit tests and integration tests. These alternative testing methods would provide better coverage.

More details about test scenarios and results can be found in Appendix 11.3 and 11.4. For the import functional testing, there were seven test cases written and all of them passed.

7. Next steps

Improve mapping performance when the database is full of records. Explore other variables ways to map records in the database with each other to achieve better import performance.

Data filtering when importing should be improved to remove some of the not viable cases. For example, sometimes sensor data can be unreliable because some players could be transporting turned-on sensors and that would skew data for the machine learning model as the data would not be of the athlete's training.

Create a user interface for configuring import fields and machine learning parameters. It could be helpful for the end-users to be able to configure the import system themselves. The current model of the import system allows to relatively easily achieve this functionality

It could be possible to improve import time by a significant amount by using other data import methods (instead of pandas dataframes). Some alternative data reading methods could work. For instance, "csv.DictReader", "dask.dataframe", "datatable" (Yanchev, 2020)

Improve chart rendering performance, analyze use cases to reach an optimal balance between performance and records displayed. Currently, training dada rendering in the front end can be performance-heavy (depending on the size of the dataset provided). It could be optimized by having a limit of how much athlete's training data is provided by the back-end, or some other solution to increase performance.

Make an application more international. At this point the only language of the application is English. It could be useful to have multiple languages like Dutch or German because it is mostly directed at Dutch football clubs.

Create data sharing functionality so clubs could share their data to create better machine learning models and improve accuracy.

Implement unit and integration tests to find problematic places and provide an assurance of product quality.

8. Conclusion

Athlete’s performance and health monitoring are one of the most important activities in the modern sport’s industry. Tools that help any club regardless of their size observe their athlete's performance and health greatly influence their training attitude. This project’s main goal is to provide football organizations with an open-source tool to monitor athlete’s performance and try to predict how hard training should be to achieve optimal performance.

This assignment is a continuation of previous research. It is meant to connect scattered parts of the previous iteration of the system, implement missing functionality, and provide documentation so that this tool can be open-sourced. To achieve all these goals, the project was split into three phases.

The first phase was refactoring. A large part of the system was refactored to improve performance, security, and stability. Most of the changes in the refactoring phase were done by changing REST API calls to adhere to best practices and splitting large calls into smaller ones to improve performance.

The second phase was implementing missing functionality. The main functionality that was missing is the athlete's training data import method. Two prototypes were created and tested how they perform with real data. Another functionality that was very important to implement was the ability to create machine learning models, based on the athlete’s training data, for a system user. These features were monumental to implement for this application to be open-sourced.

The last phase was creating functionality for data sharing among different clubs so it could be possible to create more accurate models. Unfortunately due to the unexpectedly large amount of time required to implement other functionality this part was not implemented. Furthermore, an initial meeting with “Sport Data valley” was made but no concrete collaboration plans were discussed, therefore it was not possible to achieve the desired goal of this phase.

Even Though, not all phases were completed Main research question was successfully answered. First of all, an import system, that can handle various file formats and is configurable to accommodate a larger organization amount, was implemented. Furthermore, the system overall was simplified, documented, and made possible for open-sourcing.

Overall basic functionality of this application could be used in the professional environment already. Nevertheless, the expectation for this application is to be further improved on by other developers to give even more insight into athlete’s performance.

9. Retrospective

Overall I think this project was a success. The most important functionality was implemented and major issues with the application were fixed. It is a bit unfortunate that some of the functionality was not implemented (data-sharing platform) due to unfortunate planning changes. In the future project, I would like to spend more time clearing out details of the project so the planning would be more accurate. As with most software engineering projects, planning is one of the hardest parts because there are a lot of unknown factors that can be missed in the beginning. Nevertheless, I'm proud of the product that was created and in my opinion, it was a successful project.

10. Bibliography

Berendhaus, F., Mensvoort, W. v., Mentink, J., Nijland, L., & Temmink, S. T. e. R. (2019, 07 05). Project Topsport Load MonitorTestrapport.

M-Way Solutions. (2018, 06 19). *10 Best Practices for Better RESTful API - M-Way Solutions*. Medium. https://medium.com/@mwaysolutions/10-best-practices-for-better-restful-api-cbe81b06f291

Pagani, M. (2019, 07 23). *Big Data Analysis And Machine Learning For Football Teams*.

Yanchev, M. (2020, 02 07). *The most (time) efficient ways to import CSV data in Python*. Medium. https://medium.com/casual-inference/the-most-time-efficient-ways-to-import-csv-data-in-python-cc159b44063d

Zubavicius, R. (2020, 01). *Project LoadMonitor*.

11. Appendix

11.1 Verification testing results

Test case ID	TC-1	
Test name	User login	
Description	User should be able to login into the application using the correct credentials	
Actor	Any user	
Pre-conditions	User should exist in the system	
Steps	1. Navigate to the login page 2. Input email and password 3. Click the “login” button	
Expected result	The user is authenticated and redirected to the landing page.	
Tested by	**Result**	**Additional comments**
Dovydas Valiulis	OK	Maps to “US-1: Applicatie opstarten” in an original test report
Date	2021-06-11	

Test case ID	TC-2	
Test name	User logout	
Description	User should be able to logout	
Actor	Any user	
Pre-conditions	The user is logged in	
Steps	1. Click on the user avatar in the navigation bar 2. Click the “logout” button	
Expected result	The user is logged out and redirected to the login page	
Tested by	**Result**	**Additional comments**
Dovydas Valiulis	OK	Maps to “US-2: Applicatie afsluiten” in an original test report
Date	2021-06-11	

Test case ID	TC-3	
Test name	User password change	
Description	User should be able to change his/her password	
Actor	Any user	
Pre-conditions	The user is logged in.	
Steps	1. Click on the user avatar in the navigation bar 2. Click “Edit profile” 3. In the dialog window input a new password 4. Repeat new password 5. Click the “save” button	
Expected result	The user’s password is changed	
Tested by	**Result**	**Additional comments**
Dovydas Valiulis	OK	Maps to “US-3: Wachtwoord aanpassen” in an original test report
Date	2021-06-11	

Test case ID	TC-4	
Test name	Create club	
Description	The system administrator should be able to create a new club	
Actor	System administrator	
Pre-conditions	The user is logged in as system administrator	
Steps	1. Click on the hamburger icon in the navigation bar 2. Click “Add club” in the menu 3. Enter club name 4. Upload club image (optional) 5. Click the “create” button	
Expected result	A new club is created with provided name and image (if an image is not provided default image should be used)	
Tested by	**Result**	**Additional comments**
Dovydas Valiulis	OK	Maps to “US-4: Club aanmaken” in an original test report
Date	2021-06-11	

Test case ID	TC-5	
Test name	Edit club	
Description	The system administrator should be able to edit the club's information	
Actor	System administrator	
Pre-conditions	● The user is logged in as system administrator ● Club already exists	
Steps	1. Click on the hamburger icon in the navigation bar 2. Click "Clubs" in the menu 3. Click "Edit club" on the club card, 4. Input new information in the dialog window 5. Click "Save"	
Expected result	Club information is updated,	
Tested by	**Result**	**Additional comments**
Dovydas Valiulis	OK	Maps to "US-5: Club wijzigen" in an original test report
Date	2021-06-11	

Test case ID	TC-6	
Test name	Delete club	
Description	The system administrator should be able to delete a club	
Actor	System administrator	
Pre-conditions	● The user is logged in as system administrator ● Club already exists	
Steps	1. Click on the hamburger icon in the navigation bar 2. Click "Clubs" in the menu 3. Click "Delete club" on the club card, 4. Click "OK" on the dialog window	
Expected result	Club with all its users and players is deleted.	
Tested by	**Result**	**Additional comments**
Dovydas Valiulis	OK	Maps to "US-6: Club verwijderen" in an original test report
Date	2021-06-11	

Test case ID	TC-7	
Test name	Create team	
Description	Club administrator should be able to create teams	
Actor	Club administrator	
Pre-conditions	The user is logged in as club administrator	
Steps	1. Click on the hamburger icon in the navigation bar 2. Click “Add team” in the menu 3. Enter team name 4. Upload team image (optional) 5. Click the “create” button	
Expected result		
Tested by	**Result**	**Additional comments**
Dovydas Valiulis	OK	Maps to “US-7: Teams toevoegen” in an original test report
Date	2021-06-11	

Test case ID	TC-8	
Test name	Edit team	
Description	Club administrator should be able to edit team information	
Actor	Club administrator	
Pre-conditions	● The user is logged in as club administrator ● Team already exists	
Steps	1. Click on the hamburger icon in the navigation bar 2. Click “Teams” in the menu 3. Click “Edit team” on the club card, 4. Input new information in the dialog window 5. Click “Save”	
Expected result	Team information is updated,	
Tested by	**Result**	**Additional comments**
Dovydas Valiulis	OK	Maps to “US-8: Team wijzigen” in an original test report
Date	2021-06-11	

Test case ID	TC-9	
Test name	Add trainers	
Description	Club administrators should be able to add a trainer	
Actor	Club administrator	
Pre-conditions	The user is logged in as club administrator	
Steps	1. Click on the hamburger icon in the navigation bar 2. Click “Add trainer” in the menu 3. Enter trainer’s email 4. Upload trainer’s avatar (optional) 5. Click the “create” button	
Expected result	A new trainer is created	
Tested by	**Result**	**Additional comments**
Dovydas Valiulis	OK	Maps to “US-9: Trainer toevoegen” in an original test report
Date	2021-06-11	

Test case ID	TC-10	
Test name	Link trainers with teams	
Description	Club administrators should be able to link trainers with teams	
Actor	Club administrator	
Pre-conditions	● The user is logged in as club administrator ● Trainer exists ● Team exists	
Steps	1. Click on the hamburger icon in the navigation bar 2. Click “Trainers” in the menu 3. Click “Control teams” on the trainer’s card 4. Click the “+” icon to link the trainer with a team	
Expected result	A trainer is linked with the team and can view information about that team	
Tested by	**Result**	**Additional comments**
Dovydas Valiulis	OK	Maps to “US-10: Trainers koppelen aan meerdere teams” in an original test report
Date	2021-06-11	

Test case ID	TC-11	
Test name	Delete trainer	
Description	Club administrator should be able to delete trainers	
Actor	Club administrator	
Pre-conditions	● The user is logged in as club administrator ● Trainer exists	
Steps	1. Click on the hamburger icon in the navigation bar 2. Click “Trainers” in the menu 3. Click “Delete trainer” on the trainer card, 4. Click “OK” on the dialog window	
Expected result	Trainer is deleted	
Tested by	**Result**	**Additional comments**
Dovydas Valiulis	OK	Maps to “US-11: Trainer verwijderen” in an original test report
Date	2021-06-11	

Test case ID	TC-12	
Test name	Unlink trainer from club	
Description	Club administrator should be able to unlink trainers and clubs	
Actor	Club administrator	
Pre-conditions	● The user is logged in as club administrator ● Trainer exists ● Team exists ● Trainer and team are linked	
Steps	1. Click on the hamburger icon in the navigation bar 2. Click “Trainers” in the menu 3. Click “Control teams” on the trainer’s card 4. Click the “V” icon to unlink the trainer with a team	
Expected result	Trainer no longer linked with the club	
Tested by	**Result**	**Additional comments**
Dovydas Valiulis	OK	Maps to “US-12: Trainer loskoppelen” in an original test report
Date	2021-06-11	

Test case ID	TC-13	
Test name	Delete team	
Description	Club administrator should be able to delete teams	
Actor	Club administrator	
Pre-conditions	● The user is logged in as club administrator ● Team exists	
Steps	1. Click on the hamburger icon in the navigation bar 2. Click “Teams” in the menu 3. Click “Delete team” on the team card, 4. Click “OK” on the dialog window	
Expected result	The team is deleted.	
Tested by	**Result**	**Additional comments**
Dovydas Valiulis	OK	Maps to “US-13: Team verwijderen” in an original test report
Date	2021-06-11	

Test case ID	TC-14	
Test name	Add players	
Description	Club administrator should be able to add new players	
Actor	Club administrator	
Pre-conditions	The user is logged in as club administrator	
Steps	1. Click on the hamburger icon in the navigation bar 2. Click “Add player” in the menu 3. Enter players information (name, position, avatar, team) 4. Click the “create” button	
Expected result	A new player is created and added to a specified team	
Tested by	**Result**	**Additional comments**
Dovydas Valiulis	OK	Maps to “US-14: Spelers toevoegen” in an original test report
Date	2021-06-11	

Test case ID	TC-15	
Test name	Move players among teams	
Description	Club administrator should be able to move players from one team to another	
Actor	Club administrator	
Pre-conditions	● The user is logged in as club administrator ● At least 2 teams exist ● Player exists	
Steps	1. Click on the hamburger icon in the navigation bar 2. Click “Players” in the menu 3. Expand player’s card 4. Click “Manage player” 5. Click “Select new Team for this player” 6. Select new team 7. Click “OK”	
Expected result	A player is moved to another team	
Tested by	**Result**	**Additional comments**
Dovydas Valiulis	OK	Maps to “US-15 : Speler verplaatsen naar andere team” in an original test report
Date	2021-06-11	

Test case ID	TC-16	
Test name	Delete player	
Description	Club administrator should be able to delete players	
Actor	Club administrator	
Pre-conditions	● The user is logged in as club administrator ● Player exists	
Steps	1. Click on the hamburger icon in the navigation bar 2. Click “Players” in the menu 3. Expand player’s card 4. Click “Manage player” 5. Click “Delete this player” in a dialog 6. Click “OK”	
Expected result	Player is deleted	
Tested by	**Result**	**Additional comments**
Dovydas Valiulis	OK	Maps to “US-15 : Speler verplaatsen naar andere
Date	2021-06-11	

		team” in an original test report

Test case ID	TC-17	
Test name	Player load overview	
Description	A trainer should be able to see player’s load in graphs	
Actor	Trainer	
Pre-conditions	● The user is logged in as a trainer ● Player exists ● The Player has training data	
Steps	1. Click on the hamburger icon in the navigation bar 2. Click “Players” in the menu 3. Expand player’s card 4. Select an attribute to see in the “Physical condition” dropdown	
Expected result	A trainer is able to see performance graphs of the player	
Tested by	**Result**	**Additional comments**
Dovydas Valiulis	OK	Maps to “US-18 : Belastings overzicht van een team zien” in an original test report
Date	2021-06-11	

Test case ID	TC-18	
Test name	Detailed player training history	
Description	Trainer should be able to see detailed player’s training history with performance indicators	
Actor	Trainer	
Pre-conditions	● User is logged in as a trainer ● Player exists ● Player has training data	
Steps	1. Click on the hamburger icon in the navigation bar 2. Click “Players” in the menu 3. Expand player’s card 4. Click “Training list”	
Expected result	The user is redirected to the player's training history page.	
Tested by	**Result**	**Additional comments**
Dovydas Valiulis	OK	Maps to “US-24 : Inzicht in

Date	2021-06-11	trainingen per speler” in an original test report

11.2 Machine learning connection testing scenarios

Test case ID	TC-ML-1	
Test name	Create ML model	
Description	Club administrator should be able to create a machine learning model	
Actor	Club administrator	
Pre-conditions	● The user is logged in as club administrator ● Athlete’s training data exists	
Steps	1. Click on the hamburger icon in the navigation bar 2. Click “Manage training” in the menu 3. Click “Create a new model”	
Expected result	A new machine learning model is created	
Tested by	**Result**	**Additional comments**
Dovydas Valiulis	OK	
Date	2021-06-12	

Test case ID	TC-ML-2	
Test name	Create ML model with date range	
Description	Club administrator should be able to create machine learning model using a specific date range for included training	
Actor	Club administrator	
Pre-conditions	● The user is logged in as club administrator ● Athlete’s training data exists	
Steps	1. Click on the hamburger icon in the navigation bar 2. Click “Manage training” in the menu 3. Fill in “Start date” and “End date” 4. Click “Create a new model”	
Expected result	A new machine learning model is created	
Tested by	**Result**	**Additional comments**
Dovydas Valiulis	OK	

Date	2021-06-12	

Test case ID	TC-ML-3	
Test name	Create ML model and update existing predictions	
Description	Club administrator should be able to create machine learning model and update existing predictions	
Actor	Club administrator	
Pre-conditions	● The user is logged in as club administrator ● Athlete's training data exists	
Steps	1. Click on the hamburger icon in the navigation bar 2. Click "Manage training" in the menu 3. Check the "Update existing predictions" checkmark 4. Click "Create a new model"	
Expected result	A new machine learning model is created and player predictions are updated	
Tested by	**Result**	**Additional comments**
Dovydas Valiulis	OK	
Date	2021-06-12	

Test case ID	TC-ML-4	
Test name	Create ML model for configured measurements	
Description	Club administrator should be able to create machine learning model for configured measurements	
Actor	Club administrator	
Pre-conditions	● The user is logged in as club administrator ● Athlete's training data exists	
Steps	1. Click on the hamburger icon in the navigation bar 2. Click "Manage training" in the menu 3. Select measurement that model will be created 4. Click "Create a new model"	
Expected result	A new machine learning model is created for a specific measurement	
Tested by	**Result**	**Additional comments**

Dovydas Valiulis	ERROR	At the time of testing functionality not yet implemented
Date	2021-06-12	

11.3 Import system testing scenarios

Test case ID	TC-IS-1	
Test name	Import single training file	
Description	Club users should be able to import single player's training data	
Actor	Club administrator, Trainer	
Pre-conditions	● User is logged in as club administrator ● Import system is configured ● Player exists	
Steps	1. Click on the hamburger icon in the navigation bar 2. Click "Import data" in the menu 3. Select file that will be imported in one of the entry points. 4. Click "Submit"	
Expected result	Training data is imported into the system	
Tested by	**Result**	**Additional comments**
Dovydas Valiulis	OK	Checking if data was imported correctly was done by looking in the database records
Date	2021-06-12	

Test case ID	TC-IS-2
Test name	Import multiple training files
Description	Club users should be able to import multiple player's training data
Actor	Club administrator, Trainer
Pre-conditions	● The user is logged in as club administrator ● Import system is configured ● Player exists
Steps	1. Click on the hamburger icon in the navigation bar 2. Click "Import data" in the menu

	3. Select multiple files that will be imported in one of the entry points. 4. Click “Submit”	
Expected result	Training data is imported into the system	
Tested by	**Result**	**Additional comments**
Dovydas Valiulis	OK	Checking if data was imported correctly was done by looking in the database records
Date	2021-06-12	

Test case ID	TC-IS-3	
Test name	Import multiple training files from different sources	
Description	Club users should be able to import multiple player’s training data from different sources and data is mapped correctly	
Actor	Club administrator, Trainer	
Pre-conditions	● The user is logged in as club administrator ● Import system is configured ● Player exists	
Steps	1. Click on the hamburger icon in the navigation bar 2. Click “Import data” in the menu 3. Select multiple files that will be imported in more than one entry point. 4. Click “Submit” on both entry points	
Expected result	Training data is imported into the system and mapped correctly	
Tested by	**Result**	**Additional comments**
Dovydas Valiulis	OK	Checking if data was imported correctly was done by looking in the database records
Date	2021-06-12	

Test case ID	TC-IS-4	
Test name	Import training files of player that does not exist	
Description	Club users should be able to import multiple player's training data and if a player does not exist in the system data should be stored separately	
Actor	Club administrator, Trainer	
Pre-conditions	● The user is logged in as club administrator ● Import system is configured ● The player does not exist	
Steps	1. Click on the hamburger icon in the navigation bar 2. Click "Import data" in the menu 3. Select file that will be imported in one of the entry points. 4. Click "Submit"	
Expected result	Training data is imported into the system and stored in a different place than mapped data. Players with unmapped data appear on the "Unmapped player list" page	
Tested by	**Result**	**Additional comments**
Dovydas Valiulis	OK	Checking if data was imported correctly was done by looking in the database records
Date	2021-06-12	

Test case ID	TC-IS-5	
Test name	Create a new player from unmapped data	
Description	Club users should be able to create new players from unmapped player data	
Actor	Club administrator, Trainer	
Pre-conditions	● The user is logged in as club administrator ● Import system is configured ● The player does not exist	
Steps	1. Click on the hamburger icon in the navigation bar 2. Click "Manage training" in the menu 3. Click "Add new player" in the unmapped player list 4. Complete new player's creation.	
Expected result	New player is created and data is reimported.	
Tested by	**Result**	**Additional comments**

Dovydas Valiulis	OK	
Date	2021-06-12	

Test case ID	TC-IS-6	
Test name	Connect unmapped data to existing player	
Description	Club users should be able to connect unmapped player's training data to already existing player	
Actor	Club administrator, Trainer	
Pre-conditions	● The user is logged in as club administrator ● Import system is configured ● The player does not exist	
Steps	1. Click on the hamburger icon in the navigation bar 2. Click "Manage training" in the menu 3. Click "Connect to player" in the unmapped player list 4. In the dialog select a player that data will be mapped to 5. Click "Connect" in the dialog	
Expected result	Data of the unmapped player is imported to the connected player.	
Tested by	**Result**	**Additional comments**
Dovydas Valiulis	OK	Checking if data was imported correctly was done by looking in the database records
Date	2021-06-12	

Test case ID	TC-IS-6
Test name	Delete unmapped data
Description	Club users should be able to delete unmapped player's training data.
Actor	Club administrator, Trainer
Pre-conditions	● The user is logged in as club administrator ● Import system is configured ● The player does not exist
Steps	1. Click on the hamburger icon in the navigation bar

	2. Click “Manage training” in the menu 3. Click “Delete records” in the unmapped player list	
Expected result	Data of the unmapped player is deleted.	
Tested by	**Result**	**Additional comments**
Dovydas Valiulis	OK	Checking if data was imported correctly was done by looking in the database records
Date	2021-06-12	