RUGBY CLUB MAGEMENT SOFTWARE

ALGORITHM AND PROBLEM SOLVE REPORT

TABLE OF CONTENT

INTRODUCTION

When you talk about a rugby team, it doesn't matter the level or association, you only talk about which team won, how many points were scored, who were the best in the game and who were the worst, but you never talk about how difficult it is to manage a club and how it recruits its players.

Being sports teams in the entertainment industry, many people are interested in being part of this type of project, therefore, a management system has been developed for the Rugby Club that covers several needs, from the organization of applicants to the Club to the assignment of coaches and teams to those applicants. Drastically simplifying the internal administration of the club by offering a fast and interactive system for those who use it.

Throughout this report, the different design models will be reflected, reasons for the choice of key elements when developing the system and challenges that arose in its development and how they were solved, as well as why other alternatives were not taken.

ANALISYS REQUIREMENTS FOR RUGBY CLUB DATABASE SYSTEM

Before start the development of the Rugby Club System, the first step was to understand all the requirements about the software, we start breaking every requirement into more manageable pieces, imagining the outputs that should be displaying on the command line and the general structure that our code should follow, this to simplify and giving a clearer representation of how the user can interact with it.

But to do what was mentioned above, in the development of this system, tools have been used that helped us to see in a more concise way the design to follow for its correct elaboration.

One of the most important challenges in carrying out this was data modelling, since following the best practices for good data design is crucial to having a system that is successful in reading and processing it. For this, UML diagrams were created, which are be shown below:

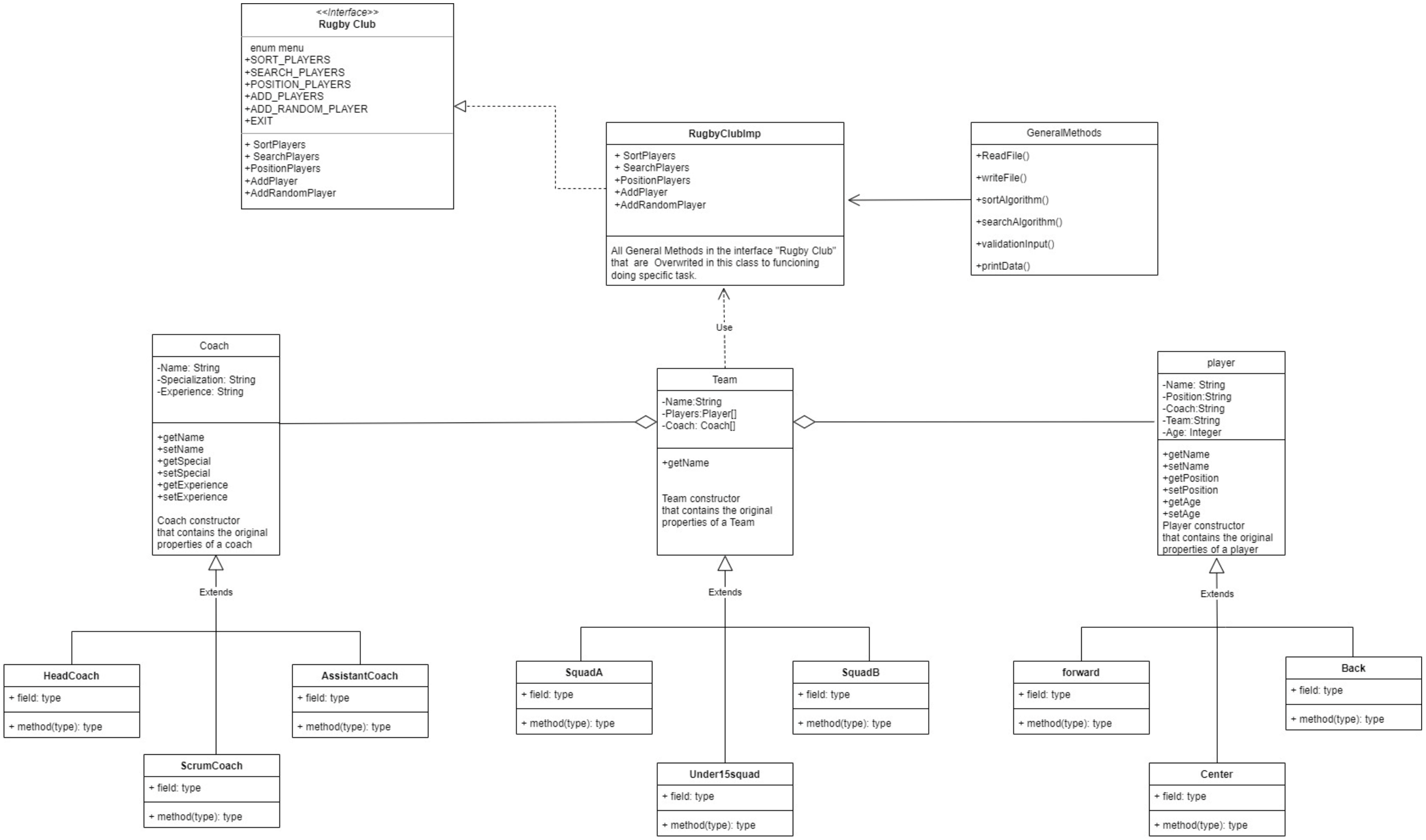


Figure 1 First UML diagram of classes, which started modelling the data before coding started

When the design of the system began, it was necessary to have a guide to follow, to be able to start, since being a system with numerous classes and functions starting the code fully can be counterproductive in times, since data management problems can be generated, but having this guide does not mean that things cannot change, therefore, a good practice to carry out is to have a change control of the different designs that were made until reaching the final system.(UML class diagrams, 2023)

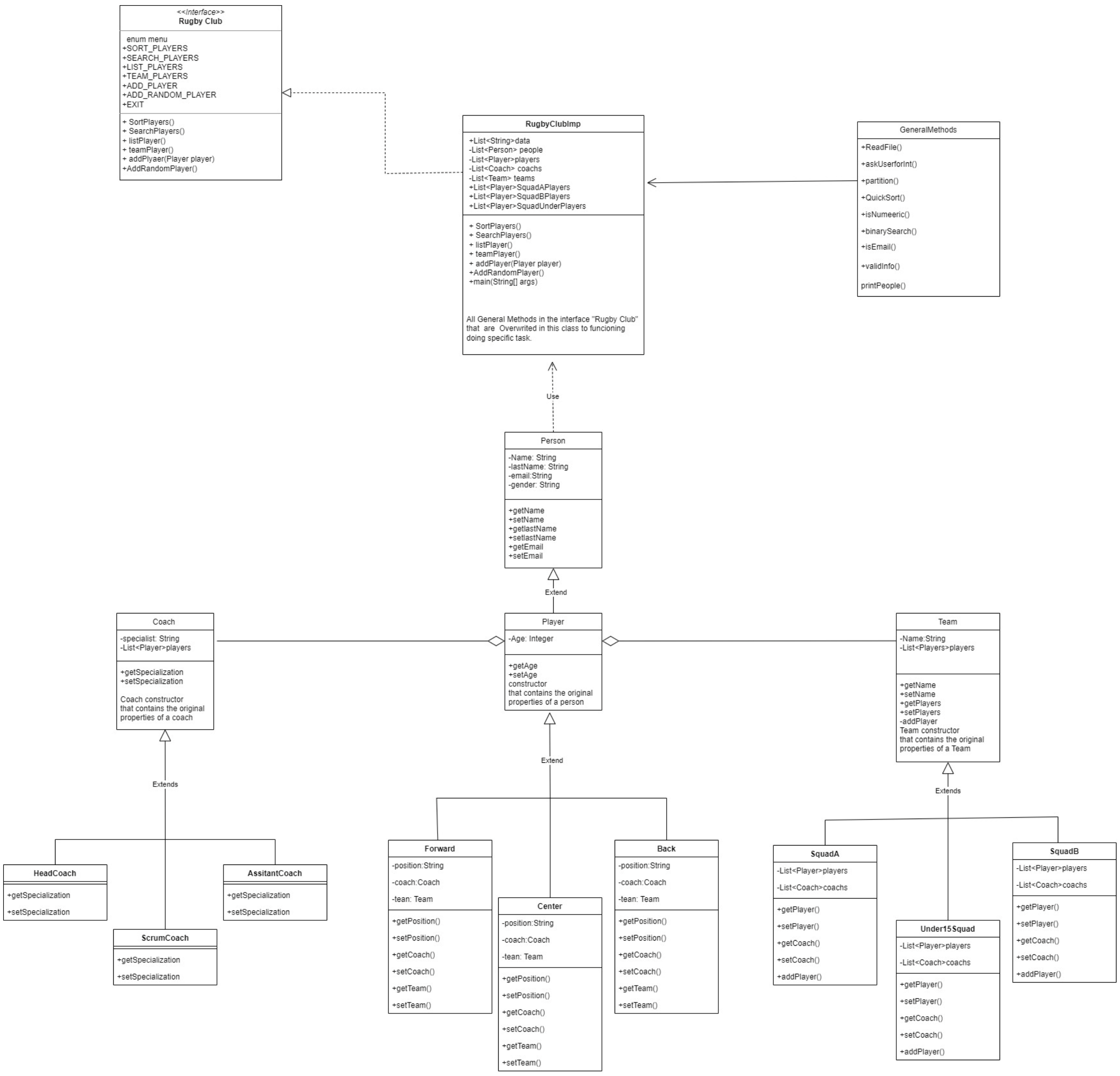


Figure 2 Update of the first UML diagram shown above.

For the first UML, a structural type called class diagram was used, which provides us with many benefits such as a clear visualization of the management of classes and data, showing us schematically an overview of how the system works thanks to the detailed information within it, with this it helps us to efficiently make decisions in the implementation of the system.

In the first instance we took this type of diagram to begin with, since it is one of the most important and common because of its attributes, this helps us to visualize the inheritances and associations between classes in the scheme to have a high-level design of a static system, it makes it one of the main modelling diagrams to choose over other types of UML Diagrams.(UML class diagrams, 2023)

For example, taking into account the first-class diagram shown (figure 1), we can realize that the one in figure 2 is quite different, when we began to code on the first UML of classes, although some processes could be carried out, due to the structure that can be observed where the teams were used as the centre where the information of the coach and the players can be stored. Which is not illogical and may be correct, however what we wanted to achieve in the system was not the best way.

That is why it was updated to a second UML, in which important changes can be observed such as the addition of some more methods to be carried out in the program which due to the nature of the structure of how the data was rearranged, can be accessed in a simple way, 3 super classes were created:

- Person
- Coach
- Team

These 3 Super Classes are extended other classes:

- Person Super Class $\rightarrow$ Players Sub-Class $\rightarrow$ 1. Forward, 2. Center and 3. Back
- Team Super Class $\rightarrow$ 1. Squad A, 2. Squad B and 3. Under 15 Squad
- Coach Super Class $\rightarrow$ 1. HeadCoach, 2. ScrumCoach and 3. AssistantCoach

To help us define how the structure would be managed, when a person became a player and how to add the classification in a team, we had several doubts and it was posed as a high-risk problem to solve since it would define a very important part of our data management within the system. Therefore, a problem map was made to help us clarify this which can be seen below:

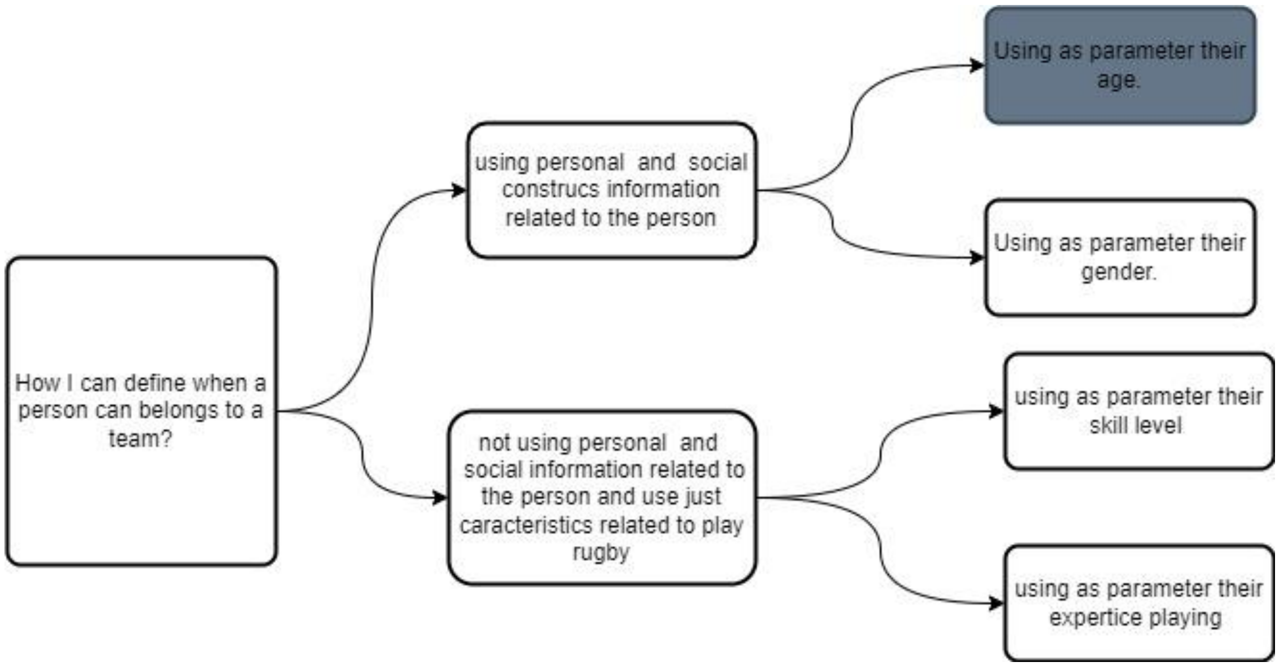

How I can define when a person can belongs to a team?
using personal and social construcs information related to the person
not using personal and social information related to the person and use just caracteristics related to play rugby
Using as parameter their age.
Using as parameter their gender.
using as parameter their skill level
using as parameter their expertice playing

Figure 3 Mapping issue solving to solve a problem when a player belongs to a team

In this map we can see some of the options that were in mind, and in grey-blue colour the option chosen, the other alternatives even though they were good, and could have been implemented, by using age as a parameter, it was easier to make sure that people could be included within the teams in a fairer way. Considering that nowadays, using gender as a parameter could be controversial, the parameters that are based on the characteristics related to playing rugby although they would be the most logical to implement, were not taken because we believe that it is job of the club staff to identify those parameters after the coaches do their tests on the players already within the team for their classification.

Once this was defined, when we were carrying out the system we found ourselves in a situation of uncertainty, since we did not have a clear idea of how our objects should look with real data, but to have a better idea of how the handling of our objects looks in practice, a UML diagram of objects has been made which can be seen below:

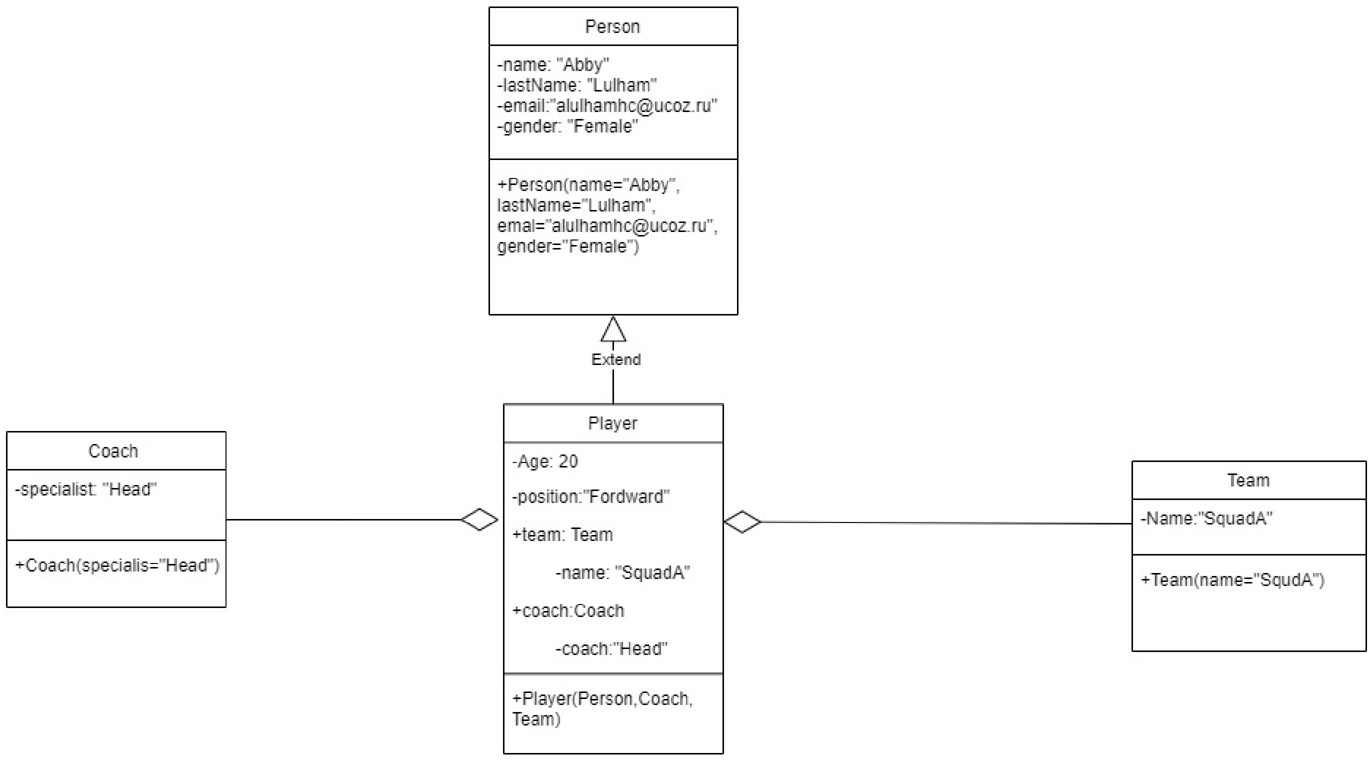


Figure 4. UML diagram of objects for a player object in the Rugby Club system

This diagram gives us the facility to be able to see a little more in detail the attributes and connections that each class has with each other, we can reflect real information, of the assignment of an applicant to the Rugby Club, and how he becomes a player of the club, integrating him with his coach and his team(All UML Diagrams in 10 minutes, 2021).

One of the great advantages that this type of diagram gives us with respect to others and why it was chosen, is that it can show the handling of our objects in a flexible way by being able to show real data, giving those who see it in a practical way the relationships and functions of our classes. solving the problem of uncertainty by not knowing what these relationships would be like.

A good alternative to this type of diagram was the component diagram, with which we could explain the relationships between our objects in more depth, but we used this object diagram to be able to visualize how the real data behaved in a process statically.

Once we were clear about what our structure would be, we had to know how the behaviour or use that a user could give to the complete system would be, to solve this problem we used a UML behaviour diagram called "Use case", which is illustrated below:

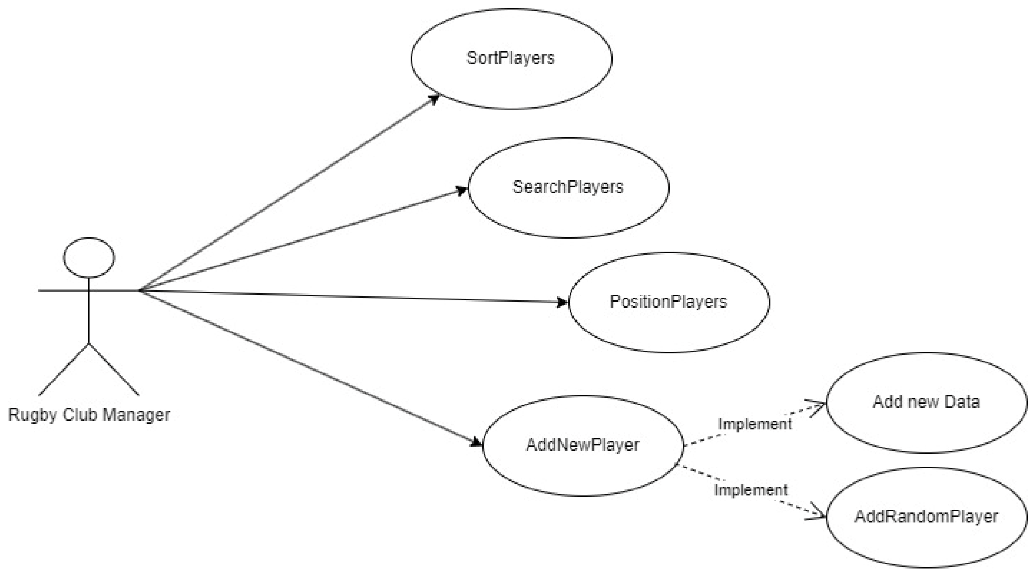


Figure 5 UML behaviour diagram -user case-

With this diagram we can observe in broad strokes the context in which a user would interact with the system, this helped us to clarify the main functions that the system had to perform, since we only wanted to know on a large scale the operation at this stage, we did not opt for another UML that gave us in-depth detail of the interaction of a user and the program, we just needed to see the influence of the system's functionalities on the user at a high level.

SYSTEM PERFORMANCE AND OPTIMIZATION

With the realization of these diagrams, it is clearly shown the path through which the Rugby Club System should go, the code was completed according to the changes mentioned above, but we arrive at one more problem, the performance of the system and its optimization, one of the most important parts to take into account within the development of a software, it's how it processes information, and much of the tasks that are done with information are done with algorithms.

One of our main objectives was to choose the best algorithms that would adapt to our client's requirements, for this the following algorithms were taken into account.

The following recursive algorithms were used to perform the sort data tasks:

1. Quick Sort algorithm (O (n Log n))
2. Merge Sort algorithm (O (n Log n))

For data search tasks:

1. Linear Search algorithm (0 (n))
2. Binary Search algorithm(O (Log n))

Choosing the best among these algorithms to implement it in the system was not an easy task, since for this we have to take into account its time and space complexity. For the first one the time it took to process the data according to its size was recorded, for this several tests were carried out with different data sizes (1000, 3000, 5000 and 10000 data rows) to see it in an easier way in tables and graphs.

1k of data											
Algorithm	1	2	3	4	5	6	7	8	9	10	Average
QuickSort	2,641,999	2,454,601	2,261,099	2,411,400	2,539,701	1,992,899	2,113,000	2,359,100	2,585,000	2,591,000	2,394,980
MergeSort	2,360,000	2,128,900	2,135,600	2,266,600	2,108,600	2,225,800	2,191,700	2,213,399	2,084,201	2,155,599	2,187,040

3k of data											
	1	2	3	4	5	6	7	8	9	10	Average
Quicksort	5,387,600	4,996,300	4,944,000	5,073,500	5,065,300	5,114,100	4,850,100	4,935,800	5,129,400	5,127,900	5,062,400
MergeSort	4,009,000	4,275,300	3,691,800	4,285,700	4,040,900	4,345,400	3,948,800	3,755,700	4,009,800	4,252,600	4,061,500

5K of data											
	1	2	3	4	5	6	7	8	9	10	Average
Quicksort	9,651,400	9,416,100	10,165,900	9,612,400	9,956,700	9,428,200	9,430,100	9,641,700	9,474,500	9,609,600	9,638,660
MergeSort	6,369,300	6,086,400	6,456,000	6,052,100	6,256,600	6,518,000	6,172,900	6,446,100	6,955,100	6,249,700	6,356,220

10K of data											
	1	2	3	4	5	6	7	8	9	10	Average
Quicksort	13,632,000	13,620,400	14,007,500	14,310,100	14,500,000	13,687,200	13,711,200	13,721,700	14,118,900	14,093,000	13,940,200
MergeSort	14,096,700	11,374,400	13,144,200	12,219,300	11,667,900	11,734,300	11,917,700	11,479,500	11,755,700	11,584,700	12,097,440

*Figure 6 QuickSort and MergeSort testing results *measurements are in nanoseconds*

As can be seen in the table above, 10 tests were performed for each data size, taking the nanoseconds it took the algorithm to perform its task.

At the same time, it can be observed that for each size of the data, the algorithm behaves differently, and although it can be seen that from 1,000 to 5,000 the performance of the Merge Sort increases drastically compared to the Quick Sort since the latter shows longer times, when large amounts of data such as 10,000 are handled, some of the tests shown very close times and even in some the Quick Sort performs the task faster, But this doesn't mean it's the best, as it was only 1 time out of 10, so we made an averages table to see the overall behaviour of both algorithms

Data Size	QuickSort (Nano-second)	MergeSort (Nano-second)
1k of data	2,394,979.90	2,187,039.90
3k of data	5,062,400.00	4,061,500.00
5k of data	9,638,660.00	6,356,220.00
10k of data	13,940,200.00	12,097,440.00

Figure 7 Average results of 10 testing in each size of data for Quick Sort and Merge Sort algorithms

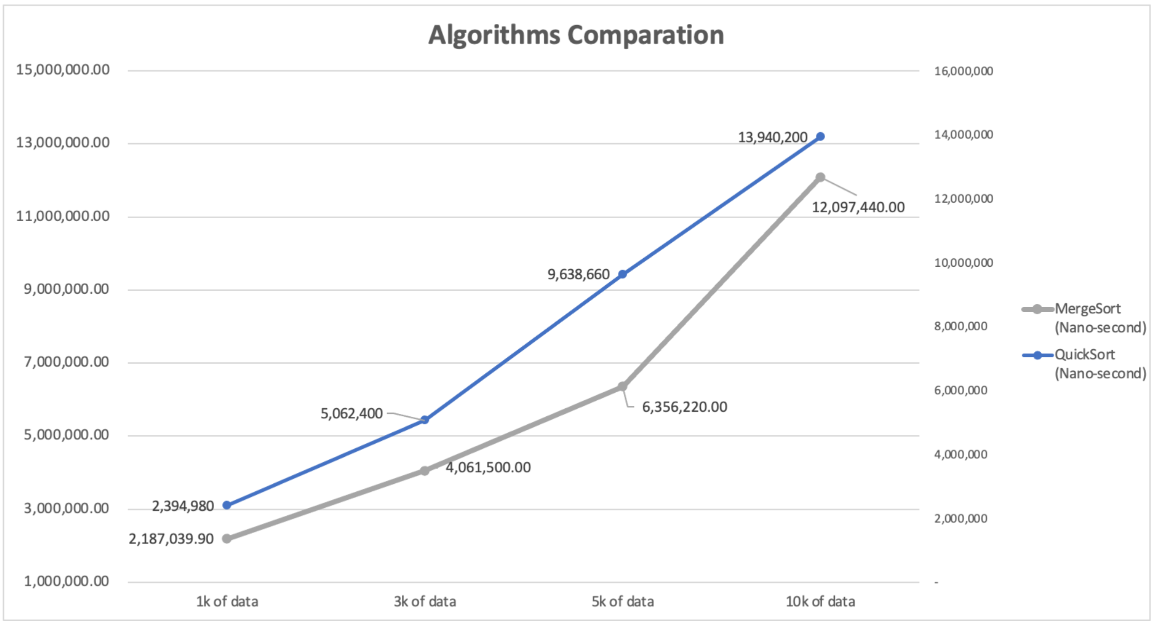
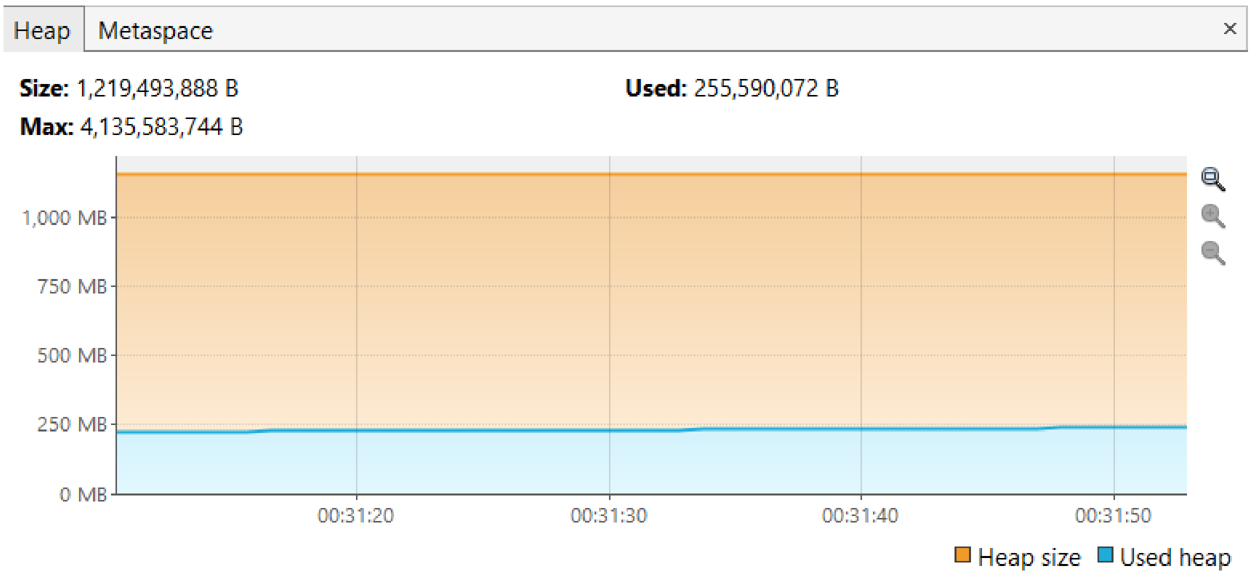


Figure 8 Quick Sort and Merge Sort Average graph results

Although the differences in the times of 1,000 and 10,000 data is not very big, Merge Sort is still below in the times over quick sort, making it the best algorithm to implement in the system in terms of time for all the data sizes used in these tests.

As far as space is concerned, one way to measure it is the processing resources that it takes to perform the task, for this we use a tool called VisualVM in its version 2.1.7(visualVM., 2023)., which gives us in real time the amounts of memory that a software is using, with this we monitor our system optimally, giving us results of how much memory the system is using when running it with the different algorithms.

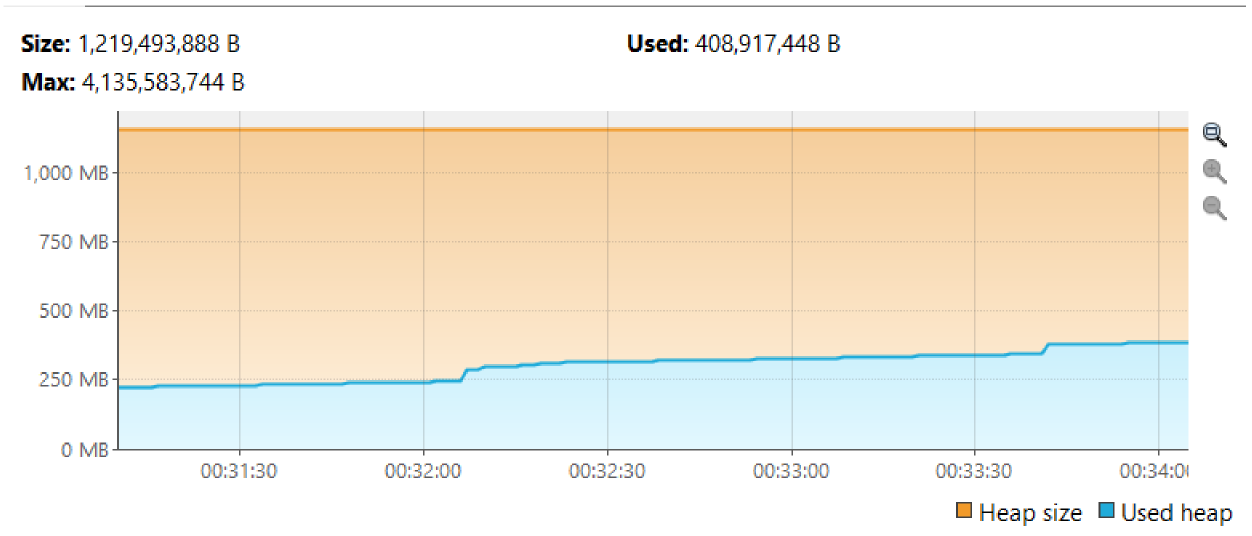
In the following image we can see a graph of how our memory is with respect to the open system before using a function that uses the Quick Sort algorithm.



Size: 1,219,493,888 B **Used:** 255,590,072 B
Max: 4,135,583,744 B

Figure 9 VisualVM System Memory before using Quick Sort

Same graph using Quick Sort:



Size: 1,219,493,888 B **Used:** 408,917,448 B
Max: 4,135,583,744 B

Figure 10 VisualVM System Memory after using QuickSort

It can be clearly seen that from 240 MB of the system when it was at rest, to an immediate increase of 290MB, but at the end of the process it continues to use resources due to the effort it has just made, reaching up to 390MB of memory before starting to go down again.

The following graph shows the system memory before using the Merge Sort:

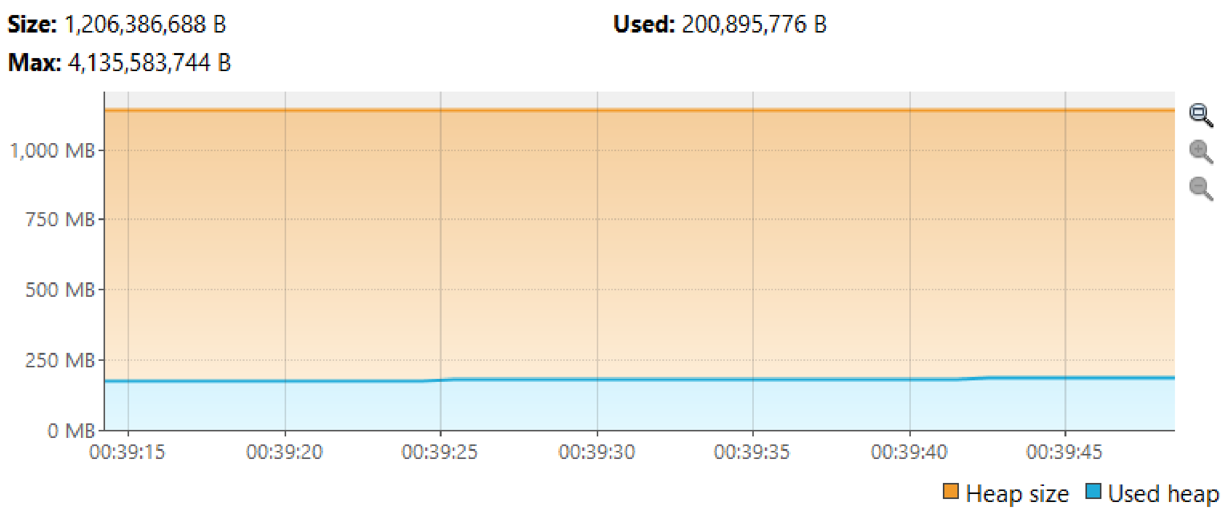


Figure 11 VisualVM System Memory before using Merge Sort

Same graph using Merge Sort:

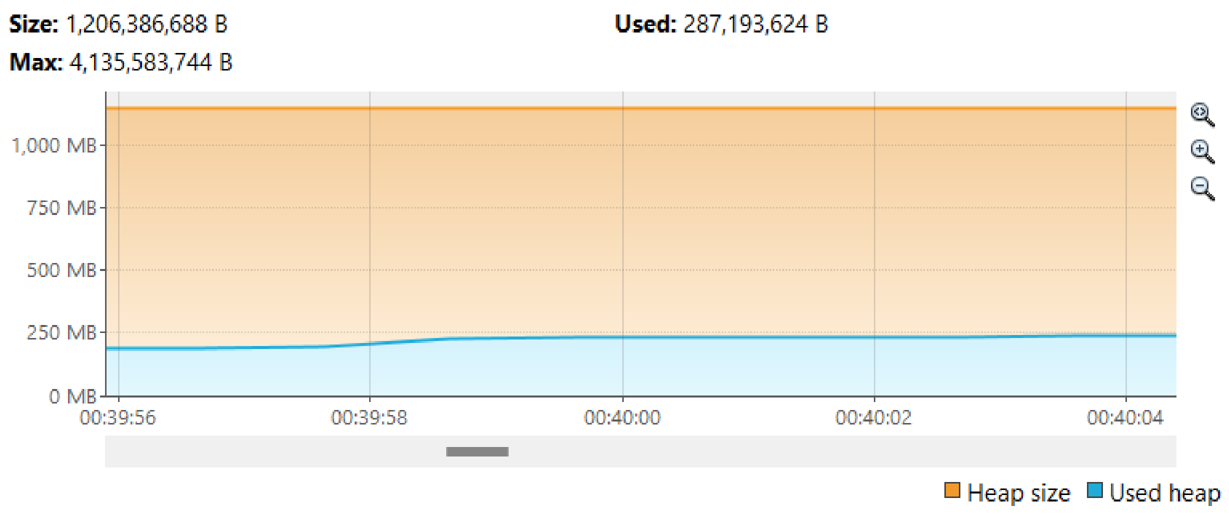


Figure 12 VisualVM System Memory after using Merge Sort

It can be clearly seen that from 192 MB of the system when it was at rest to an immediate increase to 237MB, but where it can be determined with greater certainty that the Merge Sort is a more stable algorithm is what happens after using it, since it can be seen that with respect to the Quick Sort graph which again has another peak and continues to rise taking resources, The merge sort, even though it increases the use of the system memory a little, is still stable reaching a maximum of 274MB before returning to sleep without taking up so many system resources, making it the best sort algorithm to implement for the Rugby Club system.

For the analysis of the search algorithms, something similar to that of the sort algorithms was carried out, below, is shown in table below:

Algorithm	1	2	3	4	5	6	7	8	9	10	Average
linear	192,100	123,300	339,600	151,900	38,300	48,700	46,900	37,400	29,100	86,500	109,380
binary	79,800	74,700	47,400	36,700	41,700	34,400	41,700	47,300	39,900	15,300	45,890

Figure 13 Result of testing Linear and Binary search algorithms

For the tests of these algorithms, the same search objective was used in each of the tests, in order to accurately reflect the time searching for the same objective, you can observe a great advantage of the binary algorithm over the linear one as far as time is concerned, now let's look at its graph:

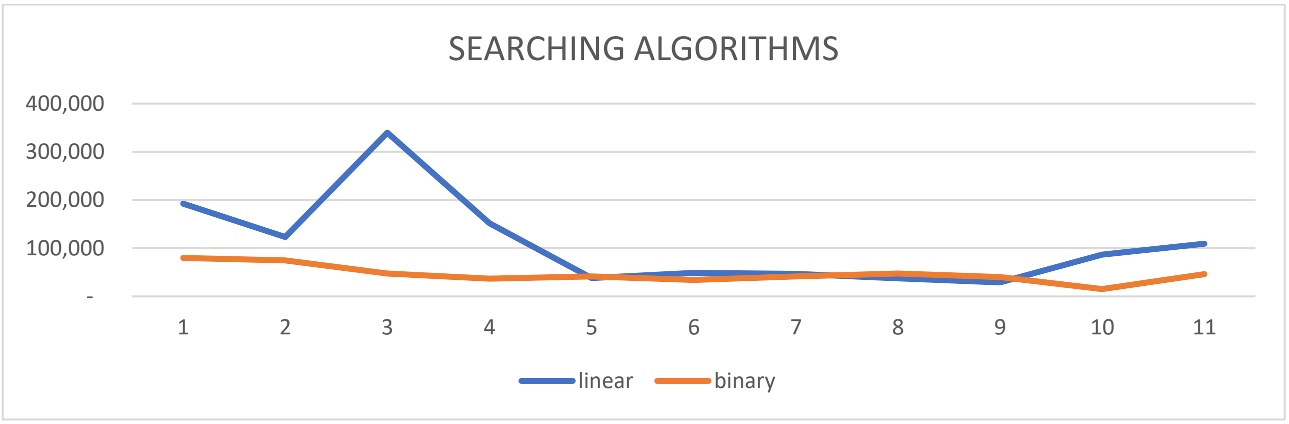


Figure 14 Graph of Linear and Binary algorithms testing results

At first glance we notice several peaks in the linear search, making it very unstable for the requirements of the system, on another hand the binary search algorithm is seen as a fairly stable algorithm in terms of time compared to the linear search, but now we must analyse how its behaviour is due to the space it uses.

Prior to using the Linear Search algorithm our system memory was as in the following graph:

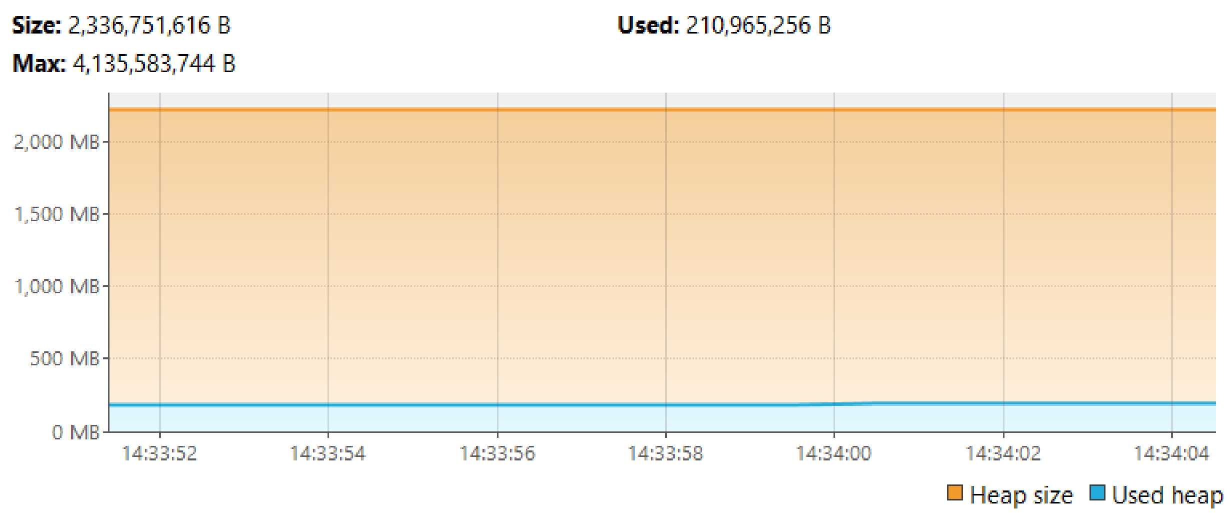


Figure 15 VisualVM System Memory before using Linear Search

Same graph using Linear Search:

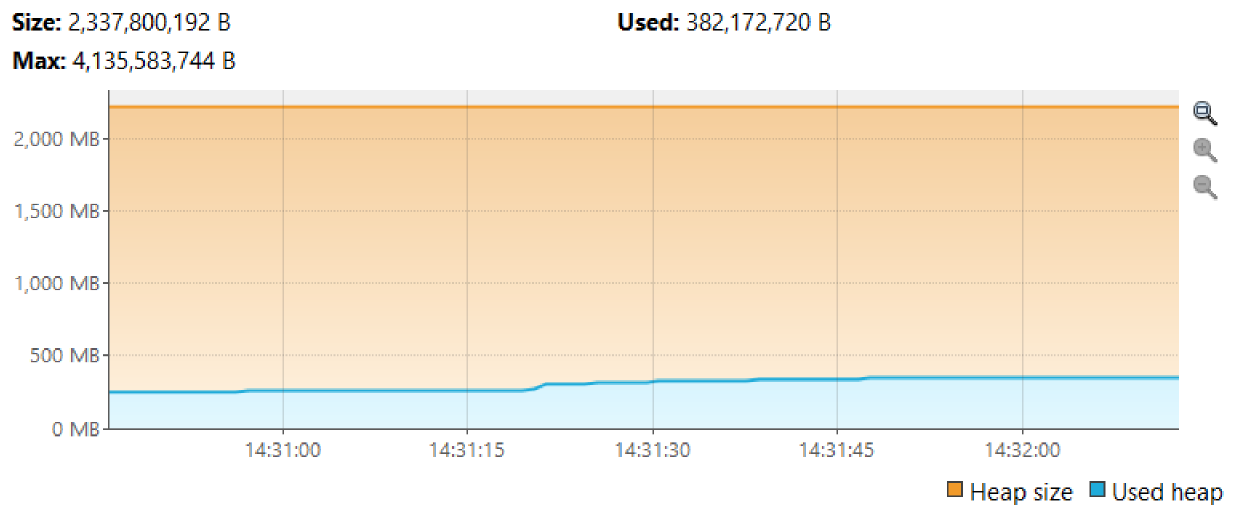

Size: 2,337,800,192 B **Used:** 382,172,720 B
Max: 4,135,583,744 B

Figure 16 VisualVM System Memory after using Linear Search

We can observe a considerable increase in resource usage from the 201MB with which it was at rest, showing an immediate peak at 320MB, continuing to use resources until it reaches 365MB.

Below is the system memory utilization at rest before using Binary Search.

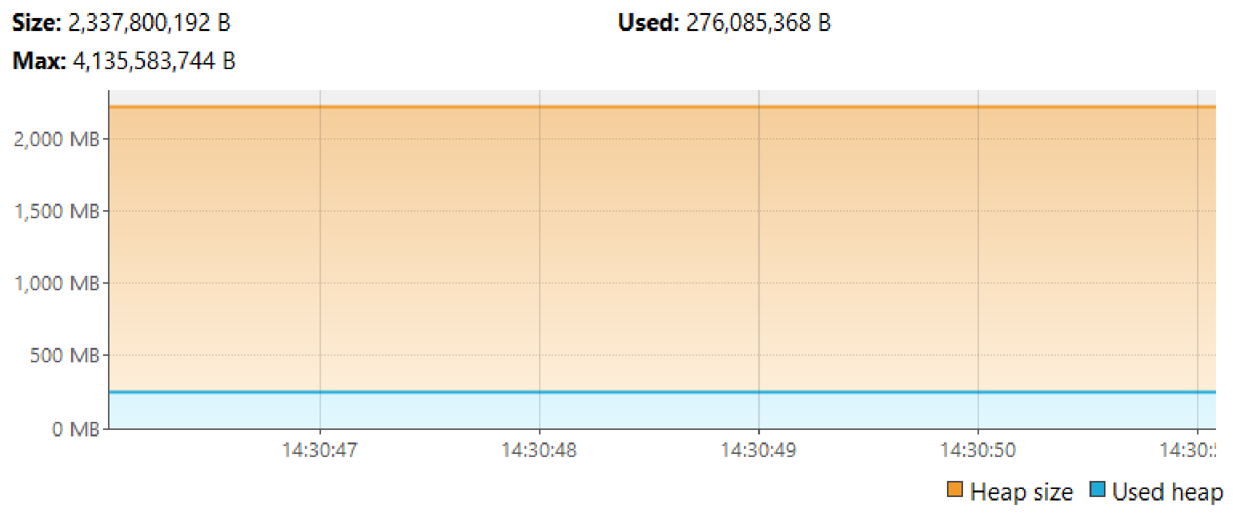
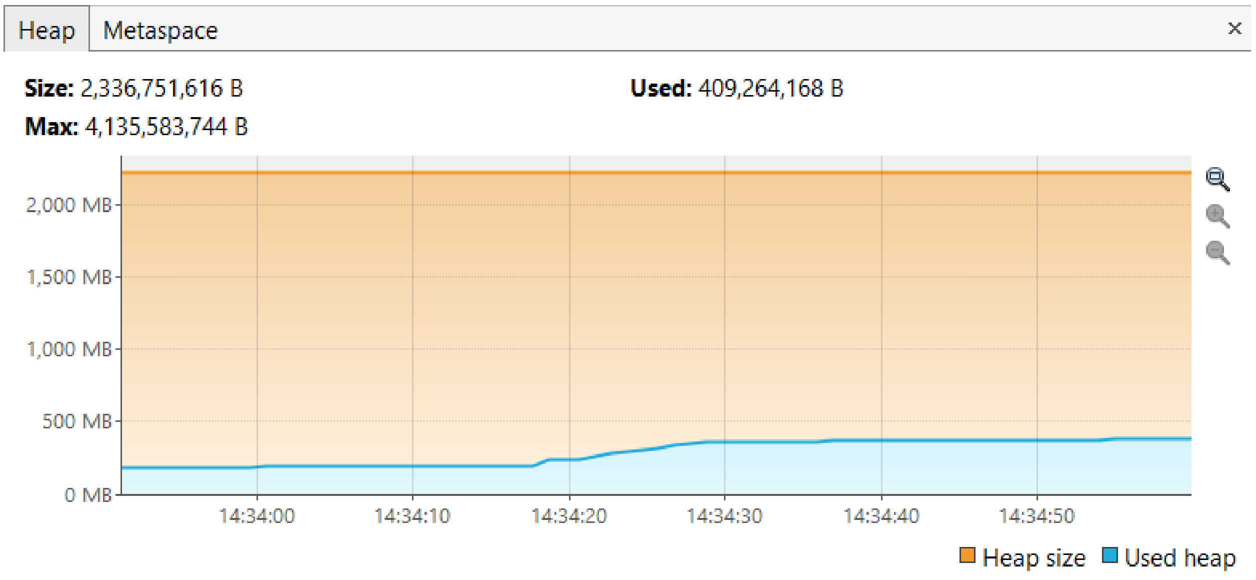

Size: 2,337,800,192 B **Used:** 276,085,368 B
Max: 4,135,583,744 B

Figure 17 VisualVM System Memory before using Binary Search

Same graph after using binary search:


Heap | Metaspace

Size: 2,336,751,616 B **Used:** 409,264,168 B
Max: 4,135,583,744 B

Figure 18 VisualVM System Memory after using Binary Search

We can observe a considerable increase in resource usage from the 260MB with which it was at rest to show an immediate peak at 300MB continuing to use resources until it reaches 390MB.

To conclude with this, **Merge Sort** and **Binary Search** algorithms will be used for the Rugby Club database system, which provide us with an efficient system when handling the data no matter how large it may grow, making the system functional for longer, likewise, taking into account the different performance and times shown by the algorithms choosing merge sort and binary the computer on which the system is located will use less Both memory and CPU resources when performing the different tasks, which can translate into less wear and tear on the device and savings in energy use, which will help save on the club's electricity bills.

FINAL STRUCTURE DESIGN FOR RUGBY CLUB DATABASE SYSTEM

When we finished coding the main tasks that the club required, we wanted to know how the complete interaction would be from the moment a person came to apply to belong to the club, until the Rugby Club Manager closed the system, therefore, with the help of a behaviour UML called sequence we could see exactly what we needed, which is described below:

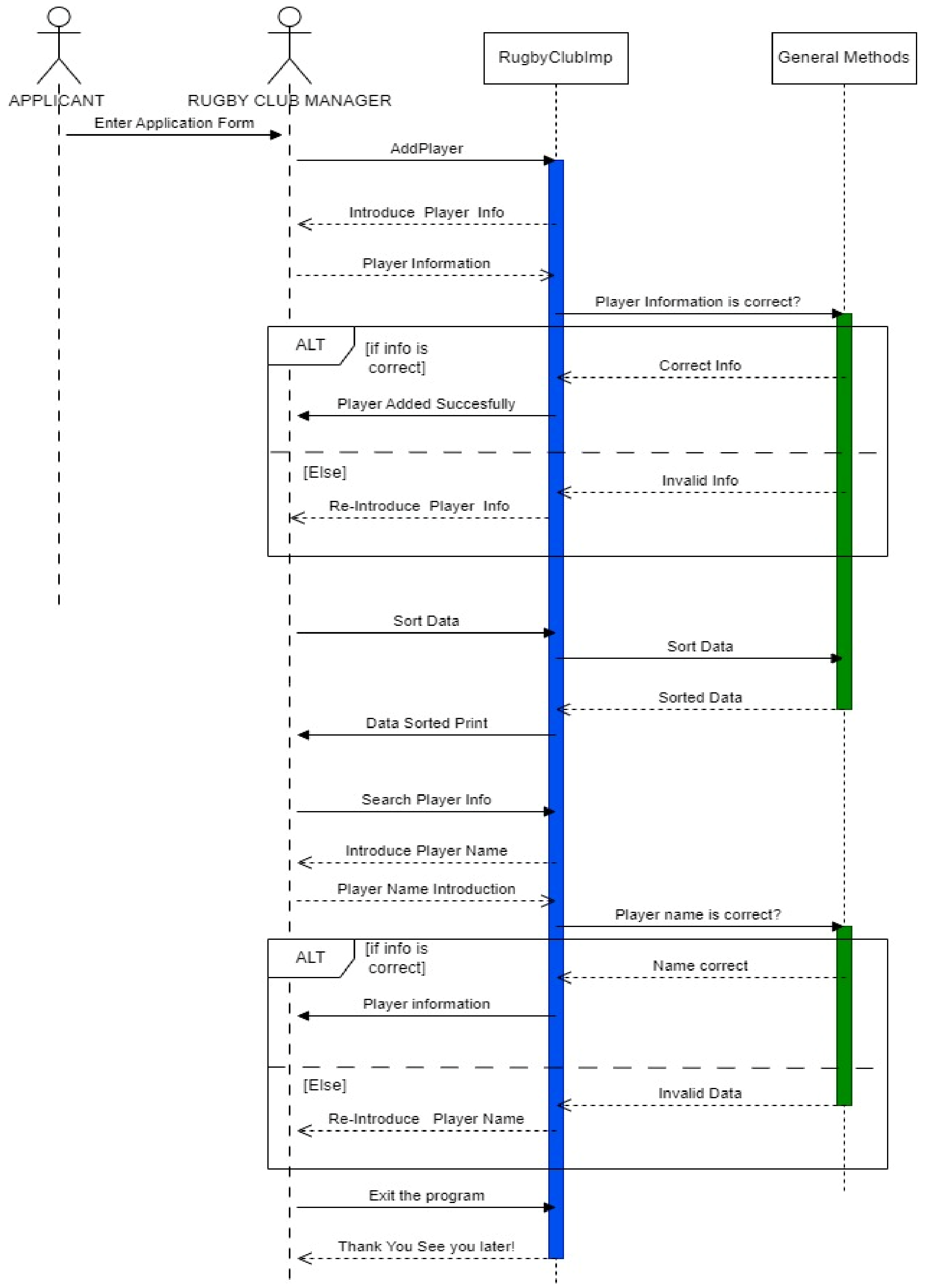


Figure 19 Interaction between user and proposed Rugby Club database

This diagram helps us to understand the operation and interaction that the user has with the system, thanks to this UML we can analyse in more detail each -use case- shown above, focusing especially on the -lifeline- of each of the processes, giving us a broad view of the behaviour that our implementations would have.

Compared to other behavioural UMLs, it's easy to visualize a system's interaction flow dynamically at a low-level, it's easy to update and keep track of what's added, as well as pretty good at helping us identify different types of problems that can arise only when you're interacting with a system and how to improve it so that it works properly(Lucid chart Team, 2023).

Taking into account the above, we realized that our system did not provide all the basic services that were required by the rugby club, we needed one more use case, that of being able to search for any applicant before they were a player, as well as that it was important for the user to be able to see a list of all the players who were part of their club at the time, and how many players each of their teams had, taking into account that there was little time left for the delivery of this prototype to the club, we created a Mapping Diagram and a Project Definition Card where we could see exactly the structure of the system and its processes in a clear way to focus only on those works, which is shown below:

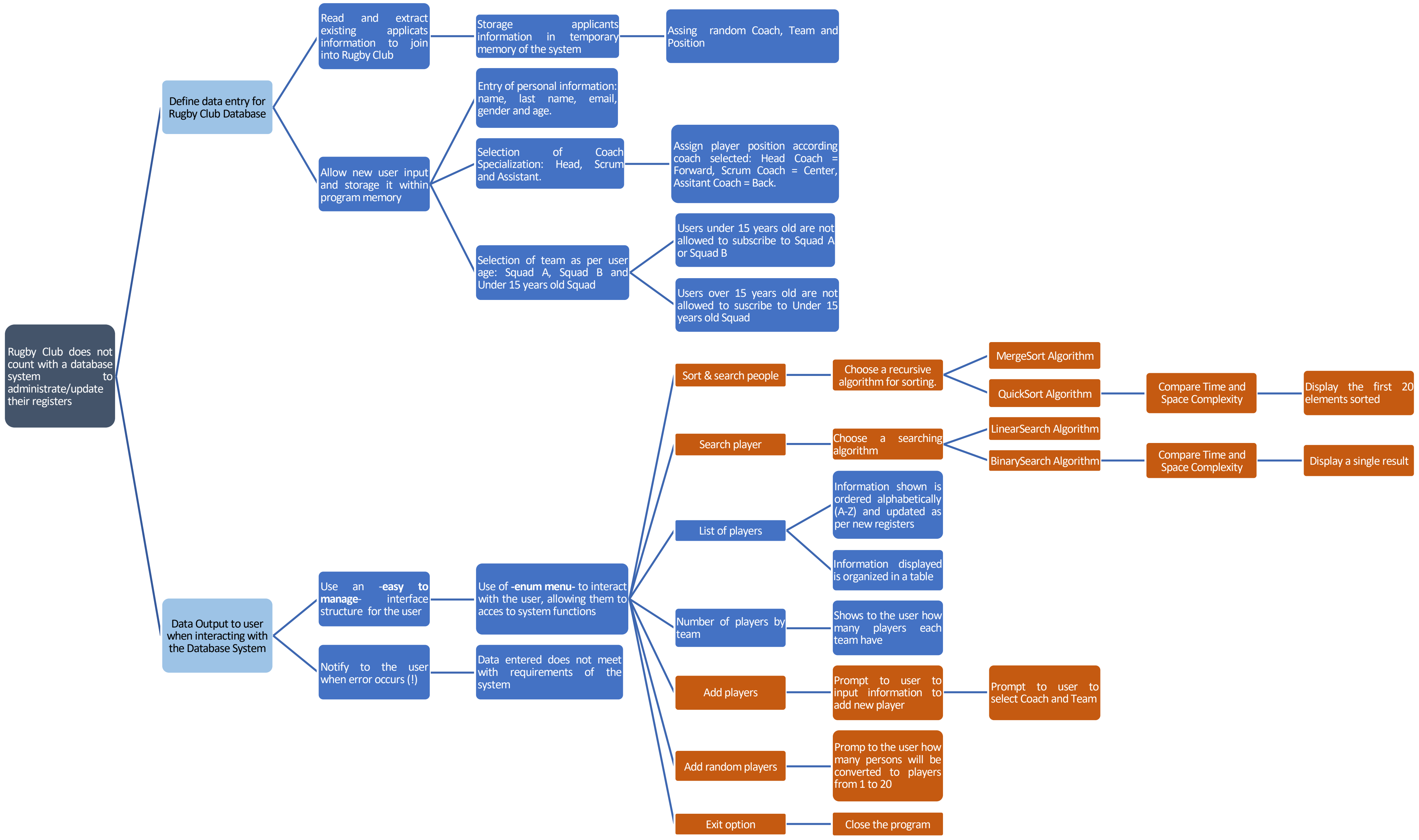

Rugby Club does not count with a database system to administrate/update their registers
Define data entry for Rugby Club Database
Read and extract existing applicats information to join into Rugby Club
Storage applicants information in temporary memory of the system
Assing random Coach, Team and Position
Allow new user input and storage it within program memory
Entry of personal information: name, last name, email, gender and age.
Selection of Coach Specialization: Head, Scrum and Assistant.
Assign player position according coach selected: Head Coach = Forward, Scrum Coach = Center, Assitant Coach = Back.
Selection of team as per user age: Squad A, Squad B and Under 15 years old Squad
Users under 15 years old are not allowed to subscribe to Squad A or Squad B
Users over 15 years old are not allowed to suscribe to Under 15 years old Squad
Data Output to user when interacting with the Database System
Use an -easy to manage- interface structure for the user
Use of -enum menu- to interact with the user, allowing them to acces to system functions
Notify to the user when error occurs (!)
Data entered does not meet with requirements of the system
Sort & search people
Choose a recursive algorithm for sorting.
MergeSort Algorithm
QuickSort Algorithm
Compare Time and Space Complexity
Display the first 20 elements sorted
Search player
Choose a searching algorithm
LinearSearch Algorithm
BinarySearch Algorithm
Compare Time and Space Complexity
Display a single result
List of players
Information shown is ordered alphabetically (A-Z) and updated as per new registers
Information displayed is organized in a table
Number of players by team
Shows to the user how many players each team have
Add players
Prompt to user to input information to add new player
Prompt to user to select Coach and Team
Add random players
Promp to the user how many persons will be converted to players from 1 to 20
Exit option
Close the program

Figure 20 Mapping Diagram for Rugby Club system

PROJECT DEFINITION CARD

Project name:	Rugby Club System

Specific goals: (what are you going to do)	1.Read a file with the applicants of the rugby club. 2.Sort that list of people of applicants and display the first 20 elements. 3.Search for a specific player in the club. 4.Add new players randomly taking applicants from the file and assigning them coach and team. 5.Add new players allowing the user input specific details and choice the coach and team. 6.List all the players inside the rugby club. 7.Display the number of players in each team.

Out of scope: (what are you not going to do)	1. Add new people to the original applicant file (Club_form.txt). 2. Not allow the user create new coaches and new teams. 3. not allow the user remove people.

Timetable:	Actions	Needed time	Cumulative time
	1. Rugby Club does not count with a technological system to administrate/update their registers.		
	2. Rugby Club wants to develop a database that allows administrate and coordinate the 3 teams they have (Squad A, Squad B and Under 15 years old category), creating new team players based in their existing users library as well as add new registers, but also search and edit players from the list. To doing so is necessary to have a system that reads and manipulate information contained in their current database, that is friendly to use and that allows the entry of data by the user to update current database and different functionalities like searching or sort actual information from rugby players. Having as result of this interaction between the user and the -num menu- displayed, allowing the creation and edition of rugby players, and searching information.		

	Define the diagnostic key question and identify possible causes. **Diagnostic Key question**: how a database system for Rugby Club can be developed/designed with an efficient and -easy to use- interface for the user to register/update their current records? **Possible causes:** **-Data structure:** to make consistent how information has to be read and saved into the database. **-System interface:** enum menu with multiple but simple options to interact with the interface of the new Rugby Club database system proposed.		
	Collect the diagnostic evidence, analyse, and draw conclusions.		
	3. Identify solutions (find the how) The database system for Rugby Club was tested several times to identify opportunities of improvements and to empathize with the user with the entry of information.		
	Define the solution key question and identify potential solutions		
	Collect evidence, analyse, and decide which solution(s) to implement		
	4. Implement the solution(s) (do)		

Resources:	Time, mapping, UML, Algorithms

Possible problems:	1.Errors at input from the user 2.Reading the database

Mitigation actions:	Messages for the user to correct insert the data

Focused on our goals, we were able to make the pertinent changes to the system and to our UML diagrams, starting with our class diagram, which ended as follows:

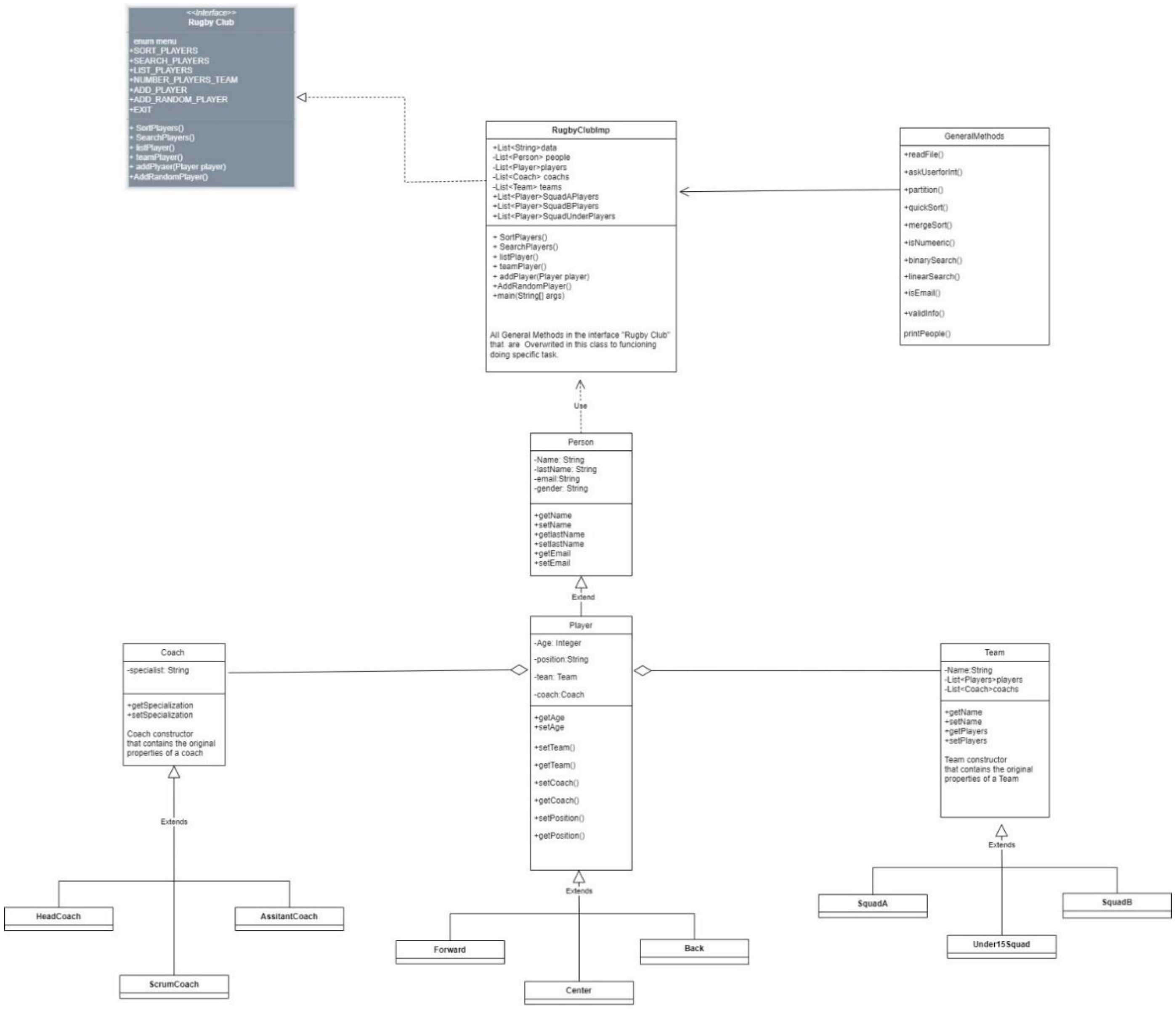


Figure 21 Class diagram

To have a final structural view of how the system turned out and what changes were made in our class structure and data handling, as shown in figure 2, the way in which inheritances and encapsulation were being handled in our system was not the best, since the way to extract the data was correct. To optimize our system, the attributes that were in our class types that extend to our Super Classes, should belong only in the Super Classes, in this way we avoid redundancies when performing the system(UML class diagrams, 2023).

We then updated our UML diagram of user cases to see a full high-level interaction between the club Manager and the system.

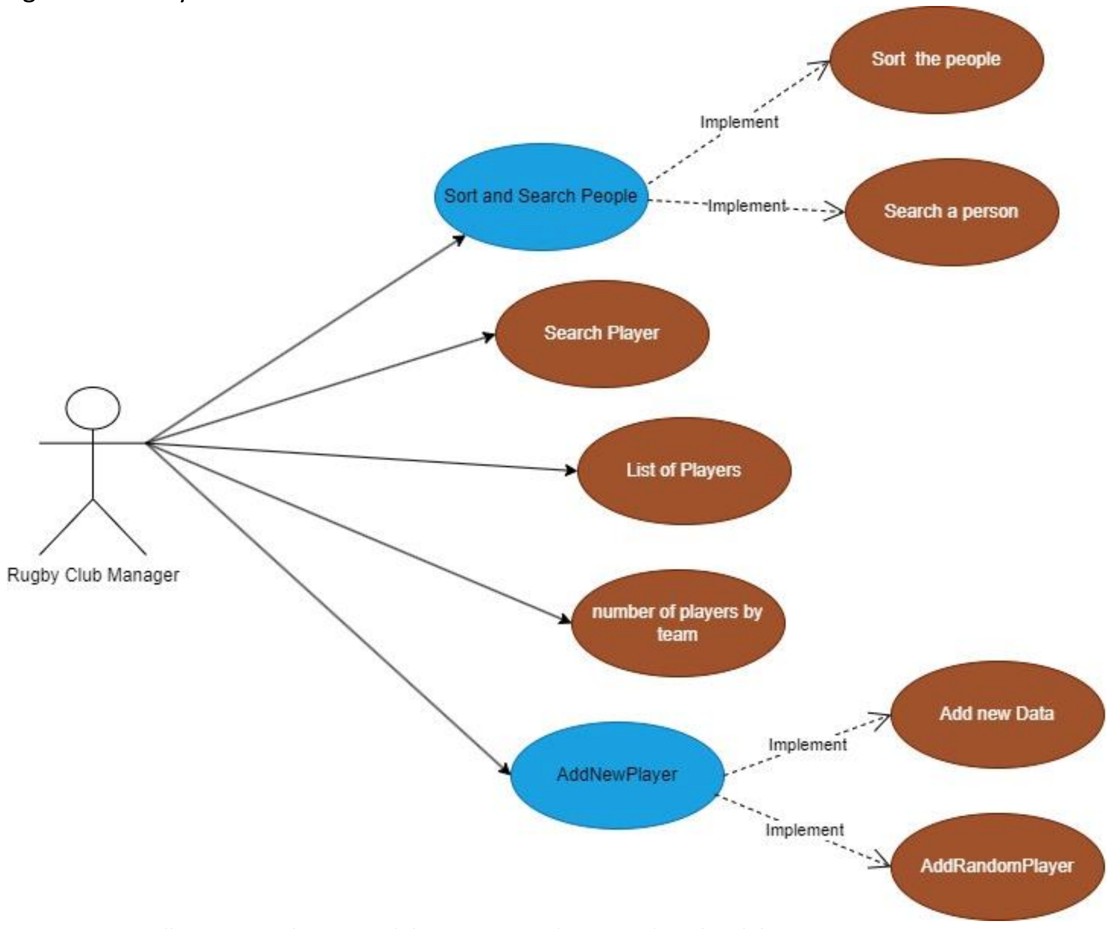


Figure 22 Full interaction between Club Manager and proposed Rugby Club System

And at the same time, we can go deeply of what the flow of the system is like from the moment an applicant arrives to the end of the system with the help of our updated sequence UML.

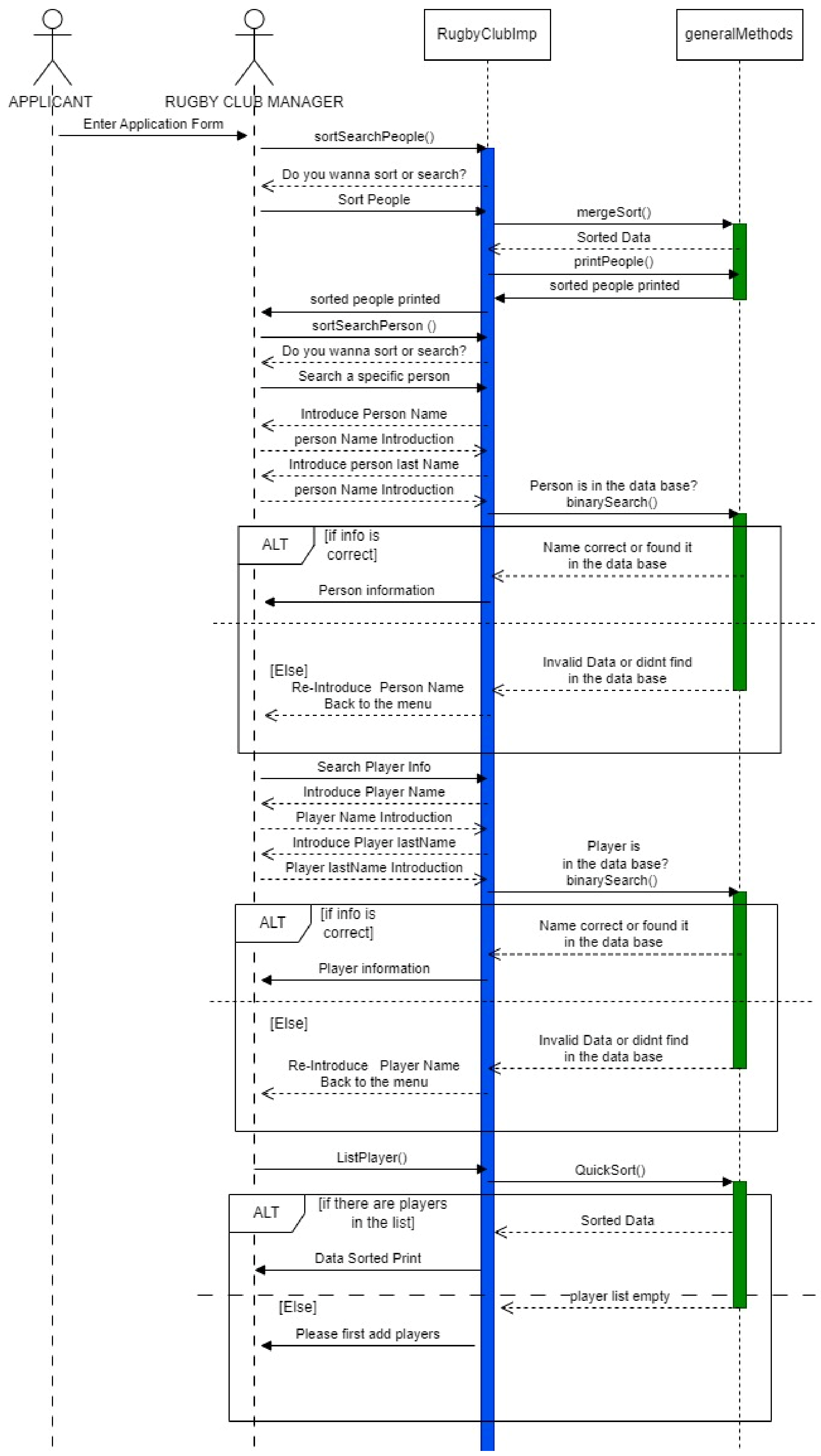


Figure 23 Sequence UML Part I

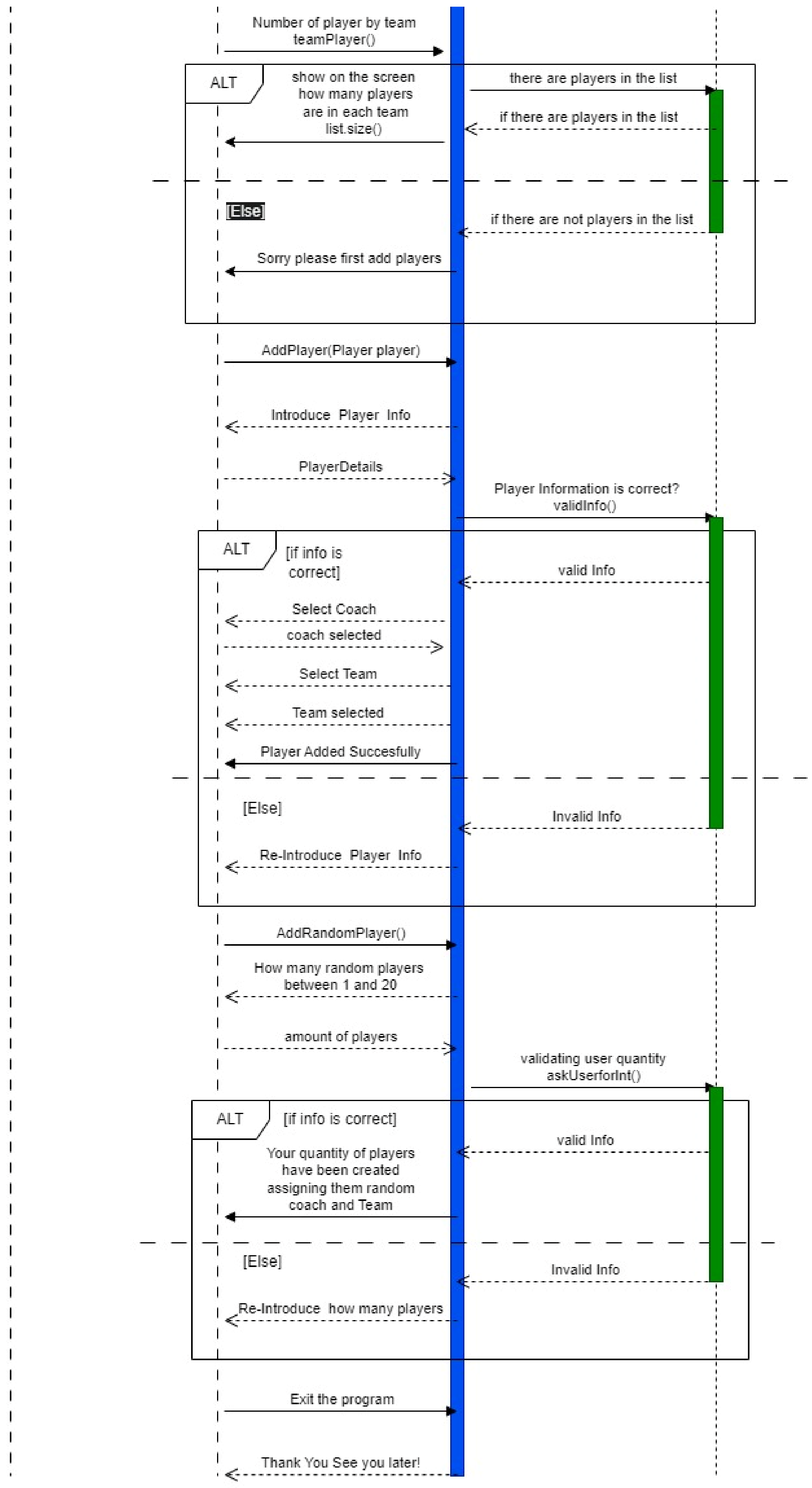

Number of player by team
teamPlayer()
ALT
show on the screen how many players are in each team list.size()
there are players in the list
if there are players in the list
[Else]
if there are not players in the list
Sorry please first add players
AddPlayer(Player player)
Introduce Player Info
PlayerDetails
Player Information is correct? validInfo()
ALT
[if info is correct]
valid Info
Select Coach
coach selected
Select Team
Team selected
Player Added Succesfully
[Else]
Invalid Info
Re-Introduce Player Info
AddRandomPlayer()
How many random players between 1 and 20
amount of players
validating user quantity askUserforInt()
ALT
[if info is correct]
valid Info
Your quantity of players have been created assigning them random coach and Team
[Else]
Invalid Info
Re-Introduce how many players
Exit the program
Thank You See you later!

Figure 25 Sequence UML Part II

USER CASE

With the correct diagrams and the system finished correctly according to what is required, we can move on to the testing part and the acceptance criteria on the User Cases, with this to mitigate the lack of errors that may occur in the system.

1. User Case

Title: Viewing information of Sorted list of rugby club applicants

Actor	Rugby Club Manager
System	1. Manager opens the rugby club software
	2. Enters the name of the file with the applicant's data.
	3. select from the menu the option of sort and search the applicants.
	4.Select from the menu the option of sort people
	5.Display of 20 first applicants sorted in the list
Exception	1.Error at introducing the name of the file
	2.There are no applicants in the file
Goal	View information of the sorted list of applicants

User story:

As a Rugby Club Manager, I want to access and view sorted the list of applicants that want be part of the rugby club so I can see who has interest on been part of.

Acceptance criteria:

- The Rugby Club Manager should be able to write the file with the applicant's information
- The system can validate if the file is there
- Information from the form is stored in the database of the system
- Functional menu where the Manager can select sort and searching people
- Functional menu where the Manager can select sort people
- Display the first 20 applicants of the list already sorted
- Error handling scenarios with appropriate error messages

2. User case

Title: Searching a specific Person information in the rugby club

Actor	Rugby Club Manager
System	1. Manager opens the rugby club software
	2. Enters the name of the file with the applicant's data.
	3. Select from the menu the option of sort and people.
	4. Select from the menu the option of search a specific person.
	5. Introduce the name of the person
	6. Introduce the last name of the person
	7. Display of the players detailed information.
Exception	1. Error at introducing the name of the file.
	2. Person not found in the database.
Goal	Get information of the searched person

User story:

As a Rugby Club Manager, I want to search a specific person that is an applicant to the rugby club so I can see their information and can contact them or assigned them to a team.

Acceptance criteria:

- The Rugby Club Manager should be able to write the file with the applicant’s information
- The system can validate if the file is there
- Information from the form is stored in the database of the system
- Functional menu where the Manager can select sort and searching people
- Functional menu where the Manager can select search for specific person
- Display the information of the correct person
- Error handling scenarios with appropriate error messages

3. **Title:** Searching a specific player information in the rugby club

Actor	**Rugby Club Manager**
System	1. Manager opens the rugby club software
	2. Enters the name of the file with the applicant’s data.
	3. select from the menu the option of search a specific player.
	4.Introduce the name of the player
	5.Introduce the last name of the player
	6.display of the players detailed information.
Exception	1.Error at introducing the name of the file.
	2. Player not found in the database.
Goal	Get information of the searched player

User story:

As a Rugby Club Manager, I want to search a specific player that is in our rugby club so I can see their information including the position, couch and team of the player.

Acceptance criteria:

- The Rugby Club Manager should be able to write the file with the applicant’s information
- The system can validate if the file is there
- Information from the form is stored in the database of the system
- Functional menu where the Manager can select search for specific Player
- The Rugby Club Manager can introduce the name and last name of the player
- Display the information of the correct player
- Error handling scenarios with appropriate error messages.

5. **Title:** Viewing list of all players in the rugby club.

Actor	Rugby Club Manager
System	1. Manager opens the rugby club software
	2. Enters the name of the file with the applicant's data.
	3. select from the menu the option of list the players in the club
	4.Display the players in the club with their detailed information
Exception	1.Error at introducing the name of the file
	2.There are no players in the rugby club
Goal	View the list of all players in the rugby club

User story:

As a Rugby Club Manager, I want to get a list of all the players that are in the rugby club so I can know their specific details and where their belongs to.

Acceptance criteria:

- The Rugby Club Manager should be able to write the file with the applicant's information
- The system can validate if the file is there
- Information from the form is stored in the database of the system
- Functional menu where the Manager can select list of players
- Display the list with the information of all the players in the system
- Error handling scenarios with appropriate error messages.

6. **Title:** Getting the numbers of players in each team

Actor	Rugby Club Manager
System	1. Manager opens the rugby club software
	2. Enters the name of the file with the applicant's data.
	3. select from the menu the option of number of players per team
	4.Display the number of players in each team
Exception	1.Error at introducing the name of the file
	2.There are no players in the rugby club
Goal	View number of all players in the rugby club in each team

User story:

As a Rugby Club Manager, I want to get the number of players in each team, so I can have a record and see where I can assign the new players or the spots that we have taken.

User criteria:

- The Rugby Club Manager should be able to write the file with the applicant's information
- The system can validate if the file is there
- Information from the form is stored in the database of the system
- Functional menu where the Manager can select the option of Number of players by team
- Display the number of players by team

- Error handling scenarios with appropriate error messages.

7. **Title:** Random assignment of coaches and teams to applicants

Actor	Rugby Club Manager
System	1. Manager opens the rugby club software
	2. Enters the name of the file with the applicant's data.
	3. select from the menu the option of add random players
	4.Display to let the user get between 1 and 20 new random players
	5.Display the list of the new random players added
Exception	1.Error at introducing the name of the file
	2.Invalid input data of how many players
Goal	Assign coaches and team to create new random players in the club and list all of them.

User story:
As a Rugby Club Manager, I want to assign randomly coaches and teams to the applicants, so I can rapidly assign to the applicants their role in the club, due to some times are lots of applicants.

Acceptance criteria:

- The Rugby Club Manager should be able to write the file with the applicant's information
- The system can validate if the file is there
- Information from the form is stored in the database of the system
- Functional menu where the Manager can select the option "add random players"
- Display the list of the people and their information assigned (new players with coach and team)
- Error handling scenarios with appropriate error messages.

8. **Title:** Adding new players with a coach and team

Actor	Rugby Club Manager
System	1. Manager opens the rugby club software
	2. Enters the name of the file with the applicant's data.
	3. select from the menu the option of add a player.
	4.Display a prompt that will ask for the players details.
	5.Display a prompt that will let choice the coach and set the position of the player
	6.Display a prompt that will let choice the team
	7.Add the player and display the coach and team that belong to.
Exception	1.Error at introducing the name of the file.
	2.Invalid input data for the player details.
	3.Player already exist in the data base.
	4.Invalid selection of a coach.
	5.Invalid selection of a team.
Goal	Add the player and shows the coach and team that have been assigned.

User story:
As a Rugby Club Manager, I want to add new players details and assign them a coach and team so I can register individual requests.

Acceptance criteria:

- The Rugby Club Manager should be able to write the file with the applicant’s information
- The system can validate if the file is there
- Information from the form is stored in the database of the system
- Functional menu where the Manager can select the option “add new players”
- Display adequate prompts to ask for the new player details
- Display adequate menus to let the Manager choose the coach
- Display adequate menus to let the Manager choose the team
- System correctly adds the player where it belongs
- Display the player added with their coach and team assigned
- Error handling scenarios with appropriate error messages

CONCLUSION

To conclude with the documentation of this prototype system that was created for Rugby Club, we can say that one of the main problems we faced was the structure of our database, the relationships between classes and their inheritances were handled, but with the tools used in the development such as the different types of UML, and problem maps, most of these could be solved or mitigated, resulting in a more efficient and aligned database according to the main functions required.

Another of the main problems was the choice of the different algorithms that were implemented in the different functions of the system, but thanks to the different tests carried out and performance monitoring through visualVM, Merge Sort and Binary Search could be chosen as the two best options so that the system has the best optimization and behaves with the best possible performance.

Together, these variables enabled an easy-to-use and reliable interface to the rugby club data management system to meet the club's needs.

COMPUTER AND SOFTWARE ASPECTS

COMPUTER MODEL NAME: ASUS – TUF GAMING A15

PROCESSOR: AMD Ryzen 7 5800H with Radeon Graphics, 3201 Mhz, 8 Core(s), 16 Logical Processor(s)

RAM: 32 GB

SYSTEM TYPE: x64-based pc

OS NAME: Microsoft Windows 11 Home

Version 10.0.22621 Build 22621

Software: Apache NetBeans IDE 15

JAVA FILES UPLOADED FOR THIS SOFTWARE:

RugbyClub.java

RugbyClubImp.java

generalMethods.java

Person.java

Player.java

Forward.java

Center.java

Back.java

Coach.java

HeadCoach.java

AssistanCoach.java

ScrumCoach.java

Team.java

SquadA.java

SquadB.java

Under15Squad.java

All of this were created in CA_2 package.

REFERENCE

3 School Team, 2023. *Java Encapsulation and Getters and Setters*. [online] Available at: <https://www.w3schools.com/java/java_encapsulation.asp> [Accessed 6 November 2023].

3School Team, 2023a. *Java Constructors*. [online] Available at: <https://www.w3schools.com/java/java_constructors.asp> [Accessed 6 November 2023].

3School Team, 2023b. *Java Inheritance (Subclass and Superclass)*. [online] Available at: <https://www.w3schools.com/java/java_inheritance.asp> [Accessed 6 November 2023].

3School Team, 2023c. *Java Modifiers*. [online] Available at: <https://www.w3schools.com/java/java_modifiers.asp> [Accessed 7 November 2023].

admin, 2015. Validar un email en Java. *ᐅ Cursos de Programación de 0 a Experto © Garantizados*. Available at: <https://unipython.com/validar-un-email-en-java/> [Accessed 4 November 2023].

All UML Diagrams in 10 minutes. 2021. Directed by Ave Coders. Available at: <https://www.youtube.com/watch?v=hF4yg1yFrdU> [Accessed 25 October 2023].

Anon. 2023. *VisualVM: Getting Started*. [online] Available at: <https://visualvm.github.io/gettingstarted.html> [Accessed 17 November 2023].

baeldung, 2016. *The Basics of Java Generics | Baeldung*. [online] Available at: <https://www.baeldung.com/java-generics> [Accessed 11 November 2023].

Cómo hacer un Diagrama de Secuencias UML. 2018. Directed by Havian. Available at: <https://www.youtube.com/watch?v=pCK6prSq8aw> [Accessed 26 October 2023].

Geek for Geek Team, 2013. Java Program for Merge Sort. *GeeksforGeeks*. Available at: <https://www.geeksforgeeks.org/java-program-for-merge-sort/> [Accessed 17 November 2023].

Georgia Tech, n.d. Collections Algorithms.

Gupta, L., 2014. *Java Email Validation using Regex*. [online] HowToDoInJava. Available at: <https://howtodoinjava.com/java/regex/java-regex-validate-email-address/> [Accessed 15 November 2023].

Java Oracle Team, 2023. *Predicate (Java Platform SE 8)*. [online] Available at: <https://docs.oracle.com/javase/8/docs/api/java/util/function/Predicate.html> [Accessed 11 November 2023].

Lucid chart Team, 2023. *UML Sequence Diagram Tutorial*. [online] Lucidchart. Available at: <https://www.lucidchart.com/pages/uml-sequence-diagram> [Accessed 13 November 2023].

Nisansala, C., 2021. UML Class Diagrams. *Medium*. Available at: <https://chathushinisansala96.medium.com/uml-class-diagrams-db86533de854> [Accessed 26 October 2023].

Oracle Team, 2023. *List (Java Platform SE 8)*. [online] Available at: <https://docs.oracle.com/javase/8/docs/api/java/util/List.html> [Accessed 11 November 2023].

Perry, K., 2022. Find Duplicates Using a HashSet in Java. *Strategio*. Available at: <https://medium.com/strategio/find-duplicates-using-a-hashset-88c4cee12f96> [Accessed 18 November 2023].

Rumyancev, P., 2021. UML Class Diagram Arrows Guide. *Medium*. Available at: <https://paulrumyancev.medium.com/uml-class-diagram-arrows-guide-37e4b1bb11e> [Accessed 26 October 2023].

Sara Sana, 2022. 11 Best Practices For Data Modelling | Saras Analytics. Available at: <https://sarasanalytics.com/blog/data-modeling-best-practices/> [Accessed 13 November 2023].

Software Testing Help Team, 2023. *Merge Sort In Java - Program To Implement MergeSort*. [online] Software Testing Help. Available at: <https://www.softwaretestinghelp.com/merge-sort-java/> [Accessed 17 November 2023].

Staff, B., 2019. *UML Diagrams – Everything You Need to Know to Improve Team Collaboration | Bluescape*. [online] Available at: <https://www.bluescape.com/blog/uml-diagrams-everything-you-need-to-know-to-improve-team-collaboration> [Accessed 23 October 2023].

UML class diagrams. 2023. Directed by Havian. Available at: <https://www.youtube.com/watch?v=6XrL5jXmTwM> [Accessed 27 October 2023].

UML Tutorial: How to Draw UML Class Diagram. 2021. Directed by Wondershare Edraw. Available at: <https://www.youtube.com/watch?v=ao1ESgIy2Ws> [Accessed 23 October 2023].

Visual Paradigms Team, 2023. *UML Class Diagram Tutorial*. [online] Available at: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/> [Accessed 23 October 2023].