A logo for college computing

Description automatically generated

**Assessment Cover Page**

*Higher Diploma in Science in Computing – Cohort September 2023*

|  |  |
| --- | --- |
| *Student Full Name* | Christopher Antonio Anich Escarpentier |
| *Student Number* | 2023202 |
| *Module Title* | Software Development Fundamentals  Project Skills and Professionalism  Algorithms & Constructs |
| *Assessment Title* | Rugby Club System |
| *Assessment Due Date* | 12/05/2024 |
| *Date of Submission* | 13/05/2024 |

**Declaration**

By submitting this assessment, I confirm that I have read the CCT policy on academic misconduct and understand the implications of submitting work that is not my own or does not appropriately reference material taken from a third party or other source.

I declare it to be my own work and that all material from third parties has been appropriately referenced.

I further confirm that this work has not previously been submitted for assessment by myself or someone else in CCT College Dublin or any other higher education institution.

Table of Contents

[**ALGORITHMS AND CONSTRUCTS** 2](#_Toc166472872)

[Why these algorithms in the Rugby Club System program? 2](#_Toc166472873)

[a) Merge Sort Algorithm 2](#_Toc166472874)

[b) Binary Search Algorithm 2](#_Toc166472875)

[**SOFTWARE DEVELOPMENT FUNDAMENTALS** 2](#_Toc166472876)

[1. Use cases for the system: 2](#_Toc166472877)

[a) Individual user use cases 2](#_Toc166472878)

[b) General use case diagram including new features and use cases for the maintenance team. 4](#_Toc166472879)

[2. UML Modelling techniques 5](#_Toc166472880)

[a) Class diagram 5](#_Toc166472881)

[b) Sequence diagram 6](#_Toc166472882)

[c) Justification for the used UML models. 7](#_Toc166472883)

[3. User stories, acceptance criteria and acceptance tests. 7](#_Toc166472884)

[User story 1: 7](#_Toc166472885)

[User story 2: 8](#_Toc166472886)

[User story 3: 8](#_Toc166472887)

[User story 4: 9](#_Toc166472888)

[User story 5: 9](#_Toc166472889)

[**PROJECT SKILLS AND PROFESSIONALISM** 10](#_Toc166472890)

[Introduction 10](#_Toc166472891)

[1. Best practices and solutions in software design and data modelling 10](#_Toc166472892)

[2. Analysing the problems associated with the design and implementation of the Rugby Club Program 10](#_Toc166472893)

[Problems faced 11](#_Toc166472894)

[3. Problem definition and mapping diagram 13](#_Toc166472895)

[**ANNEXES** 15](#_Toc166472896)

[A. Use case cards 15](#_Toc166472897)

[B. Unit test cards 20](#_Toc166472898)

[**Link to repository on GitHub** 23](#_Toc166472899)

[Bibliography 24](#_Toc166472900)

# **ALGORITHMS AND CONSTRUCTS**

# Why these algorithms in the Rugby Club System program?

## Merge Sort Algorithm

This algorithm is defined by a time complexity (Alake, 2022) of O (n log n) for small and large number of inputs, what is considerable good. Its space complexity (Woltmann, 2020) is O (n), what despite of not being the best, is not an issue in our case. While executing the program in an average personal computer with enough capacity the time complexity will stay relatively constant despite of large amount of input. The database for the Rugby Club System is not so large but is large enough to try the program.

## Binary Search Algorithm

The space complexity of this algorithm is (Sharma, 2022) O (1), very good for memory. About its time complexity, this is O (log n), which is way better than the O(n) of the linear one. It was chosen because of its efficiency. It is important to mention that this algorithm requires an already sorted algorithm to search, hence, a sorting operation must be carried out before this one. If the searching operation will be executed only a few times, we have to add the time complexity of the merge sort algorithm, but if the search will be often, the price for the sorting operation is worth it.

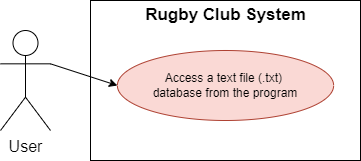
# **SOFTWARE DEVELOPMENT FUNDAMENTALS**

# Use cases for the system:

## Individual user use cases

Every use case for the system is followed by an individual use case diagram. At the end of this section an assembled use case diagram is displayed with some features that are desired to be implemented in the future. At the end of the report in the annexed data section, the use case documentation and details for each of the following is attached.

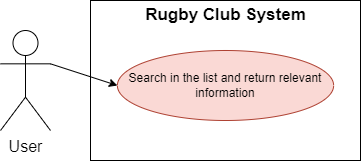
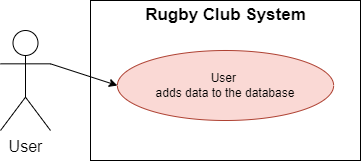
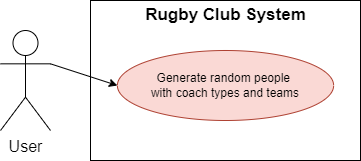
1. The user accesses a text file (.txt) database from the program.



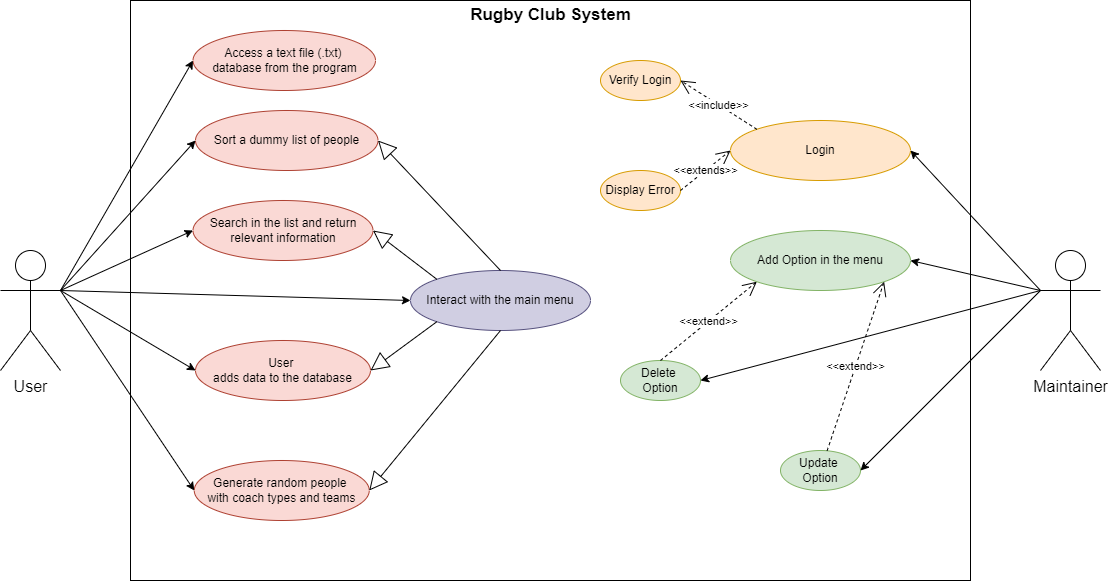
1. The user selects to sort alphabetically by first name the list of people provided.

A diagram of a rugby club system

Description automatically generated

1. The user selects to search people from the database by choosing the option on the program menu.
2. The user selects to add people to the database by choosing the option from the program menu.
3. User selects to generate a random person from the data of the provided database text file.

## General use case diagram including new features and use cases for the maintenance team.



# UML Modelling techniques

In this section, two different UML diagrams are presented to represent the software system, class and sequence.

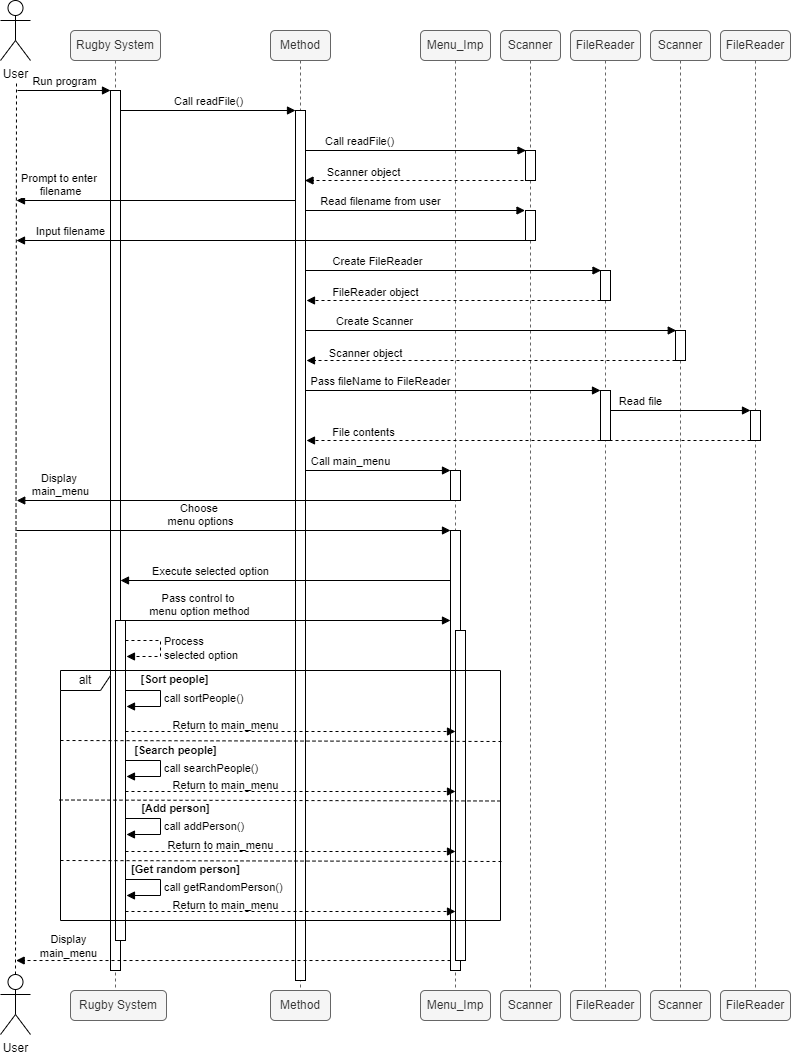
## Class diagram

**It is important to highlight that the “medical staff” class and its Enums do not exist yet and are though as future features to implement in a future version of the system.**

A black screen with white text

Description automatically generated

## Sequence diagram

In this diagram (Bell, 2023) we can walk through the whole software execution and understand the processes and interactions among the user and the different entities involved.

## Justification for the used UML models.

There exist several UML models, but each of them is utilised for different purposes. In this case I chose the class diagram and the sequence diagram model.

1. **Class diagram:** In a program that is made to read and manage data, it is crucial to understand how this is organised due to its feature to model objects (IBM, 2021). The class hierarchy in this program is paramount, considering that we are handling list of peoples of different staff categories and team, which usually manage a lot of people. To be able to understand how the Player and Coach classes inherited the attributes from Person, because both of them belong to this class type.

An alternative to this diagram could have been the user class diagram (Kealy, 2014) which can show us the process of adding a new entry to the database. This could help us to understand what the information is involved during the whole execution of the program, but it is not possible to understand how the Person class is related to the Coach or Player one , or how a team is assigned to a Person.

1. **Sequence diagram:** This model (Bell, 2023) is ideal to understand the flow of data between the user and the system, hence, by applying it to our Rugby Club System software, we are able to navigate through the different options displayed in the menu. It helps to follow how the system interacts with the user and the database to manipulate de information during its functioning.

An alternative to this diagram could have been, the activity diagram, which (Britton & Doake, 2005) shows a general approach about the flow of the software. It helps to understand broadly the idea, but without too much detail on how the user interacts with the different features of the application and how the data is handled to be asked for information and retrieve it.

# User stories, acceptance criteria and acceptance tests.

Five user stories are presented here beside their acceptance criteria and acceptance tests according to bibliography (Scrum Alliance, 2023) and information from class (Healy, 2024).

The respective unit tests are attached to the end of this report.

### User story 1:

As a user, I want to import a text file with people information to the project, so that I can import and manipulate the database from the program.

**Acceptance criteria:**

* The user must have access to the project’s main folder.
* The user must have access to a text file with a database.
* The system must prompt the user to enter the name of the file.
* Upon successful upload, the system should read the file and populate the database.
* If the file is empty or the name and/or file format provided by the user is incorrect, an error message should be displayed.

**Acceptance tests:**

* Test that the user has access to the project’s main folder.
* Test that the user possesses a valid text file with data base including.
* Test the system successfully reads the file.

### User story 2:

As a user, I want to view the information of the first 20 sportsmen arranged alphabetically by first name, so that I can quickly access the details of the initial entries in the list.

**Acceptance criteria:**

* The user must choose the right option on the program menu.
* The system displays the details of 20 people arranged alphabetically in ascending order by first name.
* Each entry includes the person’s ID, first name, last name, email address, gender, team, staff type (player or coach) and subtype of staff.

**Acceptance tests:**

* Test that the user chooses a valid option among the ones on the menu.
* Test that the number of people displayed is 20.
* Test that the entries are displayed sorted alphabetically by first name.
* Test that all the information is displayed properly.

### User story 3:

As a user, I want to search for people in the list by their first name, so that I can find a specific person and view their information.

**Acceptance criteria:**

* The user must choose the correct option on the program menu.
* The user must introduce a first name that is contained in the database.
* The system should display the information of the person or people who have that first name, including the person’s ID, first name, last name, email address, gender, team, staff type (player or coach) and subtype of staff.

**Acceptance tests:**

* Test that the user chooses a valid option among the ones on the menu.
* Test if the system looks for the introduced first name in the database.
* Test if the system displays all the information of everyone with that first name properly.

### User story 4:

As a user, I want to add people to the list by adding every attribute of the person, so that, I can keep a record of new players or coaches.

**Acceptance criteria:**

* The user must choose the correct option on the program menu.
* The system should prompt the user to add every attribute of the person to the database.
* The user must enter appropriate information.
* The system checks if the information is correct.
* The system adds the information as a new entry and displays a success message.

**Acceptance tests:**

* Test that the user chooses a valid option among the ones on the menu.
* Test that the system displays properly the instructions on the console to get the information from the user.
* Test the information provided by the user meets the requirements of the database.
* Test if the system displays an error message when wrong information is entered, or invalid options are chosen.
* Test if the new information was properly added to the database.

### User story 5:

As a user, I want to generate random data and add fictional people to the list, so that, I can populate the database and try it.

**Acceptance criteria:**

* The user must choose the correct option on the program menu.
* The system should display a message confirming the creation of a new random entry and addition of it to the database.
* The system should display the information of the new entry on the console with a coherent ID for the new random person.

**Acceptance tests:**

* Test that the user chooses a valid option among the ones on the menu.
* Test that the system, despite of the confirmation message, has really added the new entry to the database.
* Test that the system displays properly all the information of the new random entry.

# **PROJECT SKILLS AND PROFESSIONALISM**

## Introduction

When designing and implementing a software, in this case, “Rugby Club System” for command line, it is a good practice to concretise the project even if it is in a graphical fashion, to helps us and our team to abstract the ideas and share them. As stated by IBM (IBM, 2024) “data modelling is the process of creating a visual representation of either a whole information system or parts of it to communicate connections between data points and structures”. The goal of data modelling (IBM, 2024) is to illustrate the types of data used and stored within the system, the relationships among these datatypes, the ways the data can be grouped and organised and its formats and attributes.

## Best practices and solutions in software design and data modelling

1. Understand (Vivek, 2024) the business needs and required outcomes by knowing what data to prioritise, gather, store, alter and make accessible to users.
2. Visualise (Vivek, 2024) the data and information to be modelled, hopefully in a graphical way or data illustration (data visualisation).
3. To understand (IBM, 2024) the types of data modelling to be able to choose the one or ones that best suits the needs of the project (i.e. Hierarchical data models, relational data models, ER data models, Object-oriented data models, etc).
4. Respect the data modelling as a discipline and be thoroughly when working on it if it corresponds. This is the common workflow approach (IBM, 2024) :
   1. Identify the entities.
   2. Identify key properties of each entity.
   3. Identify relationships among entities.
   4. Map attributes to entities completely.
   5. Assign keys as needed and decide on a degree of normalisation that balances the need to reduce redundancy with performance requirements. Normalisation is a technique for organising data models in which numerical identifiers called keys, are assigned to groups of data to represent relationships without repeating data.

## Analysing the problems associated with the design and implementation of the Rugby Club Program

It is important to highlight an important point, the project was made and executed individually, hence, several of the practices and uses of data modelling could have been not applicable here. This depends on the team, but for me, the more useful graphs for understanding and creating the program before writing one line of code, where mainly the class followed by the sequence diagram. Moreover, most of the information to prepare this report was obtain from (Garza, 2024)

### Problems faced

1. **Problem 1:**

Considering that in this project there is a database of random people without any relation with rugby teams, the first challenge was to understand even in a basic way, some characteristics of rugby, such as coach and player positions. This entails an investigation about the subject, especially when there is no approach to this sport in advance.

**Solution 1:**

To carry out an investigation about the basis of rugby to be able to organise and manipulate the data provided in a coherent manner and give the user logical and real options to choose from when interacting with the program.

**Strength:**

To learn about rugby, helped all the team to understand the project better and improve its accuracy.

**Weakness:**

Lack of knowledge in rugby or the context of any software requires time of research.

**Alternative approach:**

To have collaborated with rugby experts or coaches for insights.

1. **Problem 2:**

To organise the classes in such a way that it is possible to assign some attributes randomly and others assigned by the user.

If we analyse the provided database in text file, we can find only personal information about several people, organised in CSV format (separated by commas), but there is no association with staff type (coach or player), player and coach type, or team. This brings a challenge when thinking about how to organise the different classes.

**Solution 2:**

After having understood how rugby works, it is important to sketch the different entities of the database with their attributes, parents, children and all related to inheritance. Moreover, when working with classes that required to be catalogued, Enums are a great alternative even when drawing a class diagram, for instance.

**Strength:**

An adequate use of inheritance and enumeration enhanced the organisation of the classes.

**Weakness:**

To work with complex code and classes can difficult the maintenance.

**Alternative approach:**

Could have explored different class organization patterns, such as composition over inheritance.

1. **Problem 3:**

Was difficult to plan if it was better to create the class Team in the same way that the class Person (which is parent of Coach and Player), so the name of the teams could be organised in like the classes of the Coach class for example, or to create a class that in the future could be the centre of the database organisation, considering the importance of the teams in this kind of massive sports.

**Solution 3:**

Team was considered a class that contains as attributes, two array lists, one for players and another for coaches (both children of the Person class). In this way, the array list of all the staff belongs to the Team class.

**Strengths:**

The use of the class Team in this way allows us to scale it in the future with more information for the club and administrative functions.

**Weaknesses:**

To work with complex code and classes can difficult the maintenance.

**Alternative approach:**

We could have considered a database-first approach, defining class structures based on existing data.

1. **Problem 4:**

There was a decision regarding to the searching algorithm to use. The provided database is not sorted alphabetically, so, finding a good algorithm and optimising the software was a challenge. Using a linear search algorithm can be useful but if the searching operation will be frequent, at the end of the day, the program will have runtime problems.

**Solution 4:**

Recycling the sort algorithm of the program to have arrange alphabetically the database before the searching operation is carried out, hence, while in the program, the searching operation will be fluid with an efficient binary search algorithm.

**Strengths:**

Sorting and optimizing the list before the search improved the program efficiency, especially when searching is used often.

**Weaknesses:**

The use of sorting for every search, can increment the time complexity compared to linear search when the option is used seldomly.

**Alternative approach:**

Could have experimented with other search algorithms.

## Problem definition and mapping diagram

There exist several methods of risk analysis (Garza, 2024), but we will focus on PPP (Project, Process and People) classification to find one and work with it. From here we take the third P (people), due to our 4 problems are related with knowledge and skills of our team.

Problem: lack of knowledge or skills enough to create and administer the program.

Likelihood: Impossible to improbable: 0 to 40%.

Impact: Marginal. Chances are that the code runs but is not efficient enough.

Implement the solution:

* **Plan:** training activities for the development team.
* **Do:** start with one department.
* **Check:** review the results by comparing efficiency of the code between the trained team and the one which was not.
* **Act:** If the results are consistent, implement the plan a large scale. Monitor changes as part of the cycle and keep a continuing training.

#### Diagnostic Issue Map



# **ANNEXES**

### Use case cards

**Use Case 1**

|  |  |  |
| --- | --- | --- |
| **Use Case ID** | RAG001 | |
| **Description** (or Overview) | The user accesses a text file (.txt) database from the program | |
| **Dependencies** | Copying the file inside the program (project) main folder | |
| **Precondition(s)** | The user has access to the text document with the database of people  The document is in the main folder of the program (project)  The user is in the program | |
| **Primary Actor** | User | |
| **Main Sequence**  << describes the actions that take place in a ‘normal interaction>> | **Step** | **Action** |
| 1 | User enters the webpage (on Moodle) and finds the “Club\_Form.txt” file |
| 2 | User downloads the “Club\_Form.txt” file to their system |
| 3 | User saves the document in the project (program) main folder |
| 4 | User enters the program |
| 5 | System prompts the user to enter a filename including extension (i.e. .txt) |
| 6 | User writes the name of the database file (.txt) |
| 7 | System reads the file |
| **Postcondition(s)** | System displays a message of success, saying that the file was properly read | |
| **Alternative Paths** (or Exceptions)  <<sets out alternative sequence(s) of actions in unusual situations, such as a user error  If there are none then this section would be omitted >> | **Step** 1 | **Action** |
| a | User does not have access to the webpage which stores the database |
| b | User cannot download the file |
| **Step** 3 | **Action** |
| a | User saves the document in a different folder. |
| b | User enters the program |
| c | System prompts the user to enter a filename including extension (i.e. .txt) |
| d | User writes the name of the database file (.txt) |
| e | The program cannot access the database and prompts the user to try again |
| f | User copies the text file in the mail folder of the program (project) |
| g | User enters the program |
| **Step** 6 | **Action** |
| a | User enters a file name that does not coincide with the name of the file in the project’s main folder or forgets to include the extension (i.e. Rugby\_Club.txt) |
| b | System displays an error message and prompts the user to try again |
| c | User writes the name of the database file correctly |
| **Comments (or Notes)** | The given name of the text file does not have importance as long as it is rewritten in the same way that is written when the program asks for it.  The implementation of this use case is in the Method.java class and is the first method on that document. Its name is “readFile”. | |

**Use Case 2**

|  |  |  |
| --- | --- | --- |
| **Use Case ID** | RUG002 | |
| **Description** (or Overview) | The user selects to sort alphabetically by first name the list of people provided | |
| **Dependencies** | User accesses a text file (.txt) database from the program (RAG001) | |
| **Precondition(s)** | The user has access to the text document with the database of people  The document is in the main folder of the program (project)  The user is in the program | |
| **Primary Actor** | User | |
| **Main Sequence**  << describes the actions that take place in a ‘normal interaction>> | **Step** | **Action** |
| 1 | System prompts the user to enter a filename including extension (i.e. .txt) |
| 2 | User writes the name of the database file (.txt) (see RUG001) |
| 3 | System reads the file |
| 4 | System displays a message of success |
| 5 | System presents five options for the user to choose |
| 6 | User chooses to sort alphabetically by first name |
| **Postcondition(s)** | System displays the data of the first 20 people sorted alphabetically by first name, including the information from the document and team plus staff position (player or coach) with their respective types, which is assigned randomly by the program. | |
| **Alternative Paths** (or Exceptions)  <<sets out alternative sequence(s) of actions in unusual situations, such as a user error  If there are none then this section would be omitted >> | **Step** 3 | **Action** |
| a | System cannot find the file |
| b | System displays an error message and prompts the user to try again |
| c | User writes the name of the database file (.txt) again |
| **Step** 6 | **Action** |
| a | User selects other option |
| b | System display alternatives or results or alternatives according to the chosen option. |
| **Step** 6 | **Action** |
| a | User selects Exit |
| b | System exits the program after saying goodbye |
| **Comments (or Notes)** | Only twenty people data is displayed not to overwhelm the user if the list is too big.  This use case has been implemented in the Menu\_Imp.java file, which displays the options of the menu and calls the method sortPeople from the Method class to execute it. | |

**Use Case 3**

|  |  |  |
| --- | --- | --- |
| **Use Case ID** | RAG003 | |
| **Description** (or Overview) | The user selects to search people from the database by choosing the option on the program menu | |
| **Dependencies** | User accesses a text file (.txt) database from the program (RAG001) | |
| **Precondition(s)** | The user has access to the text document with the database of people  The document is in the main folder of the program (project)  The user is in the program | |
| **Primary Actor** | User | |
| **Main Sequence**  << describes the actions that take place in a ‘normal interaction>> | **Step** | **Action** |
| 1 | System presents five options for the user to choose |
| 2 | User chooses to search people by first name |
| 3 | System prompts the user to enter a valid first name to search a person on the list |
| 4 | User enters a valid first name (the data is on the list of people) |
| **Postcondition(s)** | System displays the data of the searched person, including the information from the document and team plus staff position (player or coach) with their respective types, which is assigned randomly by the program. | |
| **Alternative Paths** (or Exceptions)  <<sets out alternative sequence(s) of actions in unusual situations, such as a user error  If there are none then this section would be omitted >> | **Step** 2 | **Action** |
| a | User selects other option |
| b | System display alternatives or results according to the chosen option. |
| **Step** 2 | **Action** |
| a | User selects Exit |
| b | System exits the program after saying goodbye |
| **Step** 4 | **Action** |
| a | User enters an invalid first name (the name is not on the list of people) |
| b | System displays an error message and prompts to write a valid first name again |
| **Comments (or Notes)** | A valid first name corresponds to any person already included on the database text file, one created randomly or introduced by the user as new data.  The implementation of this use case is in the Method.java class and is the 4th method on that file. Its name is “searchPeople”. | |

**Use Case 4**

|  |  |  |
| --- | --- | --- |
| **Use Case ID** | RAG004 | |
| **Description** (or Overview) | User selects to add people to the database by choosing the option from the program menu | |
| **Dependencies** | User accesses a text file (.txt) database from the program (RAG001) | |
| **Precondition(s)** | User has access to the text document with the database of people  The document is in the main folder of the program (project)  User is in the program | |
| **Primary Actor** | User | |
| **Main Sequence**  << describes the actions that take place in a ‘normal interaction>> | **Step** | **Action** |
| 1 | System presents five options for the user to choose |
| 2 | User chooses “Add a person to the list” |
| 3 | System asks to enter written data in order (first name, last name, email, gender) |
| 4 | User enters valid data |
| 5 | System asks to choose among some alternatives for the new person by entering the corresponding option number (select team, select kind of staff (player or coach) and the type of staff chosen) |
| 6 | User enters valid data |
| **Postcondition(s)** | System displays a message saying that the new person was added to the database and then it prints this information on the console. | |
| **Alternative Paths** (or Exceptions)  <<sets out alternative sequence(s) of actions in unusual situations, such as a user error  If there are none then this section would be omitted >> | **Step** 2 | **Action** |
| A | User selects other option |
| B | System display alternatives or results according to the chosen option. |
| **Step** 2 | **Action** |
| A | User selects Exit |
| B | System exits the program after saying goodbye |
| **Step** 3 | **Action** |
| A | User enters an invalid email (word followed by at, another word, a dot and a last word) |
| B | Systems prints an error and prompts the user to enter a valid email |
| C | User enters a valid email |
| **Step** 5 | **Action** |
| A | User enters an option not listed on the menu |
| B | System prints an error message and asks the user to choose a valid option |
| C | User enters a valid option |
| **Comments (or Notes)** | In the alternative path for the step 8, only the email is considered. The other attributes accept any values because of inclusion purposes.  In the step 10 of the main sequence, the options are numeric, and the choice must be among the given alternatives.  The implementation of this use case is in the Method.java class and is the third method on that document. Its name is “addPerson”. | |

**Use Case 5**

|  |  |  |
| --- | --- | --- |
| **Use Case ID** | RAG005 | |
| **Description** (or Overview) | User selects to generate a random person from the data of the provided database text file | |
| **Dependencies** | User accesses a text file (.txt) database from the program (RAG001) | |
| **Precondition(s)** | User has access to the text document with the database of people  The document is in the main folder of the program (project)  User is in the program | |
| **Primary Actor** | User | |
| **Main Sequence**  << describes the actions that take place in a ‘normal interaction>> | **Step** | **Action** |
| 1 | System presents five options for the user to choose (see RUG001) |
| 2 | User chooses “Generate a random person” |
| **Postcondition(s)** | System generates a random person by taking a random value of every attribute of the list, plus adding a random team and team position and prints their information on the console | |
| **Alternative Paths** (or Exceptions)  <<sets out alternative sequence(s) of actions in unusual situations, such as a user error  If there are none then this section would be omitted >> | **Step** 2 | **Action** |
| a | User selects other option |
| b | System display alternatives or results according to the chosen option. |
| **Step** 2 | **Action** |
| a | User selects Exit |
| b | System exits the program after saying goodbye |
| **Comments (or Notes)** | The random person is generated by taking a random element from every index of the people array list and creates a new entry, which the system stores as a new person.  The implementation of this use case is in the Method.java class and is the 8th method on that file. Its name is “getRandomPerson”. | |

### Unit test cards

**Unit test 1**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **TEST ID NUMBER** | **1** | **Developer:** | **Christopher Anich** | **Date Test Carried Out:** | **05/05/2024** | **Test Name:** | **Accessing database** | |
| **Module Tested** | Reading a file | | |  | | | | |
| **Description of Test** | Accessing the data from a text file from a program | | | **Test Carried out by:** | **Christopher Anich** |  | | |
| **Test Precondition(s)** | A database file (.txt) with the information has been added to the main folder of the project | | | | | | | |
| **Dependencies (if any)** |  | | | | | | | |
| **TEST STEP** | **DATA (Input)** | | **Expected Result(s)** | | **Actual Result(s)** | | **PASS/FAIL** | **NOTES** |
| File is in main folder and valid file name is written | Club\_Form.txt | | True | | True | | Pass |  |
| File name without extension | Club\_Form | | False | | False | | Fail |  |
| Non-sense | aklfjosaijf | | False | | False | | Pass |  |
| File is not in main folder and valid file name is written | Club\_Form.txt | | False | | False | | Pass |  |

**Unit test 2**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **TEST ID NUMBER** | **2** | **Developer:** | **Christopher Anich** | **Date Test Carried Out:** | **05/05/2024** | **Test Name:** | **Sorting alphabetically** | |
| **Module Tested** | Merge sort algorithm | | |  | | | | |
| **Description of Test** | Sorting data alphabetically | | | **Test Carried out by:** | **Christopher Anich** |  | | |
| **Test Precondition(s)** | The user is in the program and the system has been able to read a text file from its main folder | | | | | | | |
| **Dependencies (if any)** |  | | | | | | | |
| **TEST STEP** | **DATA (Input)** | | **Expected Result(s)** | | **Actual Result(s)** | | **PASS/FAIL** | **NOTES** |
| Valid file name | Club\_Form.txt | | True | | True | | Pass |  |
| Right option is chosen from the menu | 1 | | True | | True | | Pass |  |
| A valid but wrong option is chosen from the menu | 3 | | False | | False | | Pass |  |
| A random character is written as option | A | | False | | False | | Pass |  |

**Unit test 3**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **TEST ID NUMBER** | **3** | **Developer:** | **Christopher Anich** | **Date Test Carried Out:** | **05/05/2024** | **Test Name:** | Search a person and get their data | |
| **Module Tested** | Merge sort algorithm | | |  | | | | |
| **Description of Test** | Search | | | **Test Carried out by:** | **Christopher Anich** |  | | |
| **Test Precondition(s)** | The user is in the program and the system has been able to read a text file from its main folder | | | | | | | |
| **Dependencies (if any)** |  | | | | | | | |
| **TEST STEP** | **DATA (Input)** | | **Expected Result(s)** | | **Actual Result(s)** | | **PASS/FAIL** | **NOTES** |
| Valid file name | Club\_Form.txt | | True | | True | | Pass |  |
| Right option is chosen from the menu | 1 | | True | | True | | Pass |  |
| A valid but wrong option is chosen from the menu | 3 | | False | | False | | Pass |  |
| A random character is written as option | A | | False | | False | | Pass |  |

**Unit test 4**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **TEST ID NUMBER** | **4** | **Developer:** | **Christopher Anich** | **Date Test Carried Out:** | **05/05/2024** | **Test Name:** | Add new person (entry) to the database | |
| **Module Tested** | Add People | | |  | | | | |
| **Description of Test** | Add a new entry to the database from the user input | | | **Test Carried out by:** | **Christopher Anich** |  | | |
| **Test Precondition(s)** | The user is in the program and the system has access to a text file from the project’s main folder | | | | | | | |
| **Dependencies (if any)** |  | | | | | | | |
| **TEST STEP** | **DATA (Input)** | | **Expected Result(s)** | | **Actual Result(s)** | | **PASS/FAIL** | **NOTES** |
| Valid file name | Club\_Form.txt | | True | | True | | Pass |  |
| Right option is chosen from the menu | 3 | | True | | True | | Pass |  |
| A valid but wrong option is chosen from the menu | 1 | | False | | False | | Pass |  |
| A first name is written | Chris | | True | | True | | Pass |  |
| A last name is written | Anich | | True | | True | | Pass |  |
| A wrong mail is written | kjdsfasjdfh | | False | | False | | Pass |  |
| An invalid option from the menu when choosing team | 7 | | False | | Fase | | Pass |  |

**Unit test 5**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **TEST ID NUMBER** | **5** | **Developer:** | **Christopher Anich** | **Date Test Carried Out:** | **05/05/2024** | **Test Name:** | Random person | |
| **Module Tested** | Generate a random person | | |  | | | | |
| **Description of Test** | Generate a random person with data from the database and add it as a new entry. | | | **Test Carried out by:** | **Christopher Anich** |  | | |
| **Test Precondition(s)** | The user is in the program and the system has access to a text file from the project’s main folder | | | | | | | |
| **Dependencies (if any)** |  | | | | | | | |
| **TEST STEP** | **DATA (Input)** | | **Expected Result(s)** | | **Actual Result(s)** | | **PASS/FAIL** | **NOTES** |
| Valid file name | Club\_Form.txt | | True | | True | | Pass |  |
| Right option is chosen from the menu | 4 | | True | | True | | Pass |  |
| A valid but wrong option is chosen from the menu | 3 | | False | | False | | Pass |  |
| Searching by first name the new random person (entry) in the list name (Test unit 3) | Random first name | | True | | True | | Pass |  |

**Link to repository on GitHub:**

<https://github.com/CCT-Dublin/ca-2-integrated-chrisanich>

# Bibliography

Alake, R., 2022. *Merge Sort Explained: A Data Scientist’s Algorithm Guide.* [Online]   
Available at: https://developer.nvidia.com/blog/merge-sort-explained-a-data-scientists-algorithm-guide/

Bell, D., 2023. *Explore the UML sequence diagram (IBM).* [Online]   
Available at: https://developer.ibm.com/articles/the-sequence-diagram/  
[Accessed 2024].

Britton, C. & Doake, J., 2005. *Activity Diagram (Science Direct).* [Online]   
Available at: https://www.sciencedirect.com/topics/computer-science/activity-diagram  
[Accessed 2024].

Garza, J., 2024. *Complex Problem Solving (ppt class CCT College),* s.l.: s.n.

Healy, K., 2024. *User Stories & Acceptance CriteriaUse (class in CCT College),* Dublin: s.n.

IBM, 2021. *Class diagrams.* [Online]   
Available at: https://www.ibm.com/docs/en/rsm/7.5.0?topic=structure-class-diagrams  
[Accessed 2024].

IBM, 2024. *What is data modeling?.* [Online]   
Available at: https://www.ibm.com/topics/data-modeling

IBM, 2024. *What is data modeling?.* [Online]   
Available at: https://www.ibm.com/topics/data-modeling  
[Accessed 2024].

Kealy, K., 2014. *Use Cases & Use Case Diagrams (class in CCT College),* Dublin: s.n.

Scrum Alliance, 2023. *Everything You Need to Know About Acceptance Criteria.* [Online]   
Available at: https://resources.scrumalliance.org/Article/need-know-acceptance-criteria  
[Accessed 2024].

Sharma, R., 2022. *Linear Search vs Binary Search: Difference Between Linear Search & Binary Search.* [Online]   
Available at: https://www.upgrad.com/blog/linear-search-vs-binary-search/

Vivek, J., 2024. *Zuci - Activate Future.* [Online]   
Available at: https://www.zucisystems.com/blog/data-modeling-tips-and-best-practices/  
[Accessed 2024].

Woltmann, S., 2020. *Merge Sort – Algorithm, Source Code, Time Complexity.* [Online]   
Available at: https://www.happycoders.eu/algorithms/merge-sort/#Space\_Complexity\_of\_Merge\_Sort