SIMPLY RUGBY

ACTION PLAN REPORT

ERIK MCSEVENEY | Graded Unit 2 – March 2023

Table of Contents

PROJECT ANALYSIS

PROJECT PRESENTATION

INTRODUCTION

Simply Rugby, the client for this project, is a sport club and member of the Scottish Rugby Union. For the moment, the club uses a paper system maintained by the coaches, without any set standard or communications between them.
As such, Simply Rugby wishes to implement a computerised system that will help gather, standardise, and process information of the player base. They would like to store details about the non-player base.
The main aim of the solution is to help them make informed decisions about squads and players trainings, to further improve individuals based on their performances, and help them develop their skills for the different roles in the squad.
The system will be based around the following main areas:

- Squad composition and the players details.
- Training profiles to track and assess player's development.
- Record of the games played and their details.
- Record the training sessions and add them to a player's profile.

The solution is also aiming at keeping records of non-playing members and their relation to the squads.
In a real-world project, the requirement analysis would be based on a range of client meetings that would be documented and kept, questionnaires to the stakeholders or documents provided by the club.
For the Graded Unit, this takes the form of the Project brief (Appendix A), a formal additional Client feedback received after an invitation for interview with the chairman (Appendix B) and a questionnaire filled by the club (Appendix C). Further research is also done to define the scope and information needed about rugby in relation to the aims of the project.

SIMPLY RUGBY CLUB: BACKGROUND

The following information is based on the initial project documents mentioned in introduction. This is kept at a high level as most of it will be further detailed and explained in the Design documentation, especially in the database design section.

The membership of the club is divided between playing members and non-playing members. Non-playing members includes people like the admin team, the coaches, secretaries and chairmen.
The club has different squads of players supporting all the age grades, as defined in the AGLV, Age Grade Law Variations document, from the SRU.
To keep their administration simple, the club has regrouped those age grades in 3 categories: Mini, Junior and Senior.
The age grades are divided into these 3 categories as follow:

- Mini: Age grade U6 to U11 included
- Junior: Age grade U12 to U15 included
- Senior: Age grade U16 and beyond.

More detailed information on the official Age Grade groups defined by the SRU and World Rugby is available in Appendix D.
The club manages its player base in units called Squads. Each squad is categorised in one of the three age groups. The squads are organised based on the position/role of the players in a game for their age range.
Squads are managed by a group of coaches who usually are parents of players. The client confirmed that coaches oversee only one squad. Those coaches will be referred as the Coach team from now in this document.
There is also a small Admin team attached to a squad, comprised of a chairman and a fixture (a rugby match) secretary.
The club owns and use the following facilities:

- A field for games and training.
- A Fitness suite for specialised training.
- A Café for members to use.

CLIENT REQUIREMENTS

Defining the requirements of a project is the initial and one of the most important steps in software development. It is usually considered critical that they are captured as exhaustively as possible.
They will drive how the solution will be designed, developed and tested. The requirements usually include the features, the specifications and characteristics of the system developed.
Requirements are broken down in categories based on their nature:

- Functional requirements: those describe what the proposed software should do. Usually defined by the client expectations.
- Non-functional requirements: they described the ‘qualities’ of the solution, how it should look, present itself. But also cover concepts like reliability, scalability (how the solution can handle growth), security and performance.

- Legal requirements: some activities or requirements are also governed by local legal frameworks that must be included in the design and development.
- Constraints: other constraints can be part of the requirements, like time constraints.

For the project, the requirements were drawn from the initial brief (Appendix A) and by interviewing the client for more information about their expectations (Documented in Appendices B & C). The main method used was textual analysis by identifying the problems and needs that the solution want to address.
Again, the following is a high-level view of the requirements, the exact details will be presented in the design section. For example, keeping the details of a player is a requirement, the actual data and characteristics of those details would be explained in the Database design section.

FUNCTIONAL REQUIREMENTS

We can group the functional requirements by themes and refine them. Once done, we can then validate them with the client, in a real-world situation, this approval would be documented with an official sign-off which will include the design and project management documents so the client is able to decide if all was covered.

Membership management

The club want to store and manage its playing and non-playing members profiles. Each player record must also include the details of a Next of Kin and the player's doctor.
All players must be registered with the SRU via the SCRUMS system and the information stored with the record.
If the player is under 18, we must also store a consent form provided by the player's parent or guardian.
Non-playing members includes the admin team, chairmen, secretaries, coaches. Doctors and Next of Kin will be categorised as third-party.
The membership is managed by the admin team and the solution must be able to do the following:

- create, update, delete and consult a player record.
- create, update, delete and consult a non- player record.

Games management

The client wants to be able to track the games played, the breakdown of the score by points type, the outcome of the game and the teams involved. The games records will be maintained by the Fixture secretary assigned to the squad, so the person will need to be able to:

- Create and update a game(fixture) record.

Training sessions

The coaches must be able to document and track the training sessions. Each session will take place in one of the club facilities (field, gym, etc.). the coach must be able to:

- Create and consult a training session. A training session will have a type, and a squad assigned to it.

Player performance

The main aim of the solution, as described by the client, is to help to store, consult and track players performance. Coaches keeps performance profiles for their player and this should be duplicated by the proposed solution.

A training profile is composed of the player level in skills required for the rugby. It will also include a log of the games played by the person, how they performed in them.

The client also wishes for the training sessions done by the player to be added to training profile.

As such, a coach needs to be able to:

- Create, delete, consult a training profile.
- Add a game to a training profile.
- Add a training session to the profile.
- Adjust the skills levels of the player.

Out of scope functional requirements

During the interview, the following function were deemed as out of scope as the client do not wish for them to be included in the solution:

- Memberships fees are tracked in another system.
- Coaches' official qualifications do not need to be recorded and future provisions for any legal changes not required.
- The client does not wish to track any other player skills beyond the one agreed during the interviews.

NON-FUNCTIONAL REQUIREMENTS

The non-functional requirements are the qualities that the client wishes the application to have.
From the client briefs and interviews, it appears that there are not at this point any very specific non-functional requirements but we can imply some generic ones that are usually expected in any applications:

- The system should be user-friendly and intuitive as it will be used by people from different backgrounds and computer literacy.
- The system should be able to work on the club computer, or if needed, on an upgraded one. The club is open to a quote if needed.
- The application should be reasonably scalable, for the case of membership growth.
- The application should be secured.
- The solution should be reliable and performs within accepted standards, no apparent bugs or crashes.

CONSTRAINTS

Constraints are a specific kind of requirements that impacts the solution, its quality and can force the developer to add further functions to be compliant.

Some have already been added in the functional requirements: the mandatory registration with the SRU for players and the consent form for players under 18. All players details must include the details of the person's doctor and next of kin details.

The project will use the graded unit deadlines and length as a time constraint. It is assumed that the customer wants the software deployed by the 02/06 as they may need to use the summer to familiarise, further train users or move/create data in the new system.

There is no specific mention or discussions around a cost constraint to this project. We'll assume that the customer is agreeing the proposed cost for the project, including the optional quote for a computer to install and use the software.

There is also a legal constraint that needs to be considered by the client. The data they are gathering for the club administration means that they are responsible to make sure that they are compliant with the GDPR and Data Protection Act 18.
As defined in those regulations, the club is a 'Data Controller' and must ensure that they follow the legislation principles:

- Lawfulness.
- Limited purpose.
- Minimising the amount of data gathered.
- Accuracy.
- Temporarily stored.

- Confidential and safe.

The club should also be able to process requests from the subject (club members) like deletion, update, show a data subject what is stored.

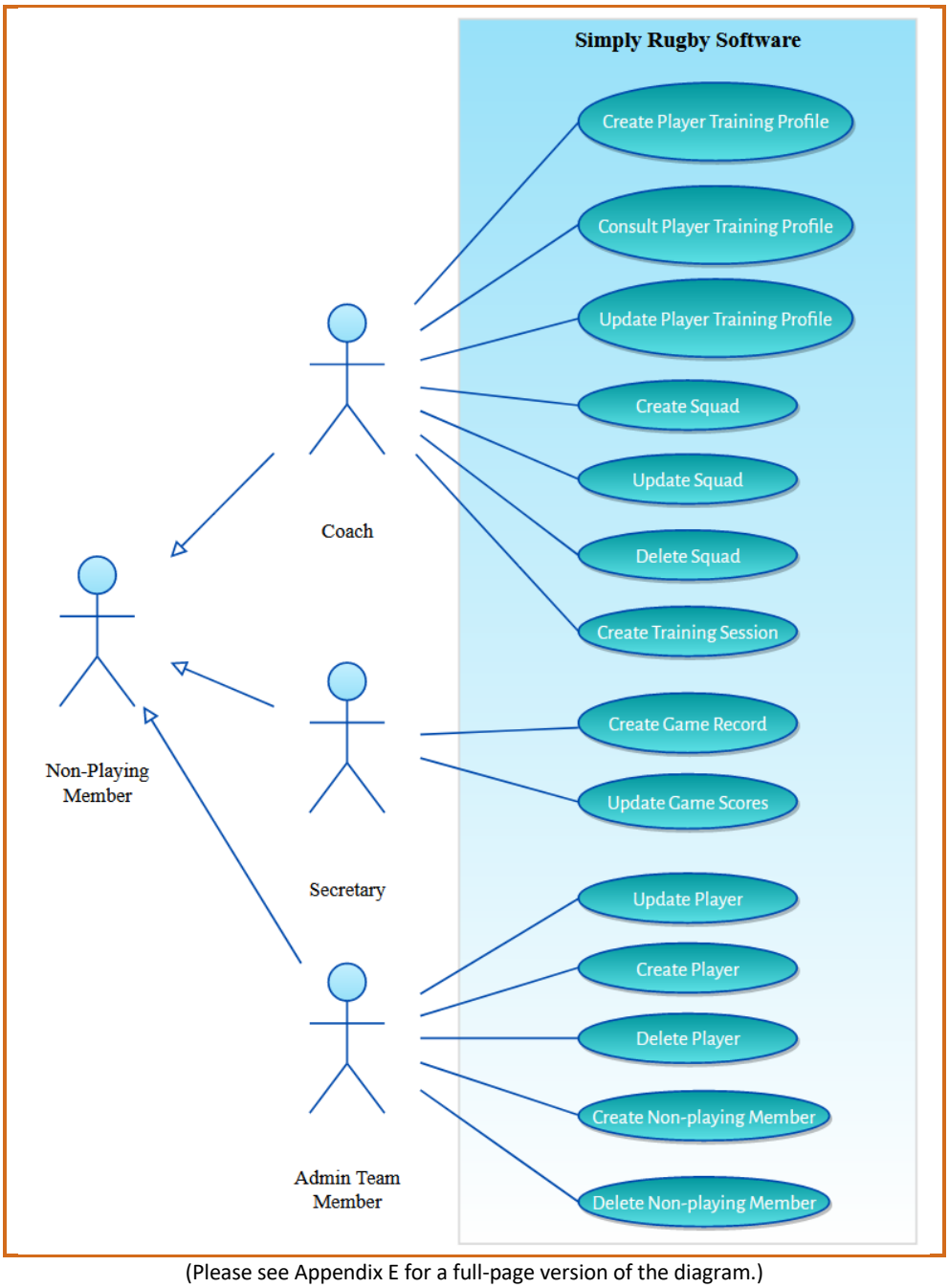
ASSUMPTIONS

The aim of this section is to list some of the assumptions that were taken during the planning and that may impact the design and implementation of the solution. They were mainly decided to keep the complexity of the project in line with the graded unit specifications, they would not be valid in a real-world scenario.

- There is no need to provide a solution that would need user profiles and roles. The functions of the application will be available to all users.
- The application will use 5 main skills to track the player's performance: Passing, Running, Support Play, Tackling, Decision making.
- The performance levels will use a 1 to 5 scale.
- The Consent form includes the health checks that may be required for junior players.
- Registering with the SRU via SCRUMS will issue a SCRUMS number that will be used to check if a player has complied with the requirement. For the purpose of the project, the number will be 7 digits long.
- It is assumed that the club already follow some of the GDPR/DPA18 requirements like Privacy notice, data processing consent forms.
- Mini and Junior squad will be merged as one in the database as they will follow the same set of rugby rules (as opposed to the real AGLVs rules).
- Mini and Junior Squads will use the Rugby 7 composition (see the relevant section in the database design section for more details). The Senior squad will follow the official rugby 15 players team composition. Replacement headcount will also be arbitrary (the official rules mention a maximum of 8 players, 3 in the front row).
- For the Coach assigned to a squad, it is assumed that 3 coaches are assigned to a squad.
- For the UX analysis part, it is assumed we could interview users and get some feedback on the initial wireframes.

HIGH LEVEL USE CASE DIAGRAM

Once the analysis of the client requirement has been done, the high-level Use Case diagram provides a summary of the functionalities expected by the client and will serve as a base to refine in the Design section of this document.



(Please see Appendix E for a full-page version of the diagram.)

RESEARCH AND RESOURCES

For this project, several resources will be used, this includes hardware, software (including web-based tools.)

HARDWARE RESOURCES

For the project, a desktop with the following specification will be used by the freelancer(student) to plan, design and develop the client's solution:

Operating System	Windows 11
Processor	AMD Ryzen 9 5900X
Ram	16.0 GB
GPU	Nvidia gpu

SOFTWARE RESOURCES

The list of software tools that will be used during the project:

Software	Description	Publisher
Microsoft Word	Word processor to write the different project documents.	Microsoft
Microsoft Excel	Spreadsheet software, mainly used to format tables.	Microsoft
IntelliJ	a Java Integrated Development environment used to program the solution	JetBrains
SQLite	Open-source library used to implement a database.	SQLite.org
DB Browser	Open-source software used to manipulate and view an SQLite database.	SQLiteBrowser
StarUML	Software used to draw UML diagrams, mainly used here for Sequence diagrams	MKLabs Co
SceneBuilder	free tool providing a visual interface to build JavaFX scenes (windows for the program)	Gluon
ProjectLibre	Open-source project management tool	ProjectLibre
GitHub	Version control tool, mainly used in this case as a backup solution for the code.	GitHub

WEB-BASED TOOLS

The following tools were used as additional resources during to project to produce some parts of the documentation.

Tool Name	Description	Website
Visual Paradigm	Free to use diagram tool. Used for Use Case and Activity Diagrams.	https://www.visual-paradigm.com/
Moqups	Another free diagram tools, used in the project for the Entity Relationship Diagram.	https://moqups.com/
UserBit	Tool used to produce persona cards	https://userbit.com/
DesignEvo	Tool used to make logos	https://www.designevo.com/
ColorDesigner	Colour palette generator	https://colordesigner.io/
Balsamiq	Wireframe tool	https://balsamiq.cloud/
Draw.io	generic UML/diagram tool. Used for the class diagram	https://app.diagrams.net/

RESEARCH MATERIALS

Research methodology

For this project, a focused approach will be taken regarding the research to be done. Working on the project requirements helps define what should be researched or not and the limited timeline means we must focus on research relevance.
This means that there is no need for example to present the history of the sport or its rules, except of course when knowing a point of rule is required in one of the solution functions, ex: to store the score breakdown, it will be necessary to know how the points are scored during the game.

Most of the research materials will be from the web and accessed from the desktop PC mentioned previously.

Those materials can be separated in two categories:

- Research around rugby and rugby clubs and their structure
- Research around the technologies that will be used to develop the software.

The resources used can be found in the References section.

Rugby resources

There are many sites available to describe rugby, its rules, organisation, from fan sites to official federations websites.
For most of the project development, official site like World Rugby or the official Scottish Rugby were used to provide context and information around the requirements.
Some more general websites were also consulted, especially for GDPR/DPA guidance relating to sports clubs, or for role position in rugby.

Technical resources

Again, there is a breadth of resource to use from the net for the application development.
Looking at the technologies involved, it will be around the following knowledge areas:

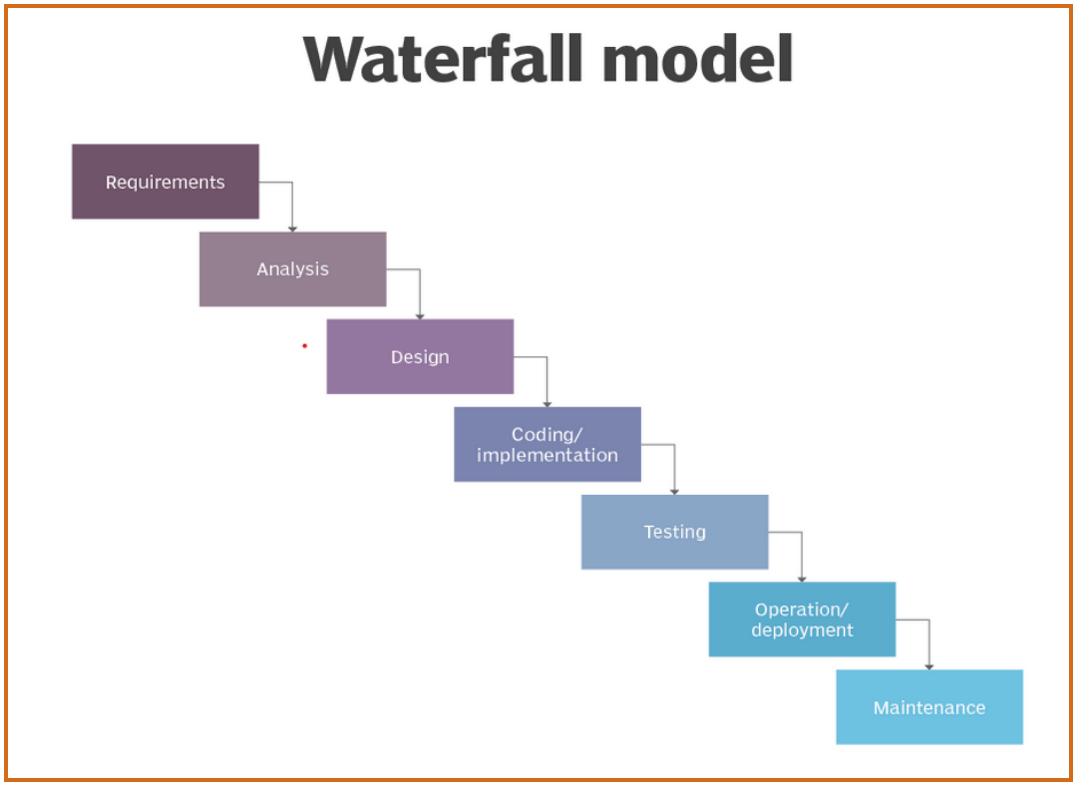
- Java resources: sites like W3School, the official Javadoc sites, specialised resource on JavaFX.
- SQLite resources for the database coding, Oracle academy for SQL.

PROJECT MANAGEMENT

SOFTWARE DEVELOPMENT LIFECYCLE

THE WATERFALL MODEL

This model is one of the earliest models developed for software development. It is a linear model, taking a step-by-step approach. Each step must be completed before the following one can start.
This model usually follows the steps shows in this diagram:



(Source: EDUCBA, https://www.educba.com/waterfall-model/)

This model can be resumed in the following way:

- Requirements: gather the specification, data, and the requirements of the project, to understand client's request, it should be as exhaustive as possible.
- Analysis: the phase of studying and understanding the requirements to design a solution to the request.
- Design: this is where the architecture, specifications, features of the solution are studied and designed.

- Implementation: the phase where the application is developed, based on the previous design.
- Testing: the product is tested according to requirements and to make sure it is for use.
- Deployment: product/result sent to end user/client.
- Maintenance: follow-up and maintenance of the product delivered.

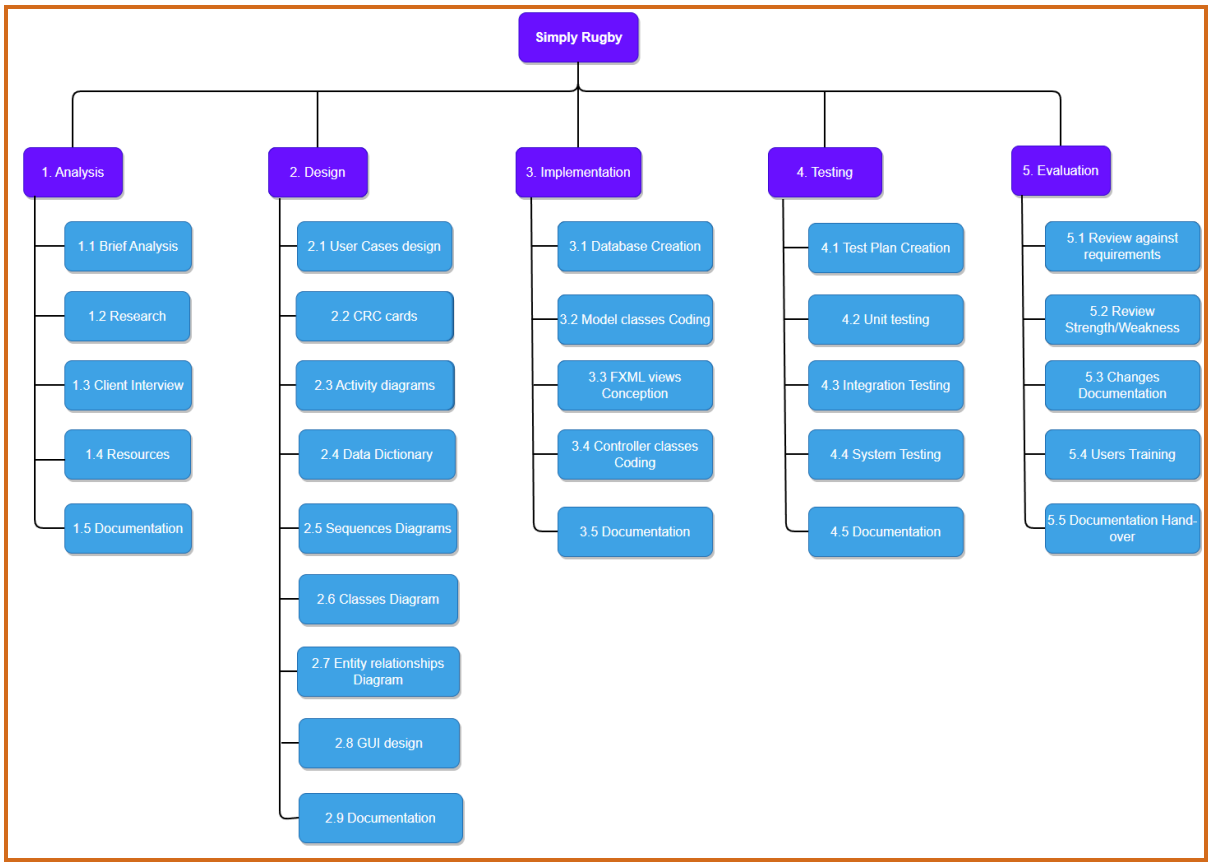
Advantages: this model provide a clear structure and path to follow for the development, it works well on small project or for freelance work.
Disadvantage: the waterfall model is not good for large or complex model. It has a rigid structure as changes cannot be made once a phase is finished and do involves risks and uncertainty in regard of the final product.

As such, the waterfall model will be the one proposed and followed during this project.

PROJECT DOCUMENTATION

WORK BREAKDOWN STRUCTURE

The WBS present a hierarchical view of the small tasks that composes the project.



(See Appendix F for a bigger view)

RESOURCES

Here are detailed the resources list that would be passed to the client as part of the project documentation and that is used in the project management software.

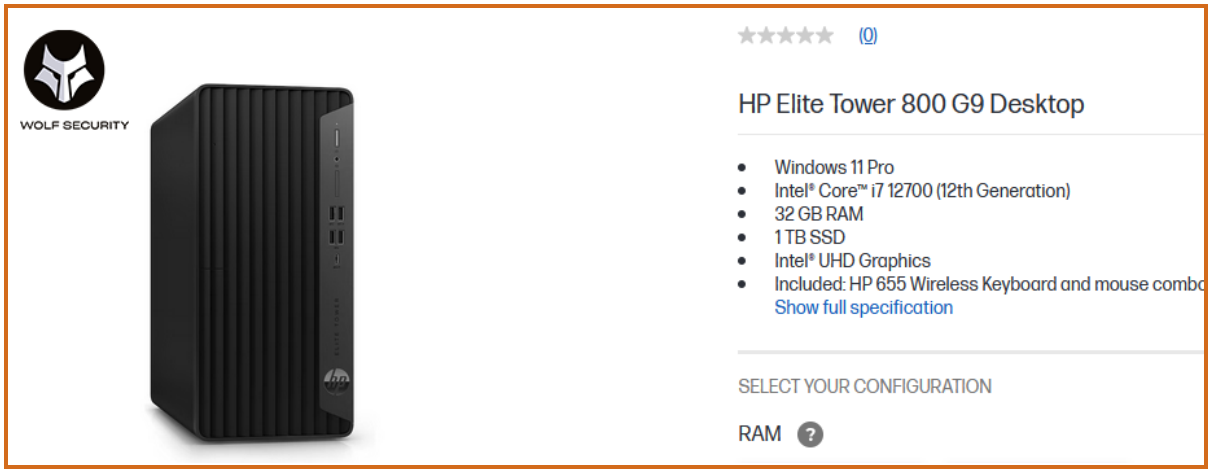
The first cost associated to the project is the freelance (Student)

Resource ID	Name	Profession	Cost per hour
1	Erik McSeveney	Software Engineer	£110

In this project, the cost of the licences for the software and tools used are integrated in the developer hourly rate., which is why they are not costed separately in the following table.

Resource ID	Resource Type	Resource name	Unit Cost
2	Chart tool	Draw.io	£0
3	Diagram tool	Visual Paradigm Online	£0
4	Spreadsheet	Excel	£0
5	Word processor	Word	£0
6	JavaFX gui builder	SceneBuilder	£0
7	ERD tool	Moqups	£0
8	Database	SQLite	£0
9	Database viewer	DB Browser (SQLite)	£0
10	Java development tool	IntelliJ	£0

As an option, the following PC desktop is quoted at the club's request:



(See Appendix H for details)

The specification would make it ideal for an office usage. A larger hard drive was selected to accommodate the possible size of the database.

PROJECT ESTIMATED COST

Based on previous resources proposal and the timeline explained in the constraints section and 8 hours a day, we would have the following cost summary for this project:

Simply Rugby

Dates			
Start	06/03/23 08:00	Finish	02/06/23 17:00
Baseline Start		Baseline Finish	
Actual Start		Actual Finish	

Duration			
Scheduled	65 days	Remaining	65 days
Baseline	0 days	Actual	0 days
		Percent Complete	0%

Work			
Scheduled	520 hours	Remaining	520 hours
Baseline	0 hours	Actual	0 hours

Costs			
Scheduled	£57200.00	Remaining	£57200.00
Baseline	£0.00	Actual	£0.00
		Variance	£0.00

This does not include the cost of a PC should the client wish to buy the proposed hardware.

GANTT ACTIVITY LIST

The Analysis stage will provide some documents as deliverables: Requirements Specification document, the Activity list and Gant bar Diagram can be provided to the client as part of the project management documentation.

Stage	Activity	Duration	Predecessor	resource ID
1. Analysis		5		
1.1 Brief Analysis	Textual analysis of the client brief, establish functional and non-functional requirements	1		1;5
1.2 Research	Research around the project theme	1	1.1	1;5
1.3 Client Interview	Acquiring further info from client, refine the requirements	1	1.2	1;5
1.4 Resources	Listing the required resources for the project	1	1.3	1;5
1.5 Documentation	Documenting the previous steps	1	1.4	1;5
Analysis Milestone		0		

Deliverables for the second phase would be most of design diagrams: Use Cases, Activity diagrams, class diagrams, most UX design documents.

Stage	Activity	Duration	Predecessor	resource ID
2. Design		15		
2.1 User Cases design	Creating high level and more precise user cases diagrams	2	1.5	1;3
2.2 CRC cards	Class Responsibility Cards design	1	2.1	1;2
2.3 Activity diagrams	designing flowcharts for the user cases	2	2.2	1;3
2.4 Data Dictionary	Listing Classes, data, methods	1	2.3	1;2
2.5 Sequences Diagrams	Designing interactions diagrams between actors	2	2.4	1;3
2.6 Classes Diagram	Formalise the data dictionary in a class diagram	1	2.5	1;2
2.7 Entity relationships Diagram	Design the required database	2	2.6	1;7
2.8 GUI design	Create wireframes	2	2.7	1;3
2.9 Documentation	Documenting the previous steps	2	2.8	1;5
Design Milestone		0		

At this stage, the main deliverable will be the solution itself, a software aiming to solve the problem submitted by the client.

Stage	Activity	Duration	Predecessor	resource ID
3. Implementation		21		
3.1 Database Creation	Create the database	1	2.9	1;8
3.2 Model classes Coding	Code the Model classes used by the system	4	3.1	1;9
3.3 FXML views Conception	Code the View classes	6	3,2	1;9
3.4 Controller classes Coding	Code the Controller classes	6	3.3	1;9
3.5 Documentation	Documenting the previous steps	4	3.4	1;9;5
Implementation Milestone		0		

For this phase, the deliverables are mainly documents like Testing Plan to highlight how the phase will go. Test cases document, which details each test and results. A defect report to document any issues and solutions.

Stage	Activity	Duration	Predecessor	resource ID
4. Testing		14		
4.1 Test Plan Creation	establish the testing plan	2	3.5	1;5
4.2 Unit testing	Class/method testing	3	4.1	1;4
4.3 Integration Testing	Classes interactions testing	3	4.2	1;4
4.4 System Testing	Full software testing	3	4.3	1;4
4.5 Documentation	Documenting the previous steps	3	4.4	1;5
Testing Milestone		0		

For the last phase, another set of documents can be given as deliverable: the user manual, a technical report about the application development, a maintenance plan should the client wishes to review and update the solution later, a formal user training can be included.

Stage	Activity	Duration	Predecessor	resource ID
5. Evaluation/ Maintenance		10		
5.1 Review against requirements	Review the product against the requirements	1	4.5	1;4
5.2 Review Strength/Weakness	assess the product	1	5.1	1;4
5.3 Changes Documentation	document the changes done during implementation	2	5.2	1;4
5.4 User Training	Document learning done during project	4	5.3	1;4
5.5 Documentation handover	Documenting the previous steps	2	5.4	1;4
Evaluation Milestone		0		

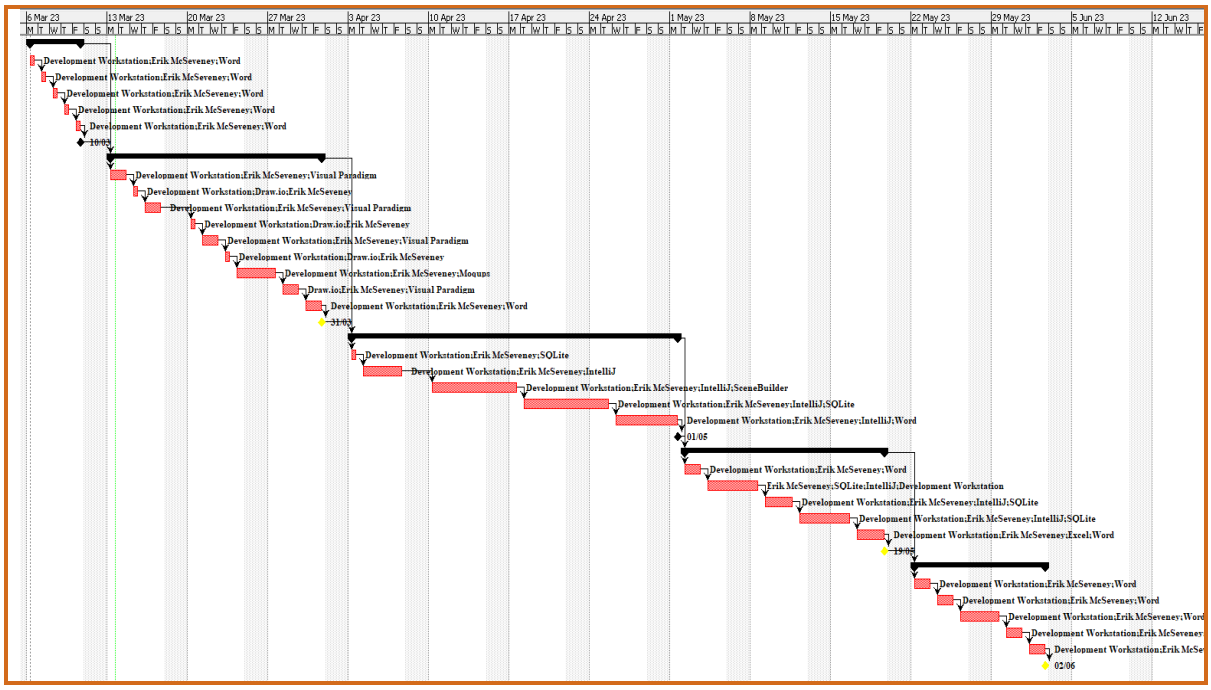
GANTT TASK LIST

The Gantt list details the tasks that compose a project, it shows their length, description, the resources used. It also shows the project milestones. This list is used by the project management tool to produce the Gantt diagram.

Name	Duration	Start	Finish	Predecessors	Resource Names
1. Analysis	5 days	**06/03/23 08:00**	**10/03/23 17:00**		
1.1 Brief Analysis	1 day	06/03/23 08:00	06/03/23 17:00		Development Workstation;E
1.2 Research	1 day	07/03/23 08:00	07/03/23 17:00	2	Development Workstation;E
1.3 Client Interview	1 day	08/03/23 08:00	08/03/23 17:00	3	Development Workstation;E
1.4 Ressources	1 day	09/03/23 08:00	09/03/23 17:00	4	Development Workstation;E
1.5 Documentation	1 day	10/03/23 08:00	10/03/23 17:00	5	Development Workstation;E
Analysis Milestone	0 days	10/03/23 17:00	10/03/23 17:00	6	
2. Design	15 days	**13/03/23 08:00**	**31/03/23 17:00**	**1**	
2.1 User Cases design	2 days	13/03/23 08:00	14/03/23 17:00	7	Development Workstation;E
2.2 CRC cards	1 day	15/03/23 08:00	15/03/23 17:00	9	Development Workstation;D
2.3 Activity diagrams	2 days	16/03/23 08:00	17/03/23 17:00	10	Development Workstation;E
2.4 Data Dictionary	1 day	20/03/23 08:00	20/03/23 17:00	11	Development Workstation;D
2.5 Sequences Diagrams	2 days	21/03/23 08:00	22/03/23 17:00	12	Development Workstation;E
2.6 Classes Diagram	1 day	23/03/23 08:00	23/03/23 17:00	13	Development Workstation;D
2.7 Entity relationships Dia	2 days	24/03/23 08:00	27/03/23 17:00	14	Development Workstation;E
2.8 GUI design	2 days	28/03/23 08:00	29/03/23 17:00	15	Draw.io;Erik McSeveney;Vis
2.9 Documentation	2 days	30/03/23 08:00	31/03/23 17:00	16	Development Workstation;E
Design Milestone	0 days	31/03/23 17:00	31/03/23 17:00	17	
3. Implementation	21 days	**03/04/23 08:00**	**01/05/23 17:00**	**8**	
3.1 Database Creation	1 day	03/04/23 08:00	03/04/23 17:00	18	Development Workstation;E
3.2 Model classes Coding	4 days	04/04/23 08:00	07/04/23 17:00	20	Development Workstation;E
3.3 FXML views Conceptior	6 days	10/04/23 08:00	17/04/23 17:00	21	Development Workstation;E
3.4 Controller classes Codi	6 days	18/04/23 08:00	25/04/23 17:00	22	Development Workstation;E
3.5 Documentation	4 days	26/04/23 08:00	01/05/23 17:00	23	Development Workstation;E
Implementation Milestone	0 days	01/05/23 17:00	01/05/23 17:00	24	
4. Testing	14 days	**02/05/23 08:00**	**19/05/23 17:00**	**19**	
4.1 Test Plan Creation	2 days	02/05/23 08:00	03/05/23 17:00	25	Development Workstation;E
4.2 Unit Testing	3 days	04/05/23 08:00	08/05/23 17:00	27	Erik McSeveney;SQLite;Inte
4.3 Integration Testing	3 days	09/05/23 08:00	11/05/23 17:00	28	Development Workstation;E
4.4 System Testing	3 days	12/05/23 08:00	16/05/23 17:00	29	Development Workstation;E
4.5 Documentation	3 days	17/05/23 08:00	19/05/23 17:00	30	Development Workstation;E
Testing Implementation	0 days	19/05/23 17:00	19/05/23 17:00	31	
5. Evaluation	10 days	**22/05/23 08:00**	**02/06/23 17:00**	**26**	
5.1 Review against require	1 day	22/05/23 08:00	22/05/23 17:00	32	Development Workstation;E
5.2 Review Strenght/Weak	1 day	23/05/23 08:00	23/05/23 17:00	34	Development Workstation;E
5.3 Changes Documentatic	2 days	24/05/23 08:00	25/05/23 17:00	35	Development Workstation;E
5.4 User Training	4 days	26/05/23 08:00	31/05/23 17:00	36	Development Workstation;E
5.5 documentation	2 days	01/06/23 08:00	02/06/23 17:00	37	Development Workstation;E
Evaluation Milestone	0 days	02/06/23 17:00	02/06/23 17:00	38	

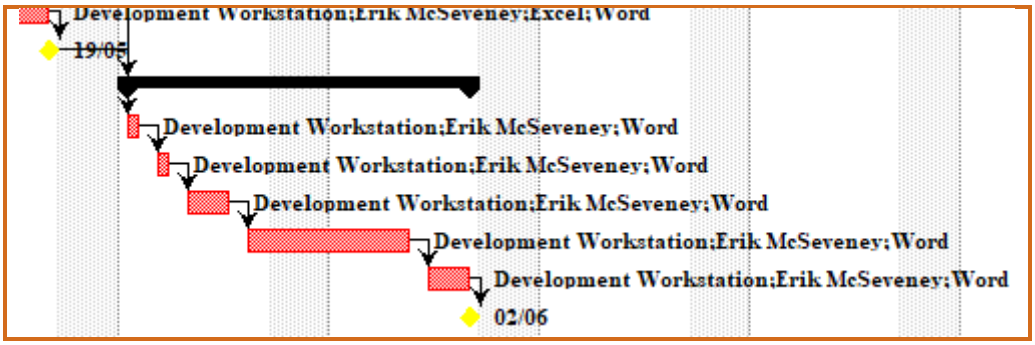
GANTT BAR DIAGRAM

This diagram shows a graphical representation of the tasks list. It shows the dependencies and the timeline in a clearer manner.
For this project, which is done by a single person, there is no tasks that can be done in parallel. Which is also why the Waterfall model goes well with this project.



(See Appendix G for a larger view)

The diagram shows each phase, their milestones but also have 3 deadlines, based on the unit deadlines: the 31/03, 19/05 and 02/06. They could be considered as deadlines with the client who requested to be shown during the project the status of the project and makes sure it is still progressing as planned. This would be a reasonable request considering the cost of the project is hourly based.



DESIGN

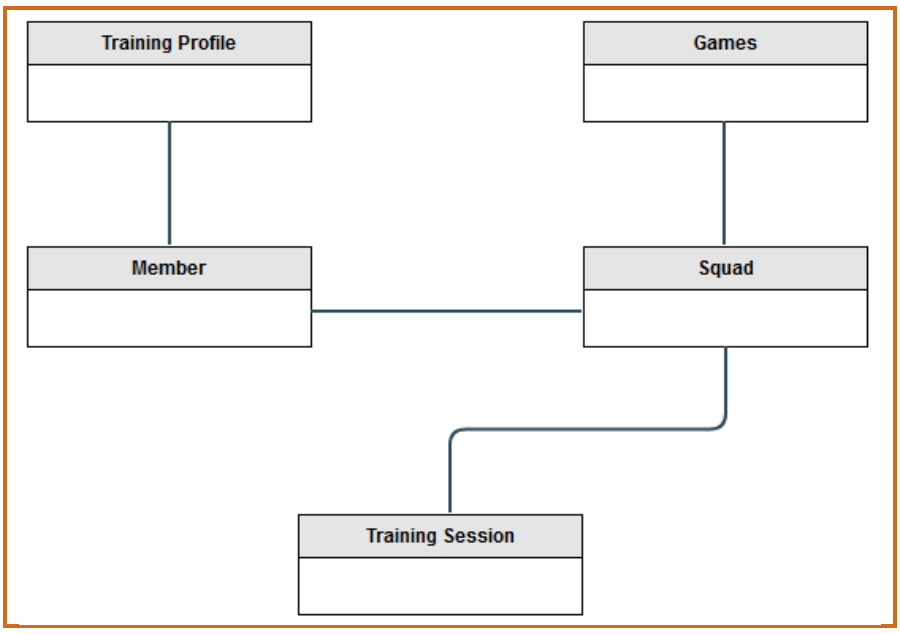
DATABASE DESIGN

INTRODUCTION

One of the main design choices taken for the solution proposed to the client is to use a relational database to store the data required.
This means that the structure needed by this database needs to be established for the data it will store, in what form and how.
To design the database, a textual analysis will provide the data that the model needs to capture. It will translate in an initial, simple representation of the data entities required in the database.
It will be then extended to include a more detailed list of the data that is required which will form the entities of the database.
At this point we will look at the entities attributes and refine them and work on their relationships with others.
The aim of those steps is to develop a complete Entity Relationships Diagram that will be the blueprint for the database used by the software.

INITIAL ANALYSIS

From the requirements analysis, we can extract the general entities of the database.
The club has **members** who can participate in **games**, **training sessions** or oversee the club life. Those non-playing members maintains **training profiles** of the playing base.
This gives us a very high-level view of the data model:



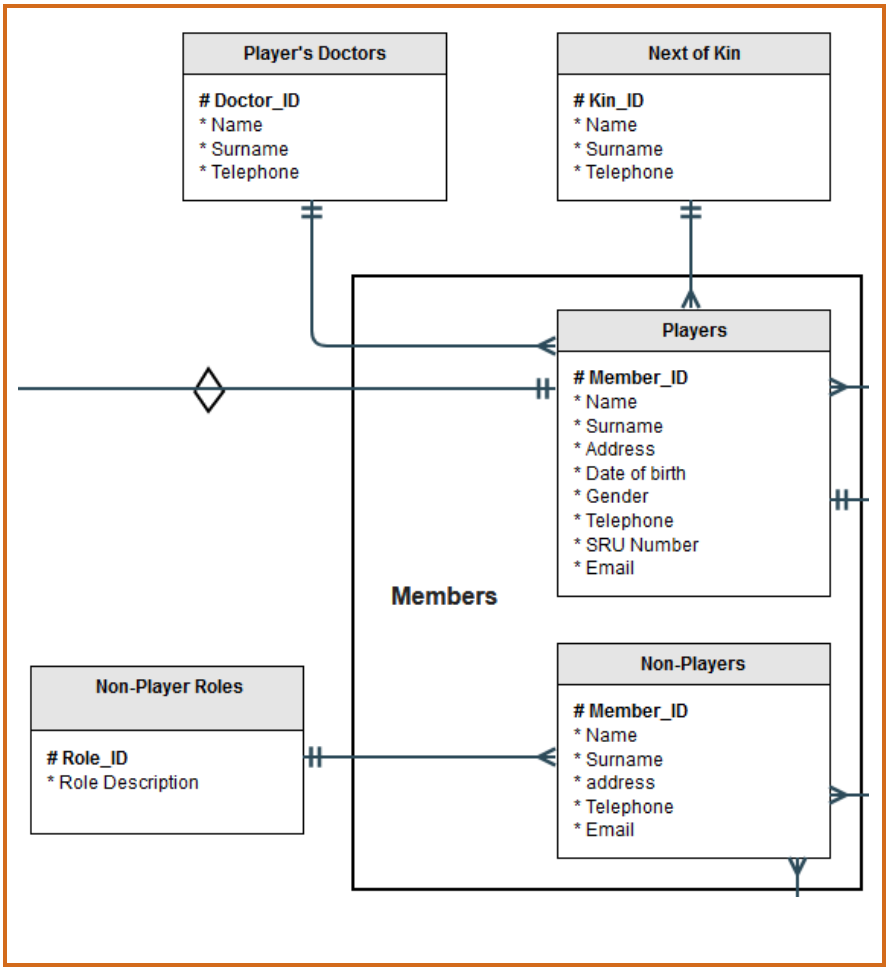
Obviously, this initial diagram needs to be refined as it does not cover all the requirements. Each 'category' shown in this diagram will now be studied and detailed below.

REFINING THE GENERAL MODEL

Members

Two types of members: Playing and Non-playing member. This will translate as subtypes.

For all members, we need to store their details: name, surname, phone, email.
For players, we need to add further information: date of birth (to calculate age), gender (determine what kind of rules/games can be player), Next of Kin details, Consent form for some, SRU number, doctors' details.
For non-player, we will add a role description.



Relationships can be described with ERDish:

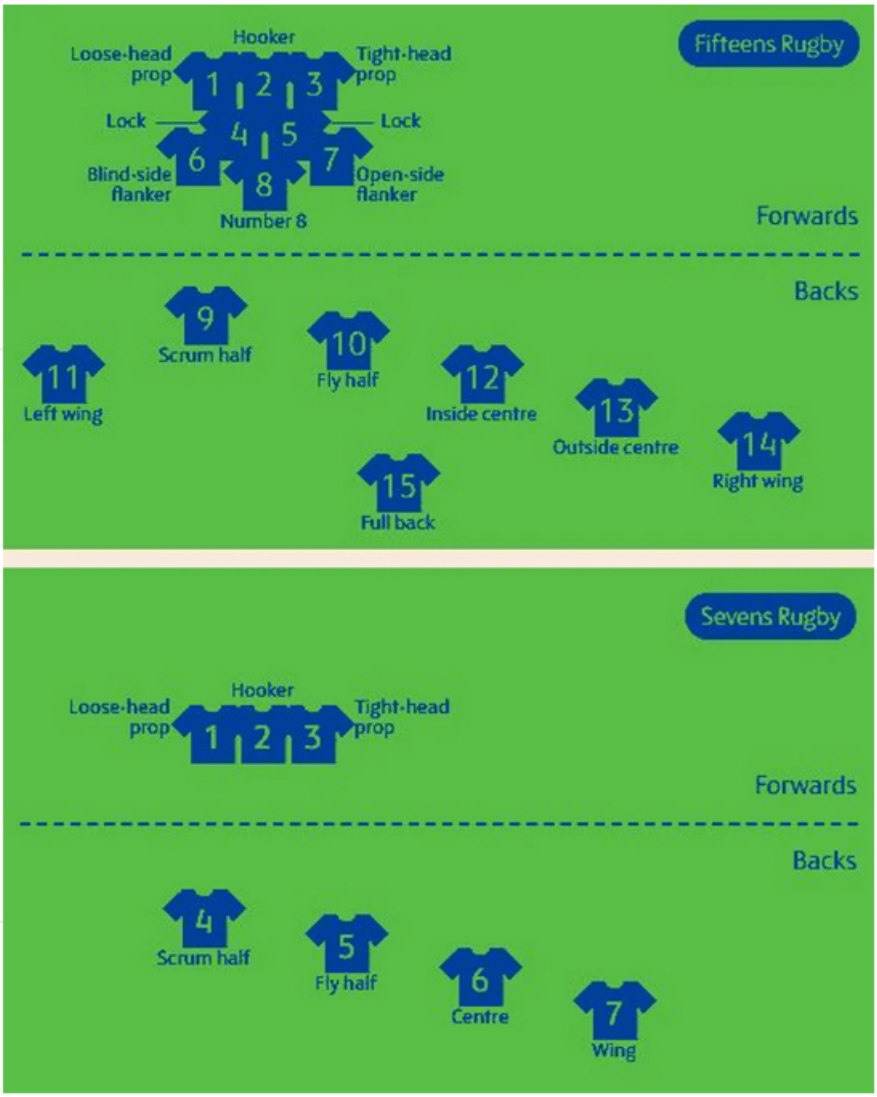
- A PLAYER have ONE AND ONLY ONE Next of Kin.
- A Next of Kin can cover MANY players.

Same as below for the doctors or roles.

Squads

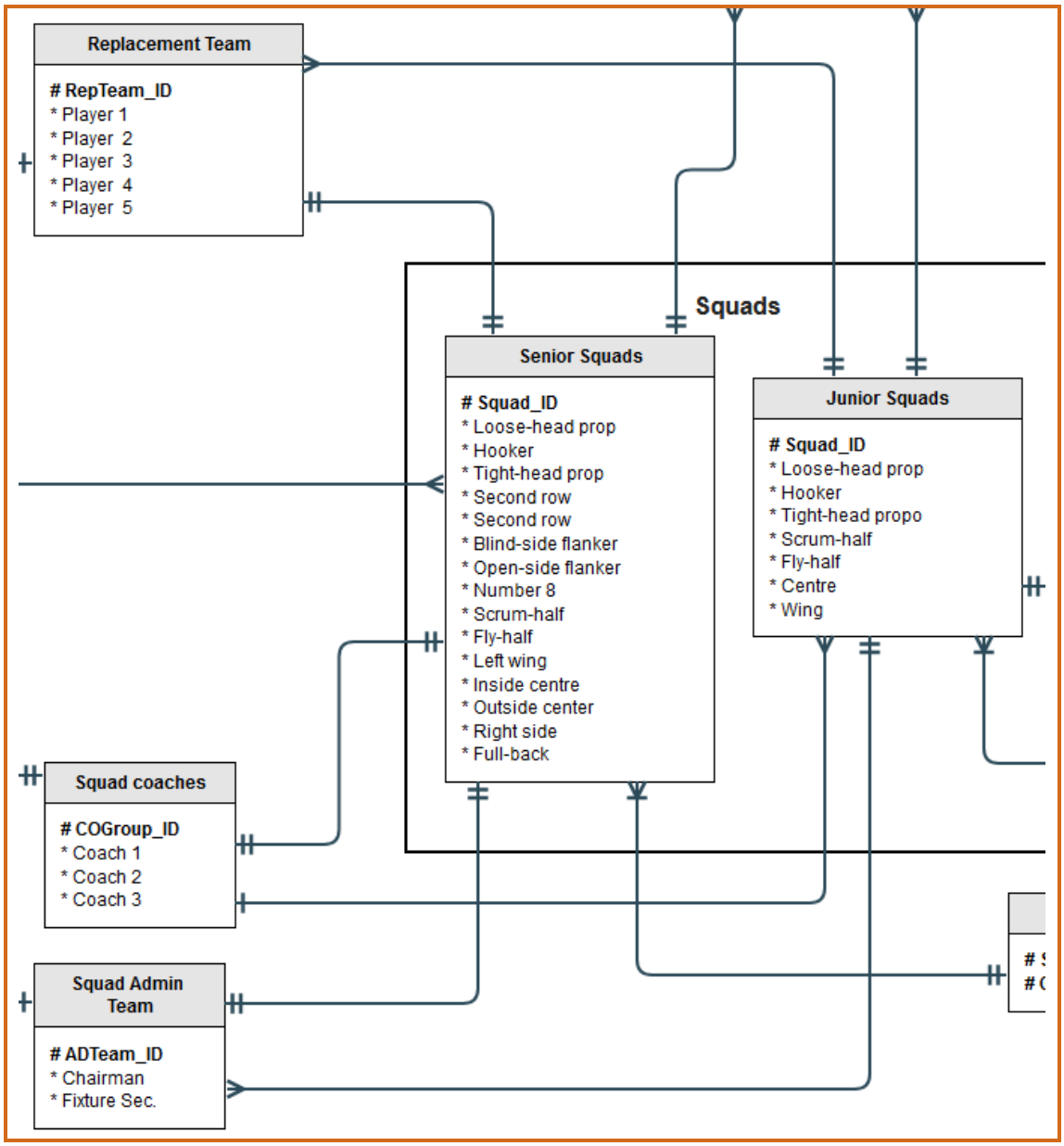
A squad can be one of the three age group: Mini, Junior or Senior. However, as mentioned in the assumption section, we will merge the Mini and Junior into one.
The Senior Squad includes the normal roles, the other will include 7 players.
A squad is assigned a coach team and an admin team, both composed with non-playing members.
A squad will also be linked to a Replacement team, which list the back-up players.

a squad is organised as follow:



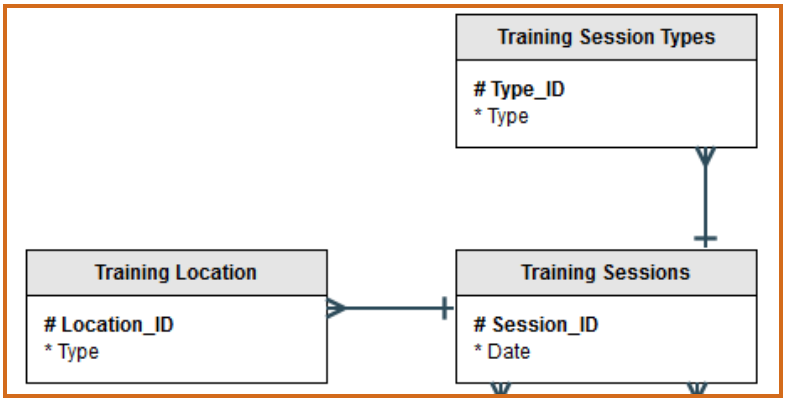
(Source: https://www.world.rugby/the-game/beginners-guide/positions)

In the ERD:


Replacement Team
RepTeam_ID
* Player 1
* Player 2
* Player 3
* Player 4
* Player 5
Squads
Senior Squads
Squad_ID
* Loose-head prop
* Hooker
* Tight-head prop
* Second row
* Second row
* Blind-side flanker
* Open-side flanker
* Number 8
* Scrum-half
* Fly-half
* Left wing
* Inside centre
* Outside center
* Right side
* Full-back
Junior Squads
Squad_ID
* Loose-head prop
* Hooker
* Tight-head propo
* Scrum-half
* Fly-half
* Centre
* Wing
Squad coaches
COGroup_ID
* Coach 1
* Coach 2
* Coach 3
Squad Admin Team
ADTeam_ID
* Chairman
* Fixture Sec.

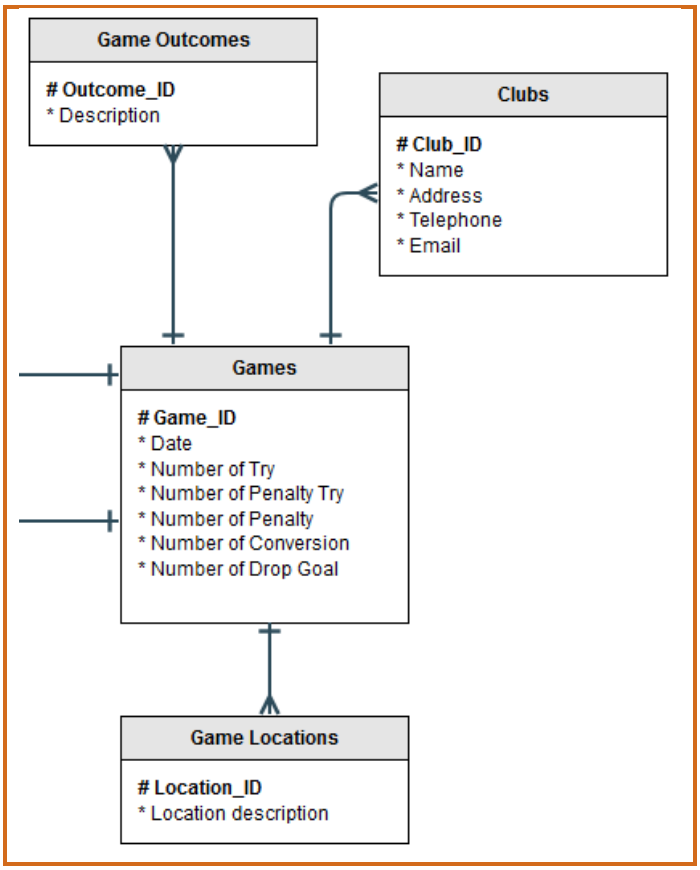
Training Sessions

A squad will participate in a training session, with its replacement team. A session will take place in one of the club's facilities and have a type (Strength, endurance, coordination, etc.).



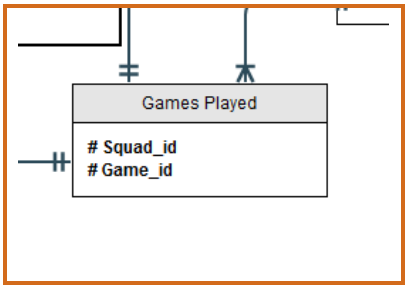
Games

The game entity will contain the details of the game: squad and opponent team, the breakdown of the score, the game location type (home or away). It will also link to a game outcome table (won, lost, forfeited, cancelled, etc.).



Click on Scoring Points from World Rugby for the points details.

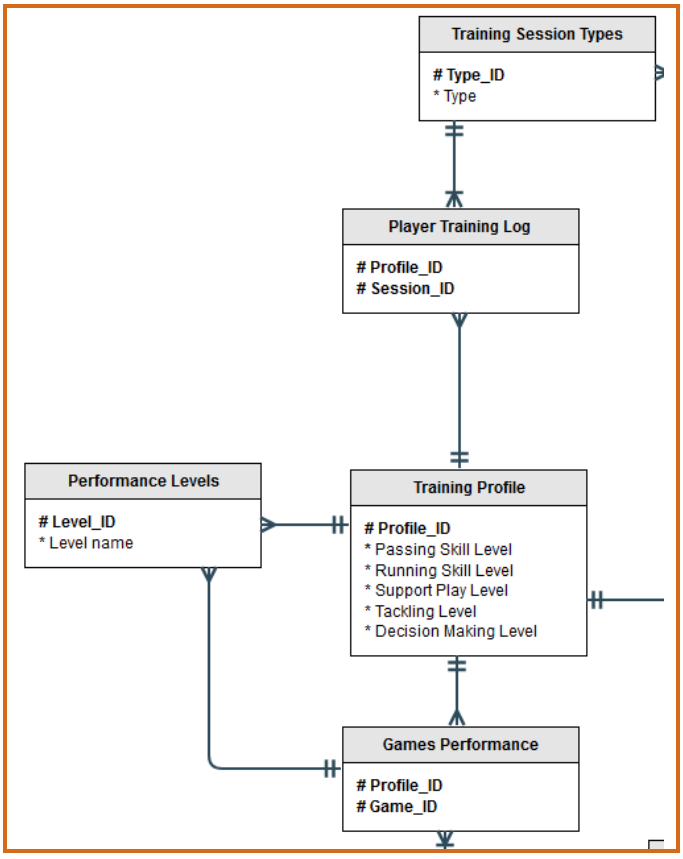
An intersection table between the game table and the squads will keep track of the games played.



Training profiles

The training profile is an entity that will link to a player, will hold the 5 core rugby skills agreed with the client linked to another table defining the skills levels.
As discussed the requirements, the training profile will hold information about the training sessions attended by the player and the player's games performances.
This will be stored in the database in two intersection tables:

- Game performance for the games played by the member.
- Player Training log will link to the training sessions.

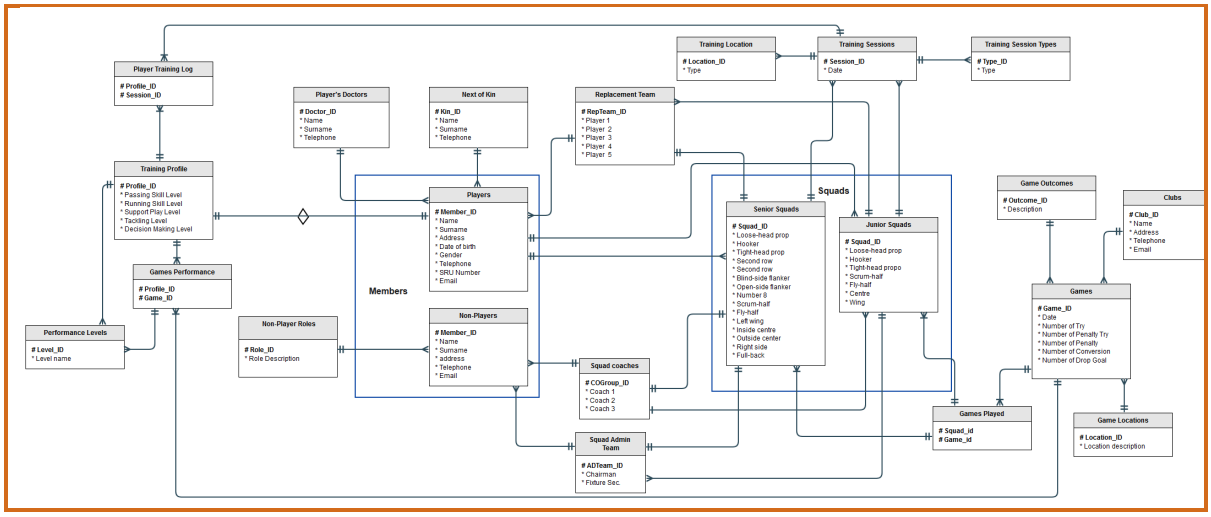


FULL ENTITY RELATIONSHIP DIAGRAM

Below is the ERD for the database for the project.

At this point, the tables have been normalised following the 3 Forms:

- First Form: no multi-valued attributes.
- Second Form: All Non-UID are dependent of the UID.
- Third Form: No dependence between non-UID attributes.



(See Appendix I for the full details)

DATABASE TABLES MAPPING

In this section, we will map the ERD in tables that will be used to create the database tables.

Legend

pk	Primary Key
fk	Foreign Key
*****	Mandatory
o	optional

PLAYERS		
Key Type	Optionality	Columns
pk	*	member_id
	*	first_name
	*	surname
	*	Address
	*	date_of_birth
	*	gender
	*	telephone
	*	email
	*	scrums_number
fk	*	kin_id
fk	*	doctor_id

NON_PLAYERS		
Key Type	Optionality	Columns
pk	*	member_id
	*	first_name
	*	surname
	*	address
	*	telephone
	*	email
fk	*	role_id

NON_PLAYERS_ROLES		
Key Type	Optionality	Columns
pk	*	role_id
	*	role_description

SENIOR_SQUADS		
Key Type	Optionality	Columns
pk	*	squad_id
fk	*	loose_head_prop
fk	*	hooker
fk	*	tight_head_prop
fk	*	second_row
fk	*	second_row
fk	*	blind_side_flanker
fk	*	open_side_flanker
fk	*	number_8
fk	*	scrum_half
fk	*	fly_half
fk	*	left_wing
fk	*	inside_centre
fk	*	outside_center
fk	*	right_side
fk	*	full_back
fk	*	cogroup_id
fk	*	adteam_id
fk	*	repteam_id

JUNIOR_MINI_SQUADS		
Key Type	Optionality	Columns
pk	*	player_1
fk	*	player_2
fk	*	player_3
fk	*	player_4
fk	*	player_5
fk	*	player_6
fk	*	player_7
fk	*	player_8
fk	*	cogroup_id
fk	*	adteam_id
fk	*	repteam_id

SQUAD_COACHES		
Key Type	Optionality	Columns
pk	*	cogroup_id
fk	*	coach_1
fk	*	coach_2
fk	*	coach_3

GAMES		
Key Type	**Optionality**	**Columns**
pk	*	game_id
	*	date
	o	nb_of_try
	o	nb_of_penalty_try
	o	nb_of_penalty
	o	nb_of_conversion
	o	nb_of_drop_goal
fk	o	outcome_id
fk	*	club_id
fk	*	location_id

CLUBS		
Key Type	**Optionality**	**Columns**
pk	*	club_id
	*	name
	*	address
	*	telephone
	*	email

REPLACEMENT_TEAM		
Key Type	**Optionality**	**Columns**
pk	*	repteam_id
fk	o	player_1
fk	o	player_2
fk	o	player_3
fk	o	player_4
fk	o	player_5

PLAYER_DOCTORS		
Key Type	**Optionality**	**Columns**
pk	*	doctor_id
	*	name
	*	surname
	*	telephone

GAME_LOCATION		
Key Type	**Optionality**	**Columns**
pk	*	location_id
	*	description

GAME_OUTCOMES		
Key Type	**Optionality**	**Columns**
pk	*	outcome_id
	*	description

SQUAD_ADMIN_TEAM		
Key Type	**Optionality**	**Columns**
pk	*	adteam_id
fk	*	chairman
fk	*	fixture_sec

GAMES_PLAYED		
Key Type	**Optionality**	**Columns**
pk, fk	*	location_id
Pk, fk	*	game_id

TRAINING_SESSION		
Key Type	**Optionality**	**Columns**
pk	*	session_id
	*	date

TRAINING_LOCATION		
Key Type	**Optionality**	**Columns**
pk	*	session_id
	*	date

TRAINING_SESSION_TYPE		
Key Type	**Optionality**	**Columns**
pk	*	type_id
	*	description

PLAYER_TRAINING_LOG		
Key Type	**Optionality**	**Columns**
pk, fk	*	profile_id
pk, fk	*	session_id

TRAINING_PROFILES		
Key Type	**Optionality**	**Columns**
pk	*	profile_id
fk	*	passing_skill
fk	*	running_skill
fk	*	support_skill
fk	*	tackling_skill
fk	*	decision_skill

PLAYER_DOCTORS		
Key Type	**Optionality**	**Columns**
pk	*	doctor_id
	*	name
	*	surname
	*	telephone

GAME_PERFORMANCES		
Key Type	**Optionality**	**Columns**
pk, fk	*	profile_id
pk, fk	*	game_id

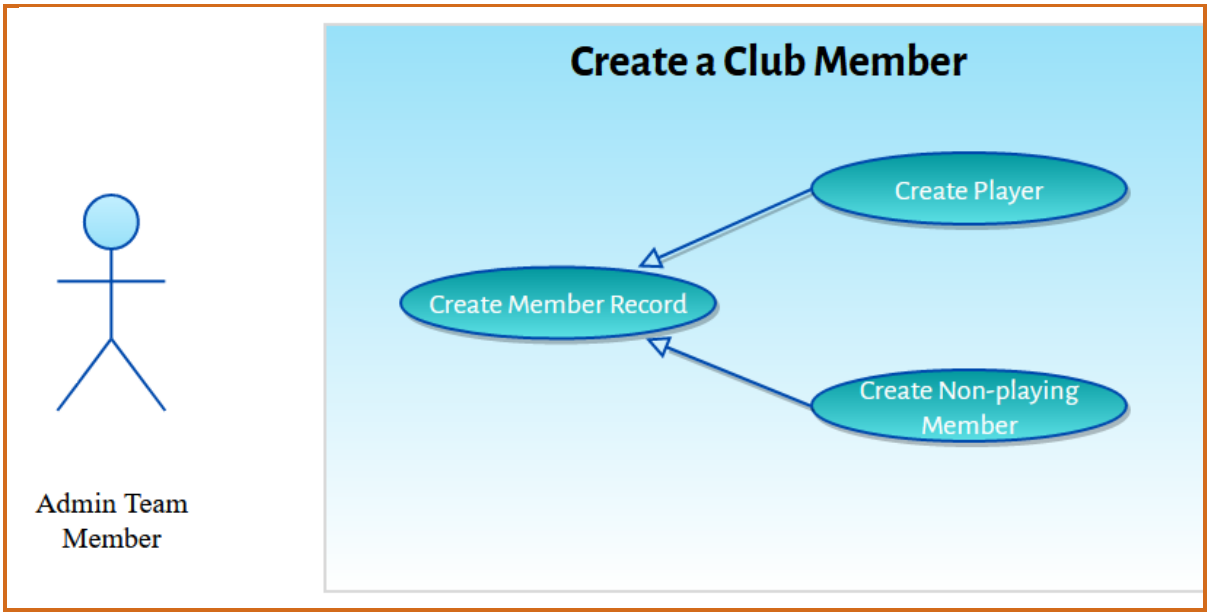
PERFORMANCE_LEVELS		
Key Type	**Optionality**	**Columns**
pk	*	level_id
	*	description

USE CASE DIAGRAMS AND DOCUMENTS

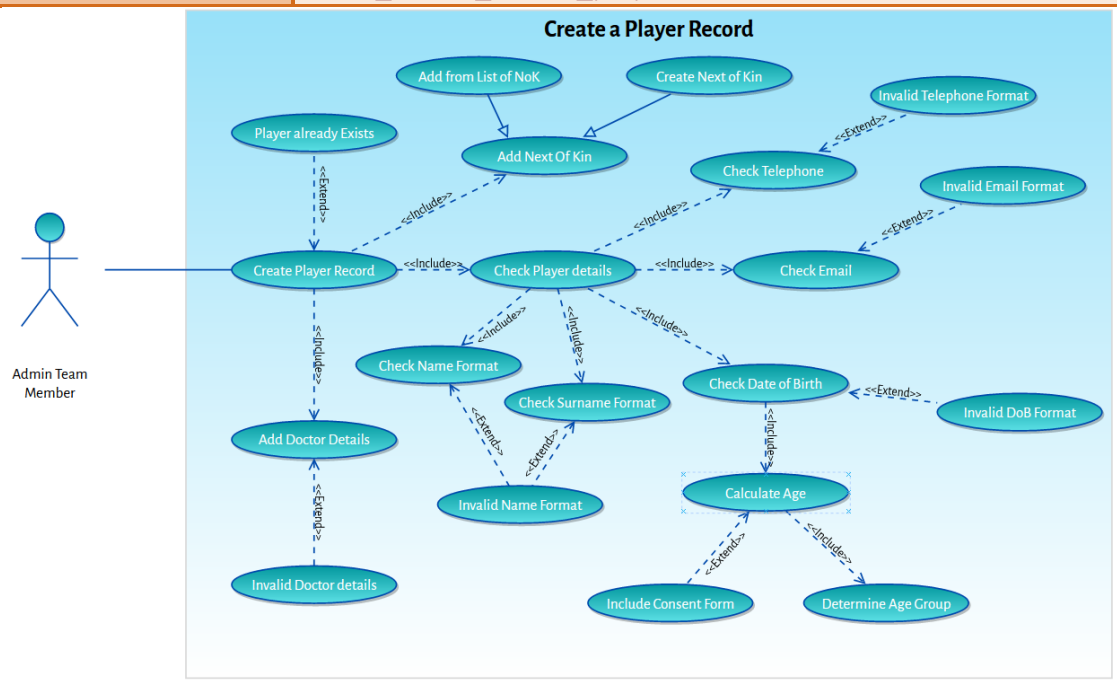
The specific details of the high-level use case diagram can be found below. Each use case will include its Use Case Documentation, an Activity Diagram and if relevant a Sequence Diagram.

USE CASE 1: CREATE A CLUB MEMBER

UC1 Diagram



Use Case Name:	UC1a_Admin_Create_player



See Appendix XX for full size Diagram

Initiating Actor(s):	Admin Team	**Receiving Actor(s):**	None

Trigger/Pre-condition(s):
The new player is registered with the SRU and the club fees are paid. If under 18-year-old, a consent form has been provided.

Main Flow of Events:
1) Admin inputs player's Name. 2) Admin inputs player's Surname. 3) Admin inputs Date of Birth. 4) Admin inputs Gender. 5) Admin inputs Telephone. 6) Admin inputs Email. 7) Admin inputs SCRUM number. 8) Admin inputs Doctor Name and Surname. 9) Admin inputs Doctor Telephone. 10) Check if the name or surname are valid. 11) Check if SCRUM number is valid. 12) Check if Player already exists in the system. 13) Check if the Date of Birth is valid. 14) Check if the Telephone is valid. 15) Check if the Email is valid. 16) Check if the Doctor's name or surname are valid. 17) Check if the Doctor's Telephone is valid. 18) Calculate Age. 19) If under 18 years old, include the Consent Form. 20) Admin adds a Next of Kin. 21) End of Use Case

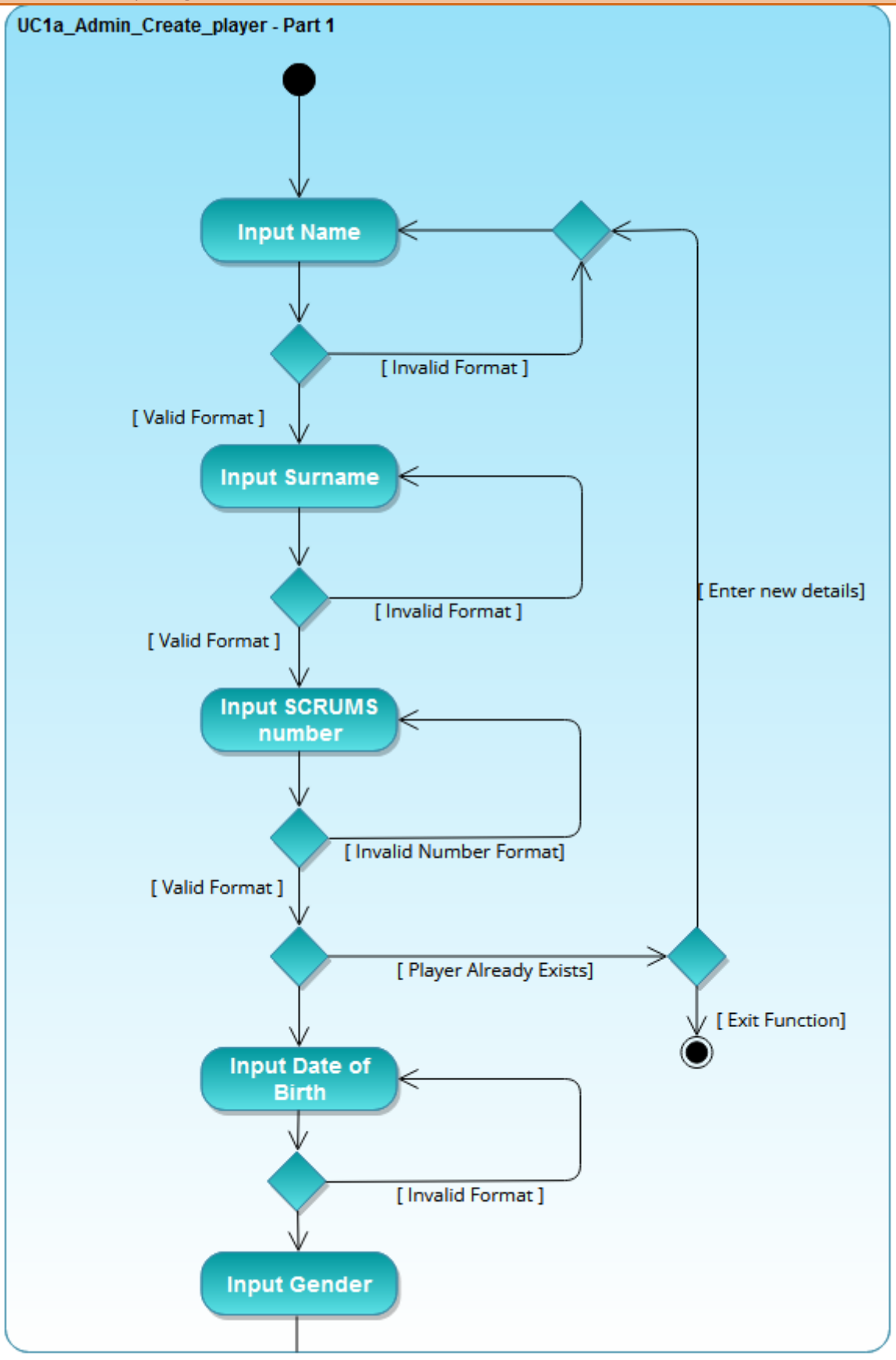
Alternate Flows:
10a. Invalid Name or Surname. 1) Display Error Message. 2) Input correct value. 3) Continue to next step. 11a. Invalid SCRUM number. 1) Display Error Message. 2) Input correct value. 3) Continue to next step. 12a. Player already Exists. 1) Display Error Message. 2) Admin chooses to correct. 3) Clear entered values. 4) Go back to step 1. 12b. Player already exists. 1) Admin chooses to exit. 2) Exit program.

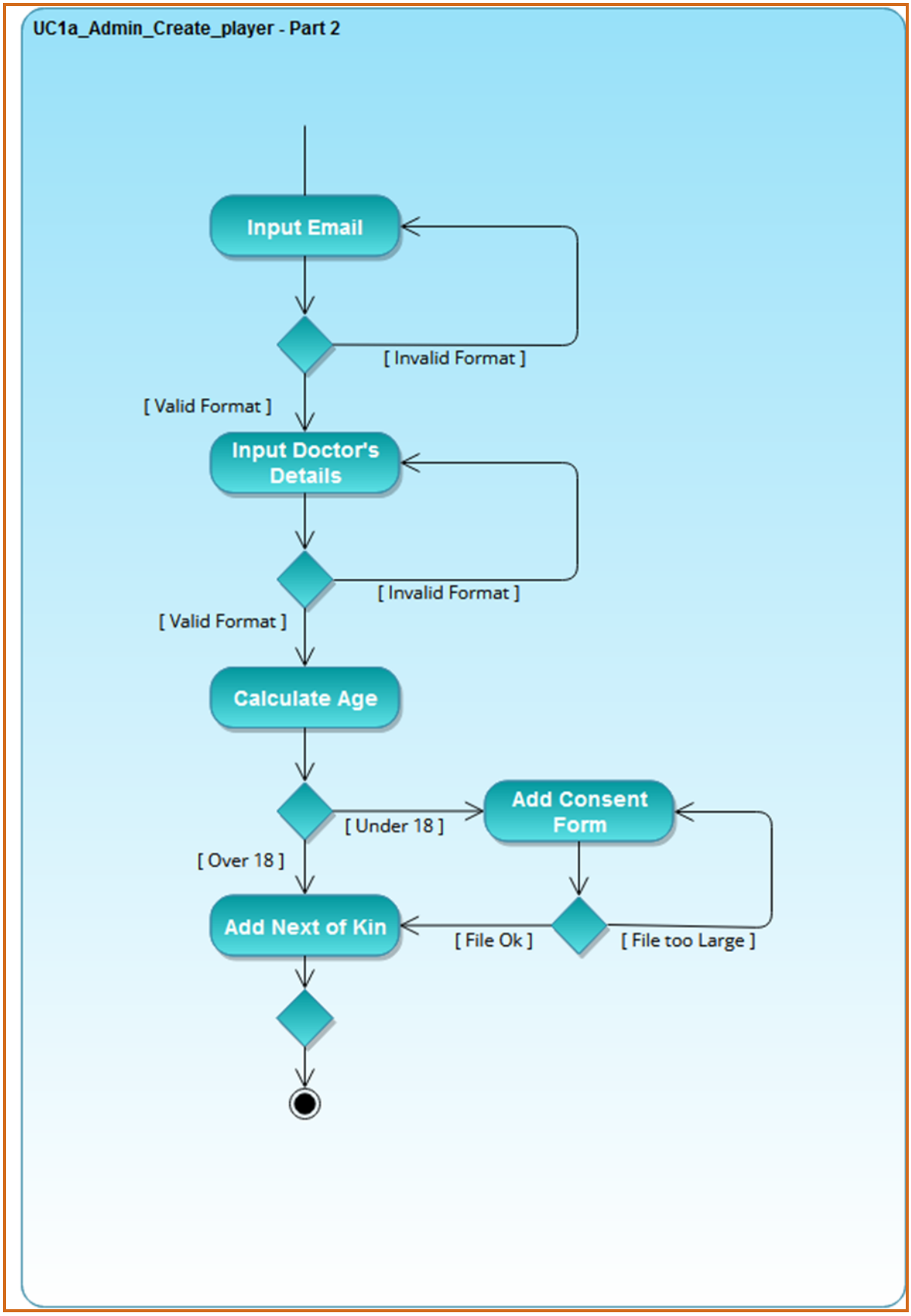
13a. Invalid Date of Birth.
1) Display Error Message.
2) Input correct value.
3) Continue to next step.
14a. Invalid Telephone number.
1) Display Error Message.
2) Input correct value.
3) Continue to next step.
15a. Invalid Email.
1) Display Error Message.
2) Input correct value.
3) Continue to next step.
16a. Invalid Doctor's Name or Surname.
1) Display Error Message.
2) Input correct value.
3) Continue to next step.
17a. Invalid Telephone number.
1) Display Error Message.
2) Input correct value.
3) Continue to next step.

Assumptions(optional):
Players must be unique, so the system will check if the combination of name, surname and SCRUM number are unique in the Database

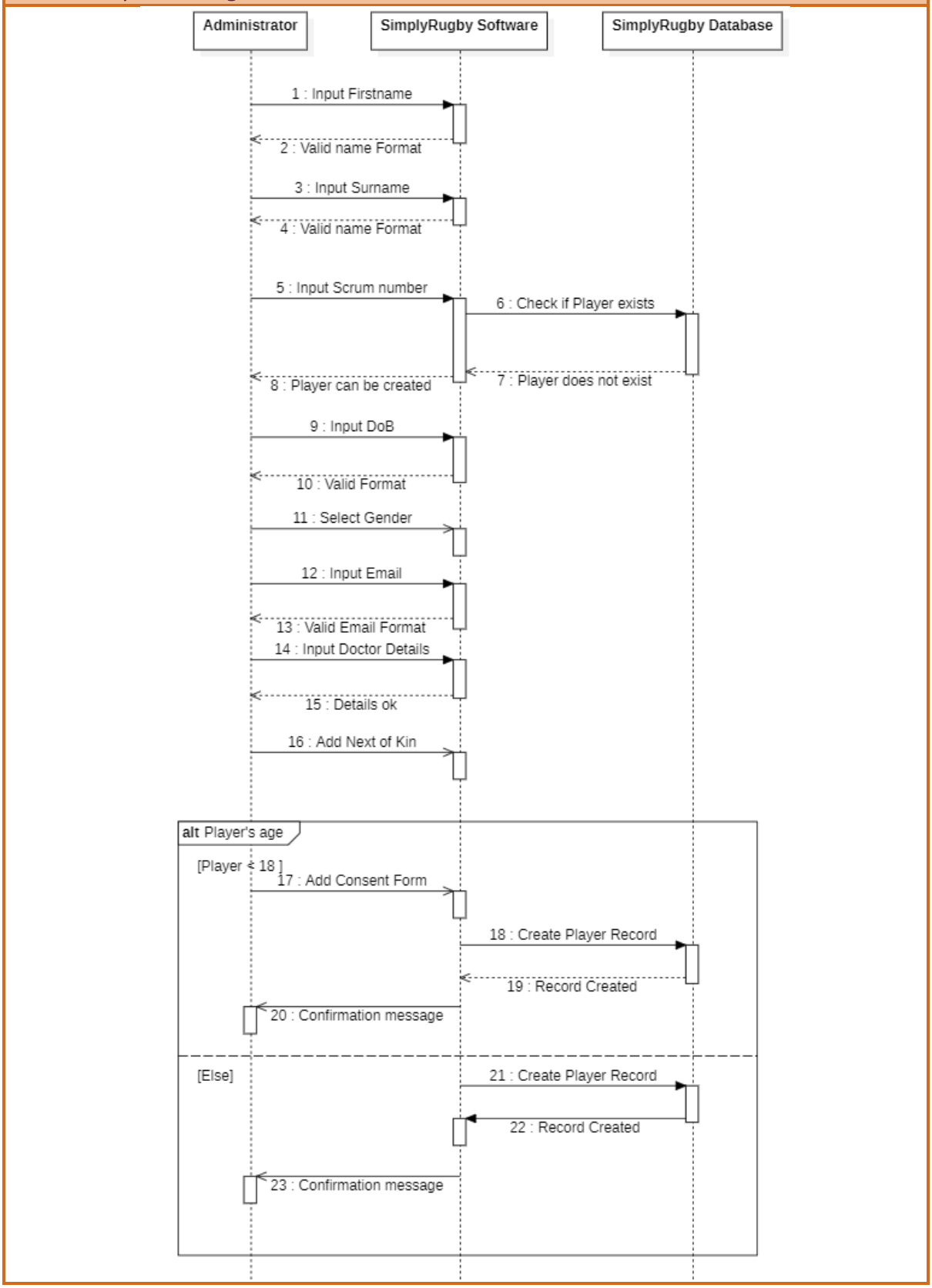
Post-conditions:
A new Player record is created and saved in the database.

UC1a – Activity Diagram

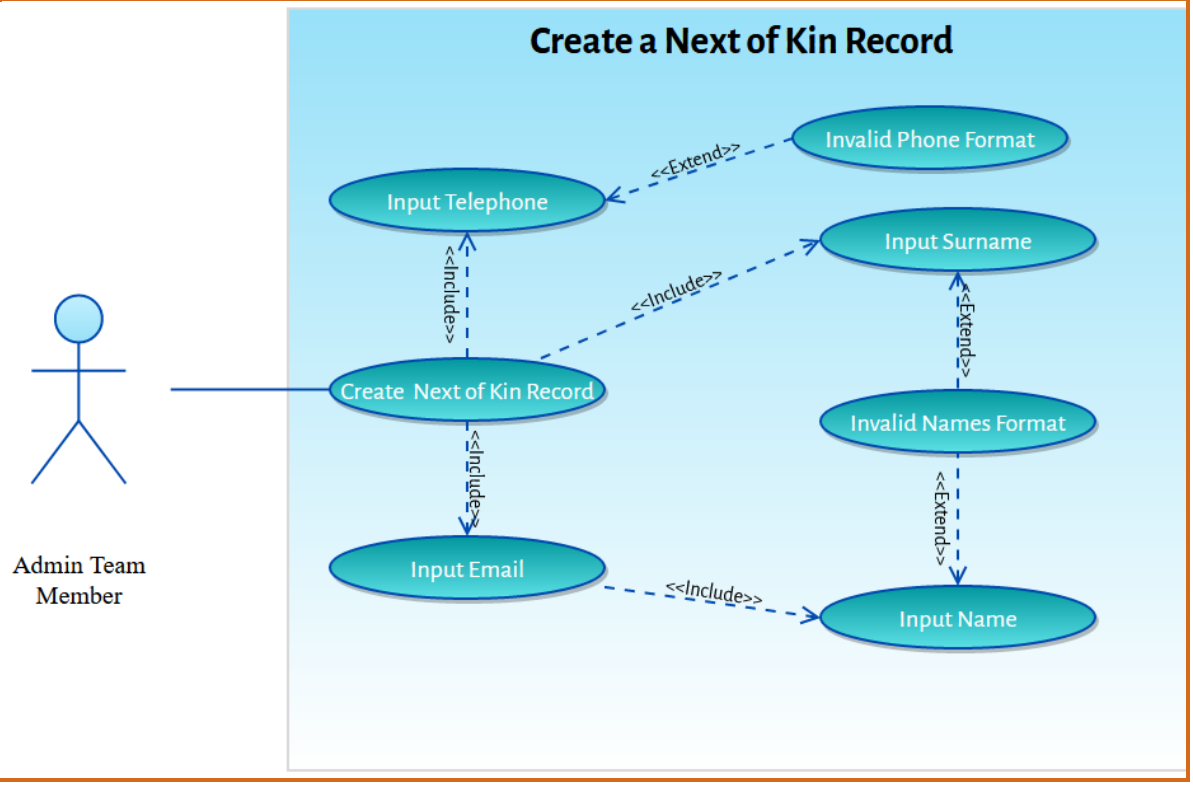



UC1a_Admin_Create_player - Part 2
Input Email
[Invalid Format]
[Valid Format]
Input Doctor's Details
[Invalid Format]
[Valid Format]
Calculate Age
[Under 18]
Add Consent Form
[Over 18]
Add Next of Kin
[File Ok]
[File too Large]

UC1a – Sequence Diagram



Use Case Name:	UC1a-1_Admin_Create_Next_of_Kin



Initiating Actor(s):	Admin Team	Receiving Actor(s):	

Trigger/Pre-condition(s):
Admin needs a new Next of Kin record for the new Player.

Main Flow of Events:
1) Admin inputs NoK's Name. 2) Admin inputs NoK's Surname. 3) Admin inputs NoK's Telephone. 4) Admin secretary inputs NoK's Email. 5) Check if the name or surname are valid. 6) Check if the Date of Birth is valid. 7) Check if Player already exists in the system. 8) Check if the Telephone is valid. 9) Check if the Email is valid. 10) End of Use Case

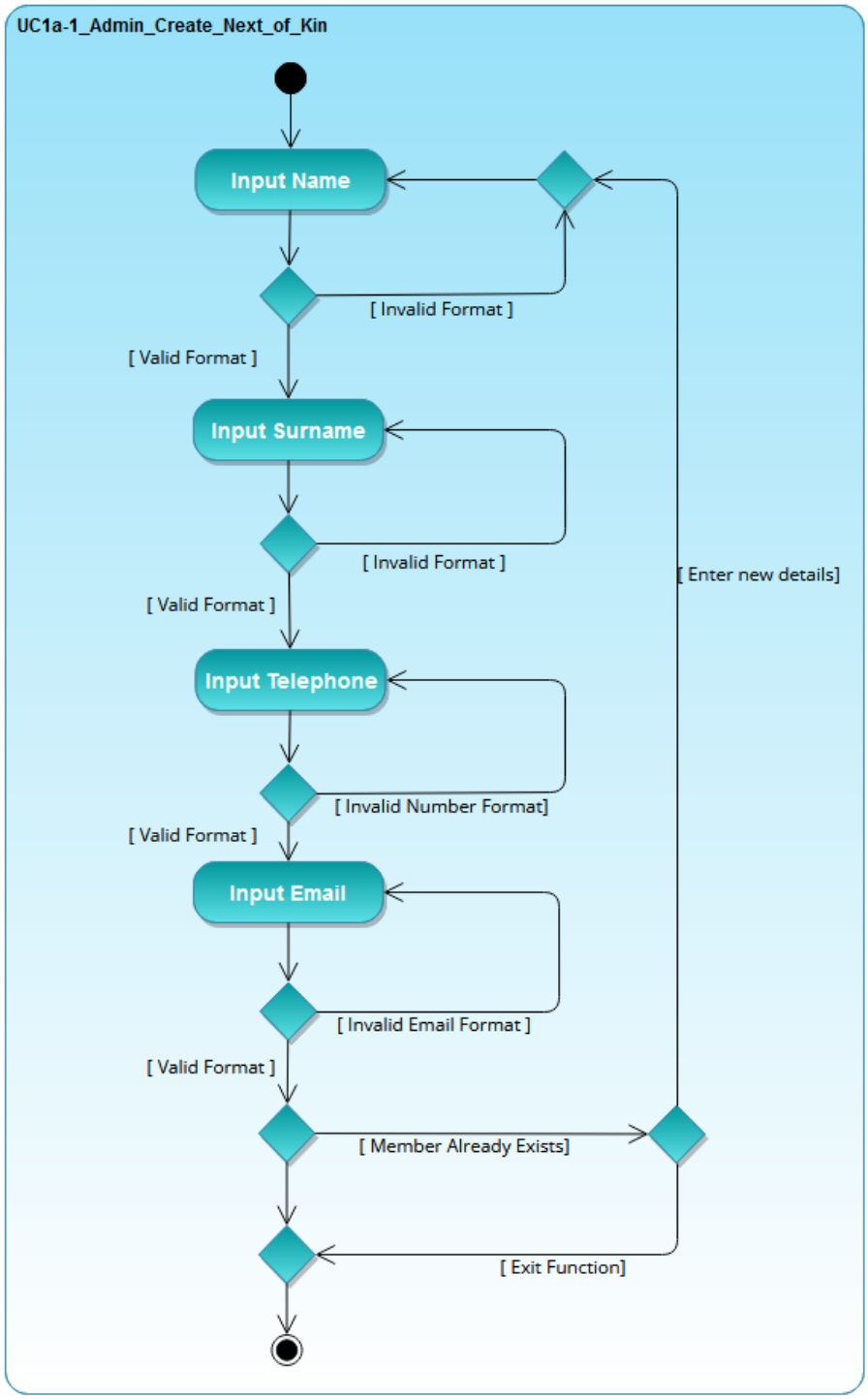
Alternate Flows:
1) Admin inputs NoK's Name. 2) Admin inputs NoK's Surname. 3) Admin inputs NoK's Telephone. 4) Admin secretary inputs member's Email. 5) Check if the name or surname are valid. 6) Check if the Date of Birth is valid.

7) Check if NoK's already exists in the system.
8) Check if the Telephone is valid.
9) Check if the Email is valid.
10) End of Use Case

Post-conditions:

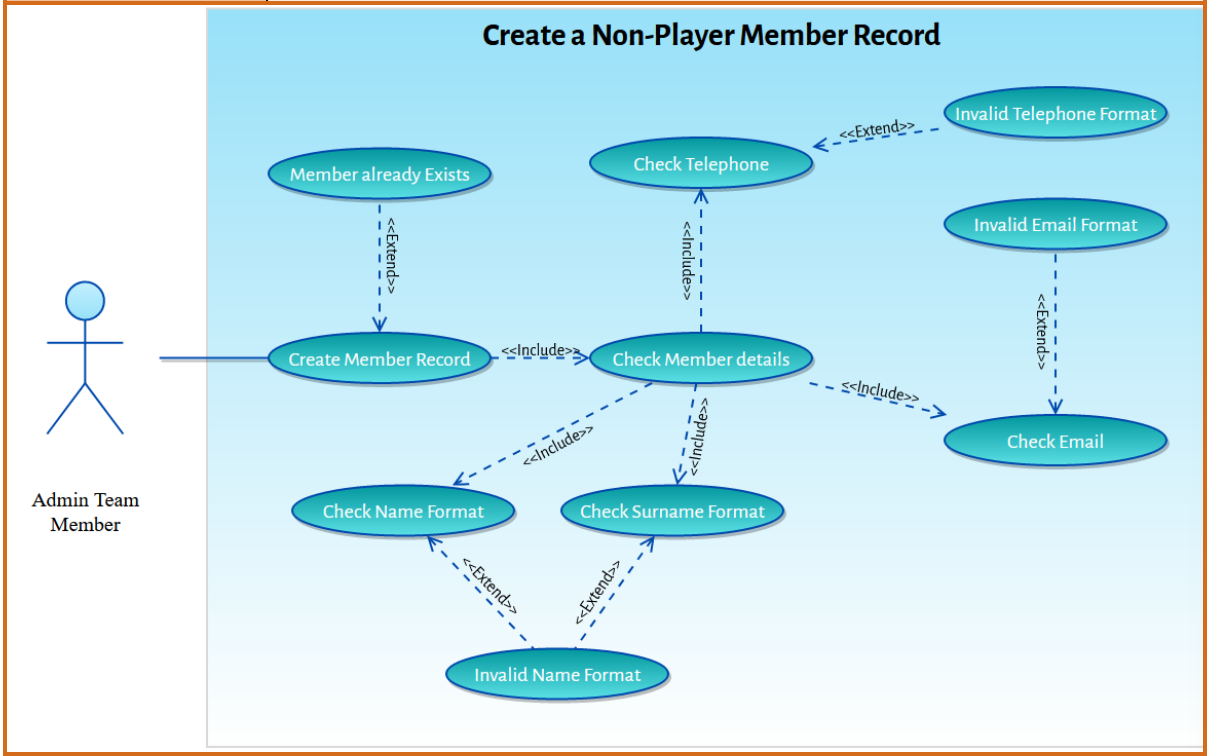
Next of Kin record saved.

UC1a-1_Admin_Create_Next_of_Kin



UC1a-1 – Sequence Diagram
Please refer to *UC1b – Sequence Diagram* from the next case as both follow the same sequences

Use Case Name:	UC1b_Admin_Create_NonPlaying_Member



See Appendix XX for full size Diagram

Initiating Actor(s):	Admin Team	**Receiving Actor(s):**	None

Trigger/Pre-condition(s):
a new non-playing member joined the club

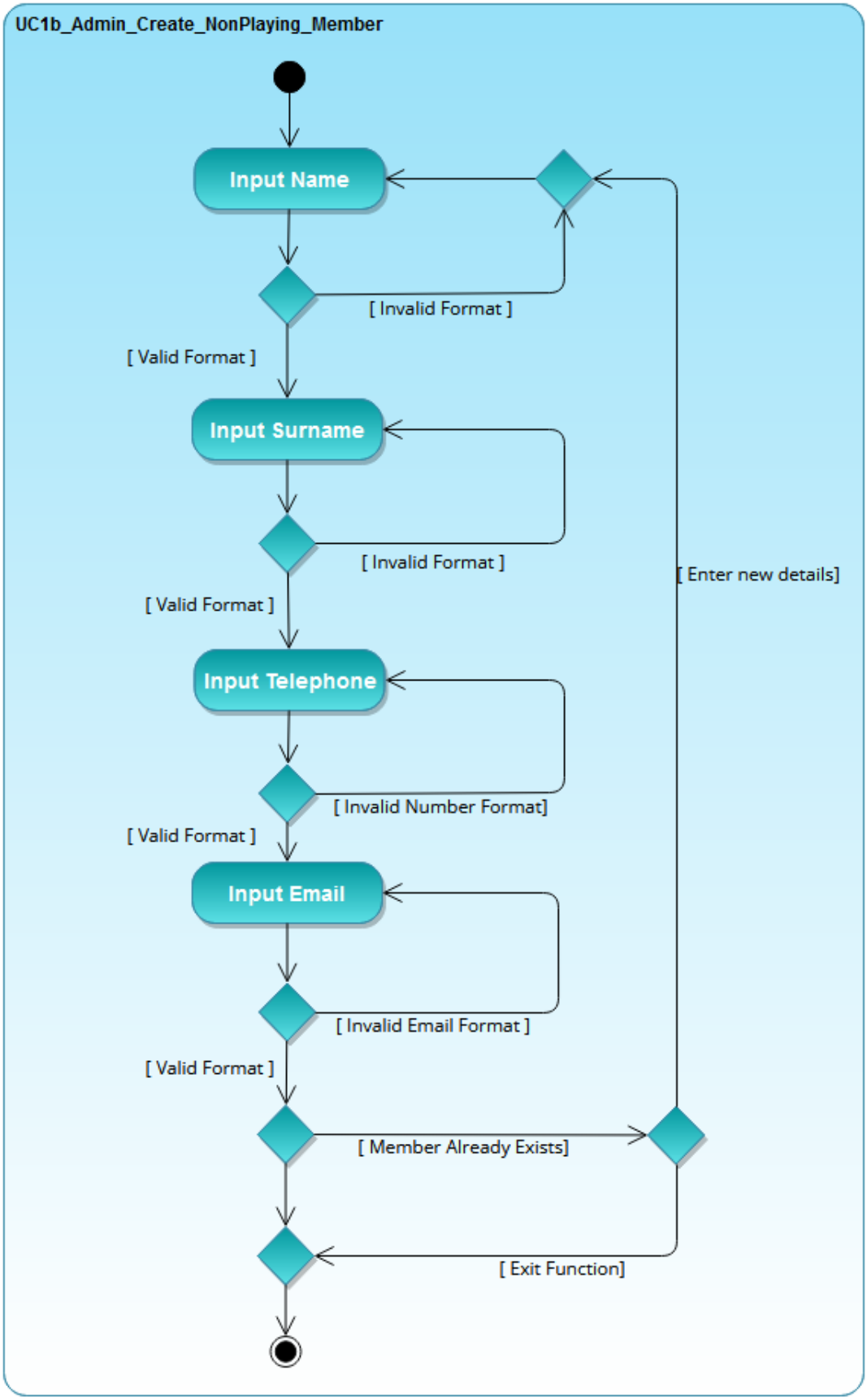
Main Flow of Events:
1) Admin inputs member's Name. 2) Admin inputs member's Surname. 3) Admin inputs member's Telephone. 4) Admin secretary inputs member's Email. 5) Check if the name or surname are valid. 6) Check if the Date of Birth is valid. 7) Check if Member already exists in the system. 8) Check if the Telephone is valid. 9) Check if the Email is valid. 10) End of Use Case

Alternate Flows:
5a. Invalid Name or Surname. 1) Display Error Message. 2) Input correct value. 3) Continue to next step. 6a. Invalid Date of Birth. 1) Display Error Message. 2) Input correct value. 3) Continue to next step. 7a. Member already Exists. 1) Display Error Message 2) Clear entered values. 3) Go back to step 1. 7b. Member already Exists. 1) Display Error Message 2) Secretary exits the function. 3) Exit Create Member function. 8a. Invalid Telephone number. 1) Display Error Message. 2) Input correct value. 3) Continue to next step. 9a. Invalid Email. 1) Display Error Message. 2) Input correct value. 3) Continue to next step.

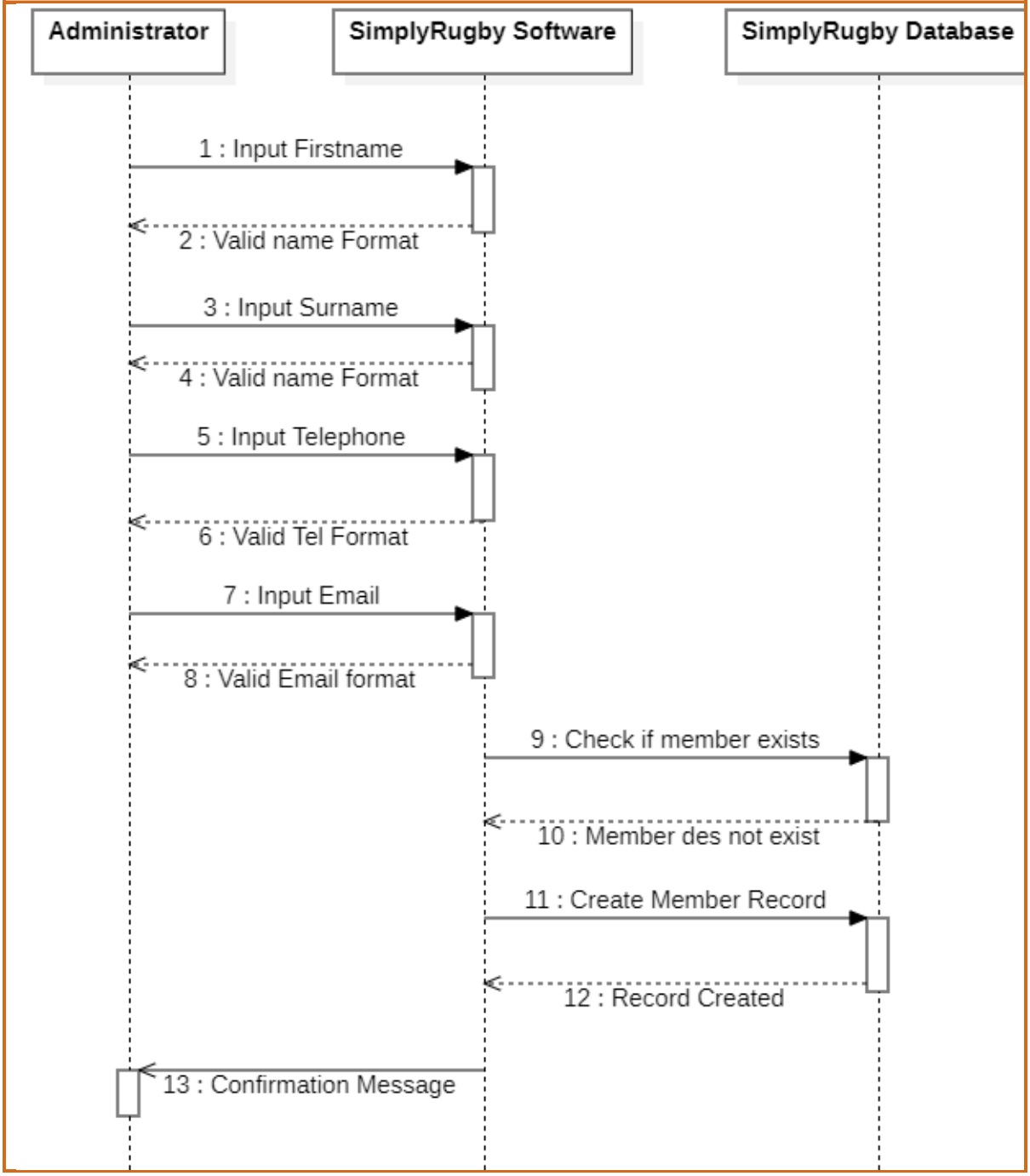
Assumptions(optional):
Members must be unique, so the system will check if the combination of name, surname and telephone are unique in the Database

Post-conditions:
A new Member record is created and saved in the database.

UC1b – Activity Diagram

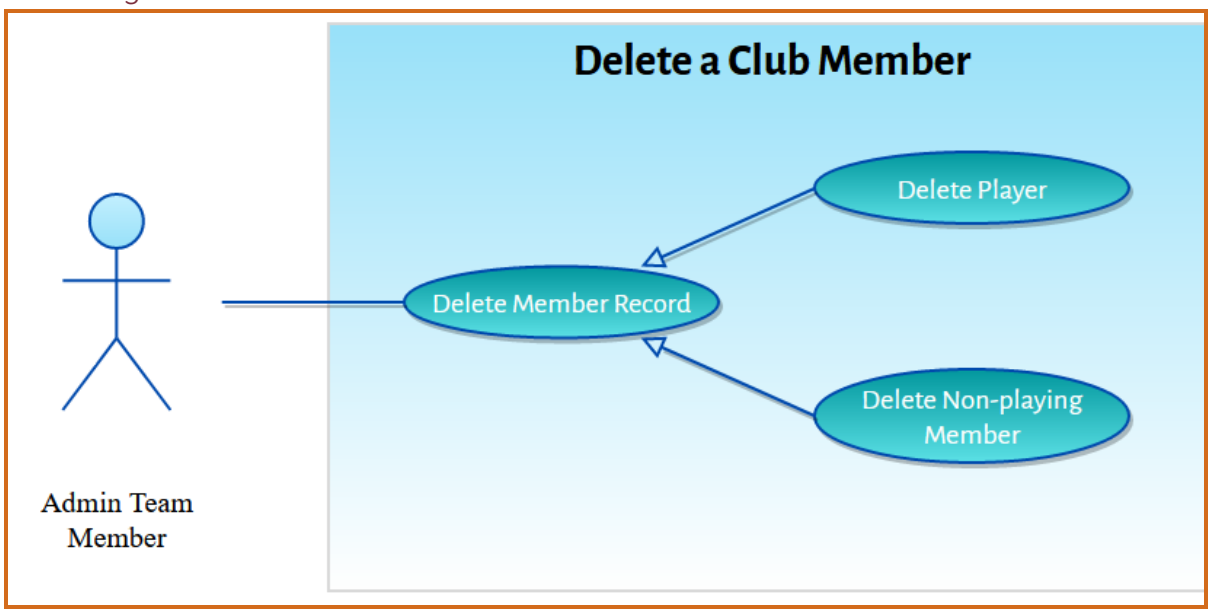


UC1b – Sequence Diagram

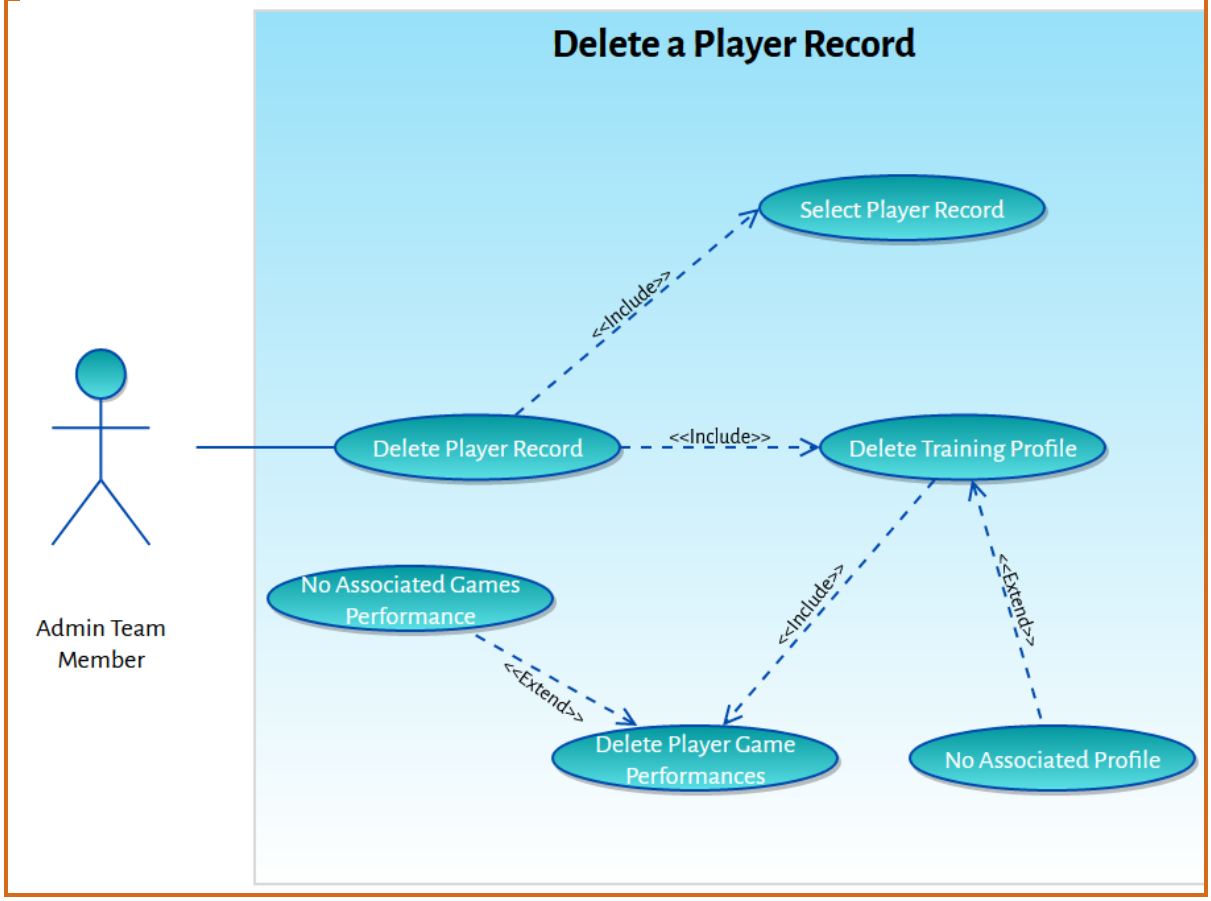


USE CASE 2: DELETE A MEMBER RECORD

UC2 Diagram



Use Case Name:	UC2a_Admin_Delete_Player_Record



Initiating Actor(s):	Admin	**Receiving Actor(s):**	None

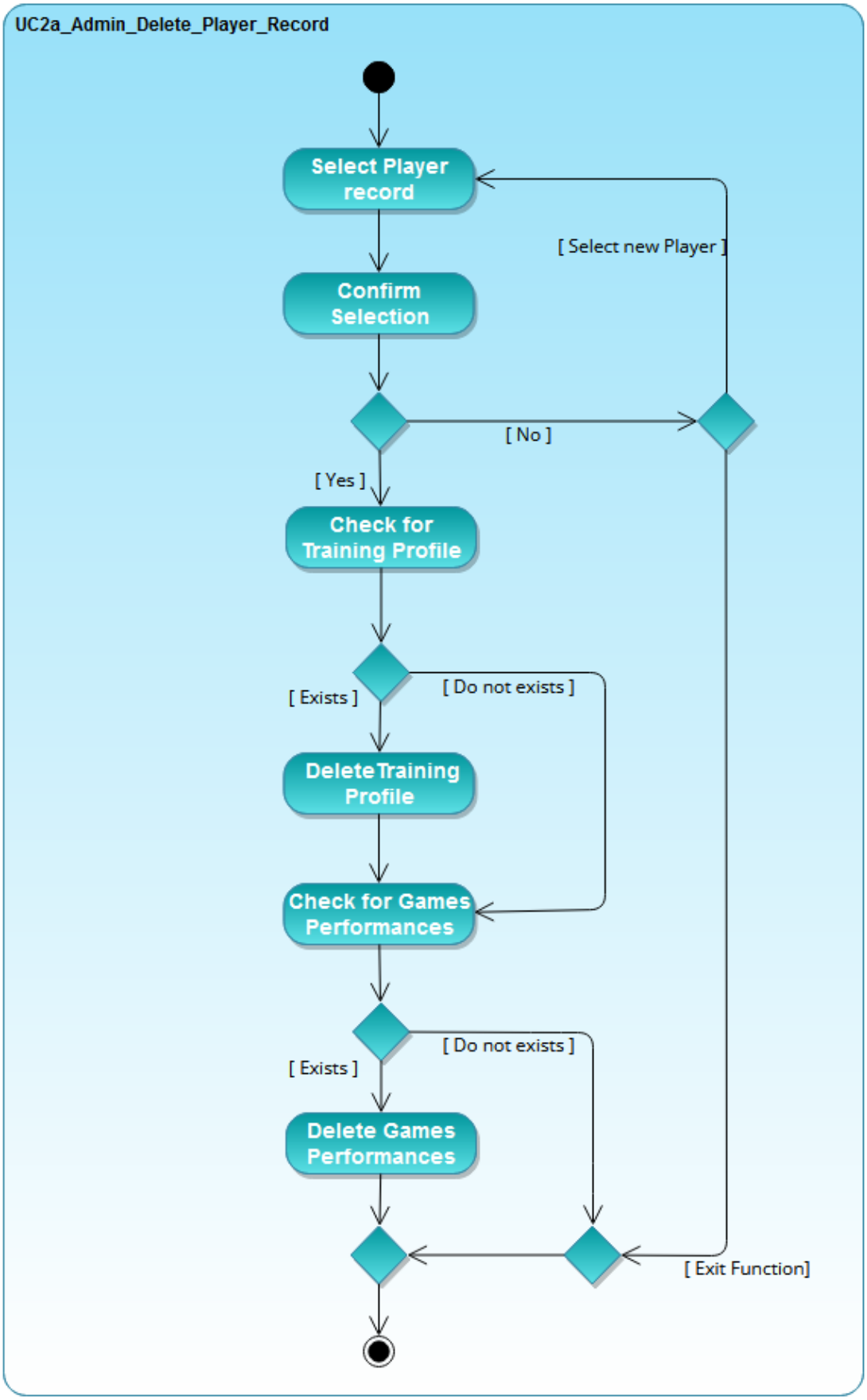
Trigger/Pre-condition(s):
Player officially left the club.

Main Flow of Events:
1) Admin selects the Player to delete from the list. 2) System asks for confirmation of deletion. 3) Admin confirms deletion. 4) System checks if a Training Profile exists. 5) Delete Training Profile. 6) Delete Player's Games performance. 7) End of Use Case

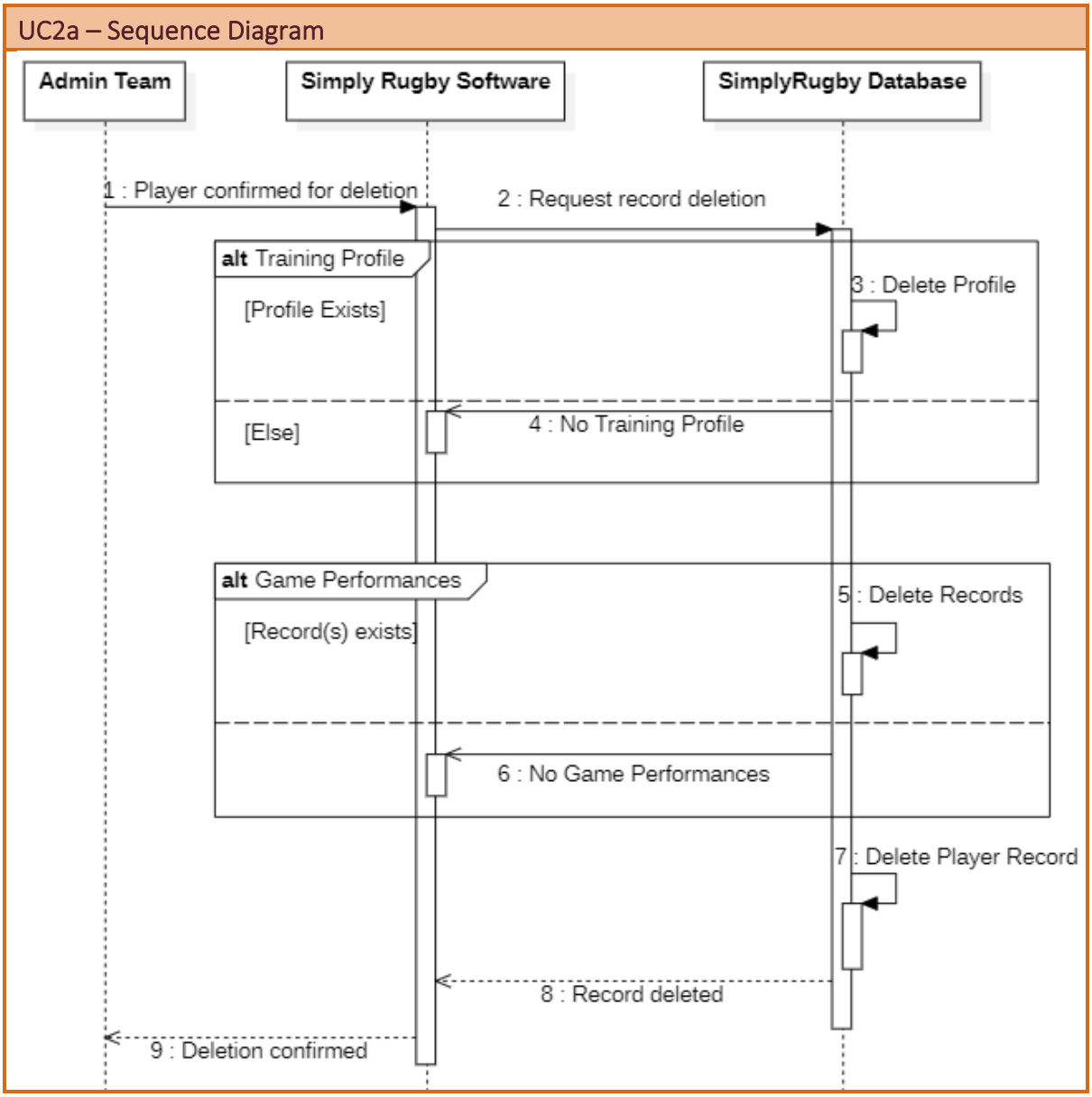
Alternate Flows:
2a. Admin cancel deletion request. 1) Exit Delete Player function. 2b. Admin cancel deletion request. 1) Select new Player Profile 4a. No Training Profile exists for that player. 1) Continue to step 7. 6a. No Game performances recorded for that Player. 1) Continue to step 7.

Post-conditions:
The Player and its related records are deleted form the system.

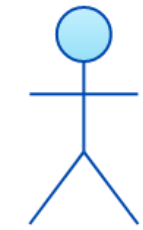
UC2a – Activity Diagram



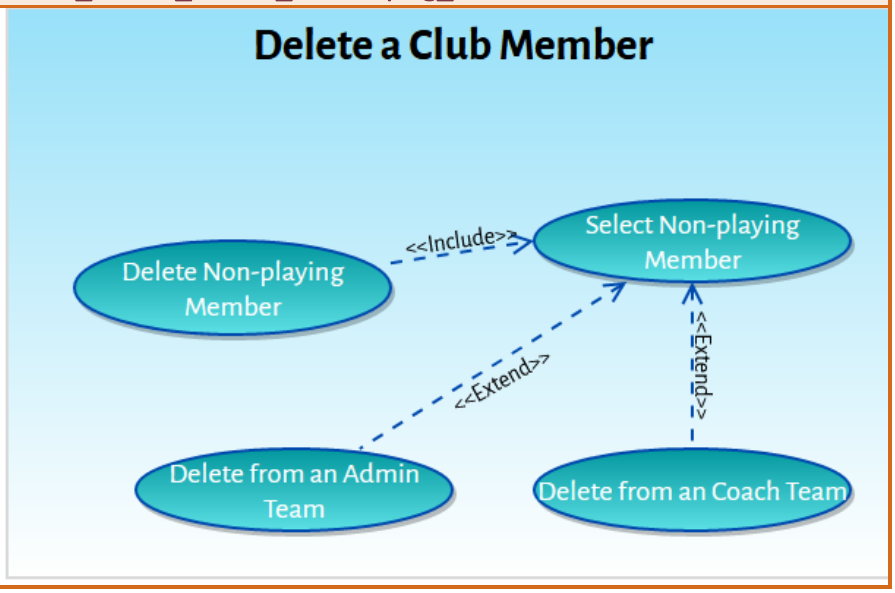
UC2a – Sequence Diagram



Use Case Name:	UC2b_Admin_Delete_NonPlaying_Member



Admin Team Member



Initiating Actor(s):	Admin	**Receiving Actor(s):**	None

Trigger/Pre-condition(s):

A non-playing member has left the club.

Main Flow of Events:

1) Admin selects the member to delete.
2) Admin confirms the deletion.
3) Check if member part of an Admin team
4) Check if member part of a Coach team
5) Member deleted from the system tables.
6) End of Use Case.

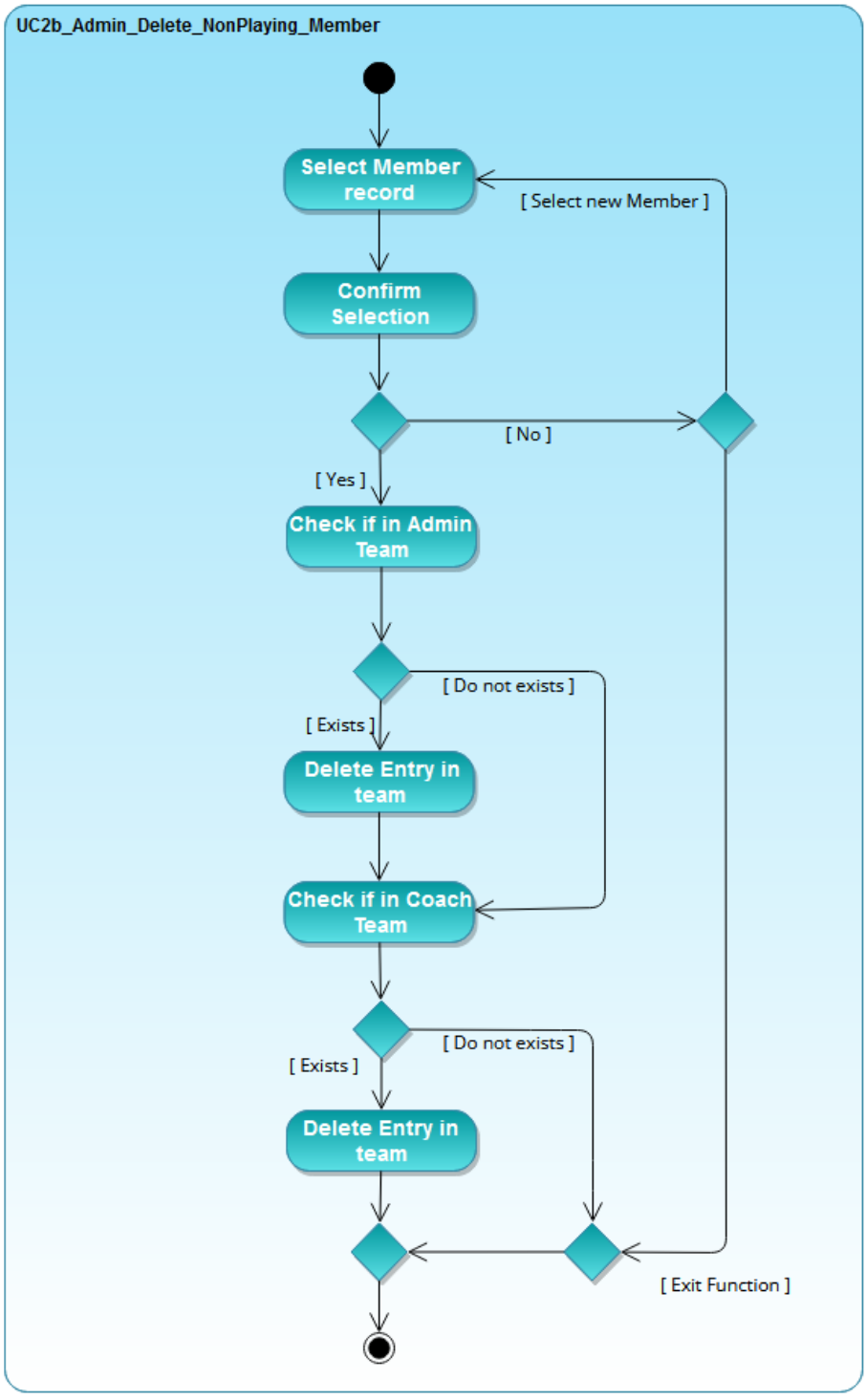
Alternate Flows:

2a. Admin cancel the deletion.
   1) Exit the Delete Member function.
2. Admin cancel the deletion.
   1) Select another Member.
   2) Proceed to next step.
3a. In an Admin Team
   1) Delete entry in the admin team.
   2) Continue to next step.
3b. In a Coach Team
   1) Delete entry in the coach team.
   2) Continue to next step.

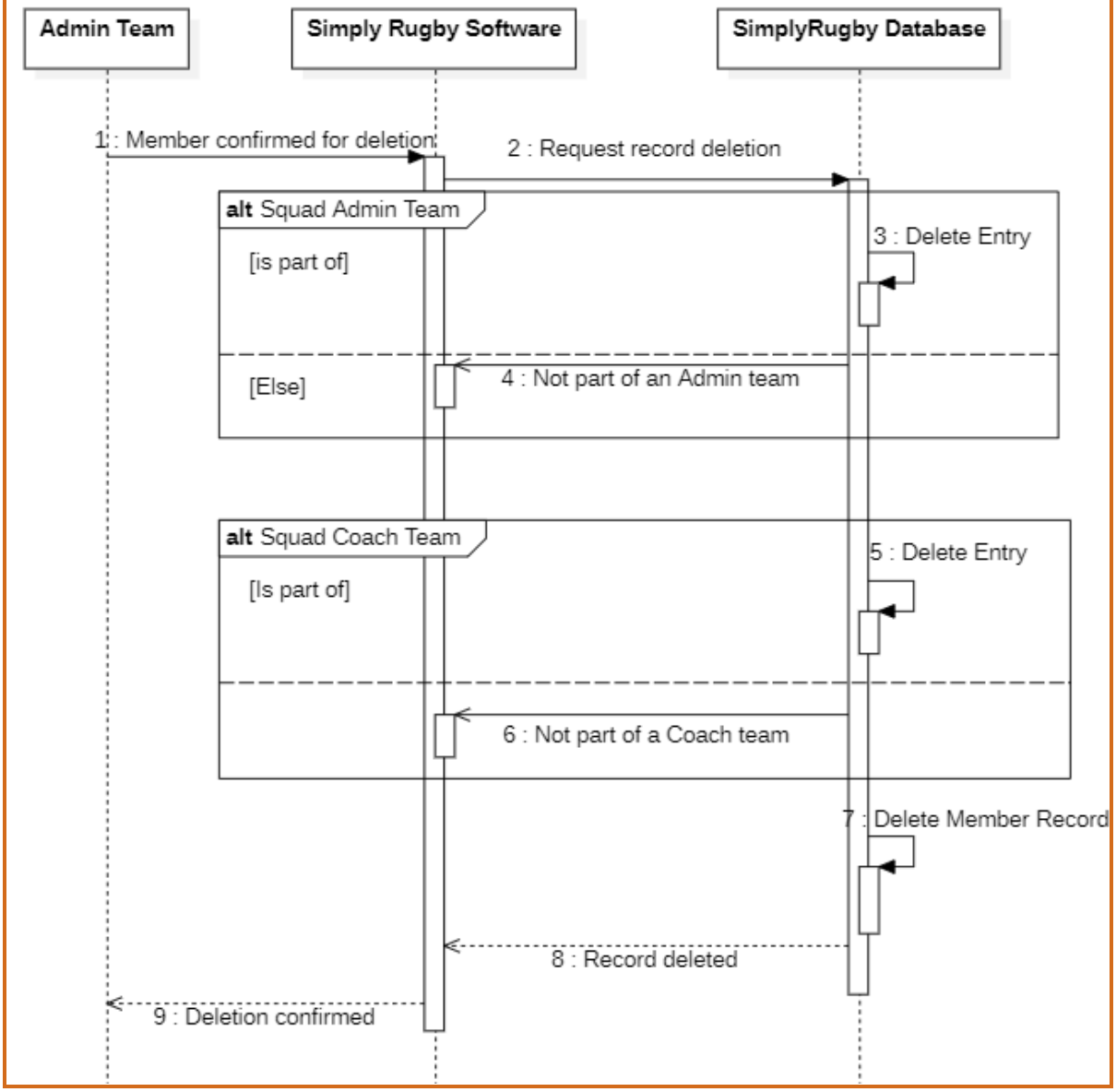
Post-conditions:

The member has been deleted from the system.

UC2b – Activity Diagram

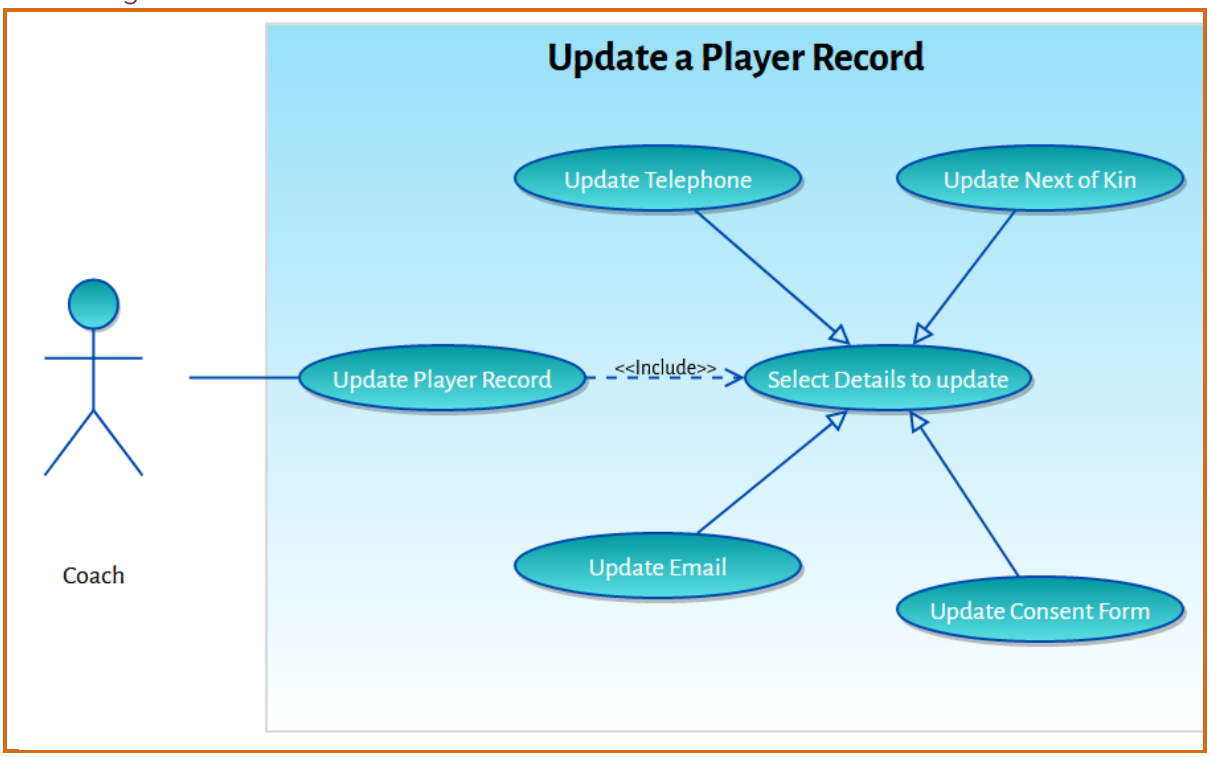


UC2b – Sequence Diagram

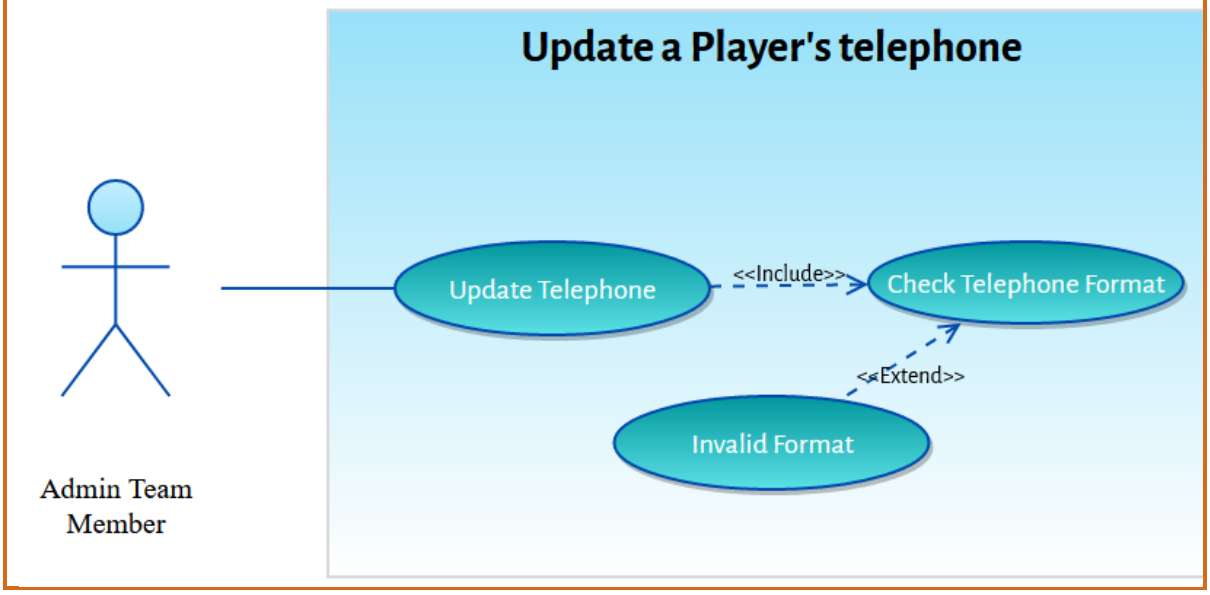


USE CASE 3: UPDATE A PLAYER RECORD

UC3 Diagram



Use Case Name:	UC3a_Admin_Update_Player_Record_Telephone



Initiating Actor(s):	Admin	Receiving Actor(s):	None

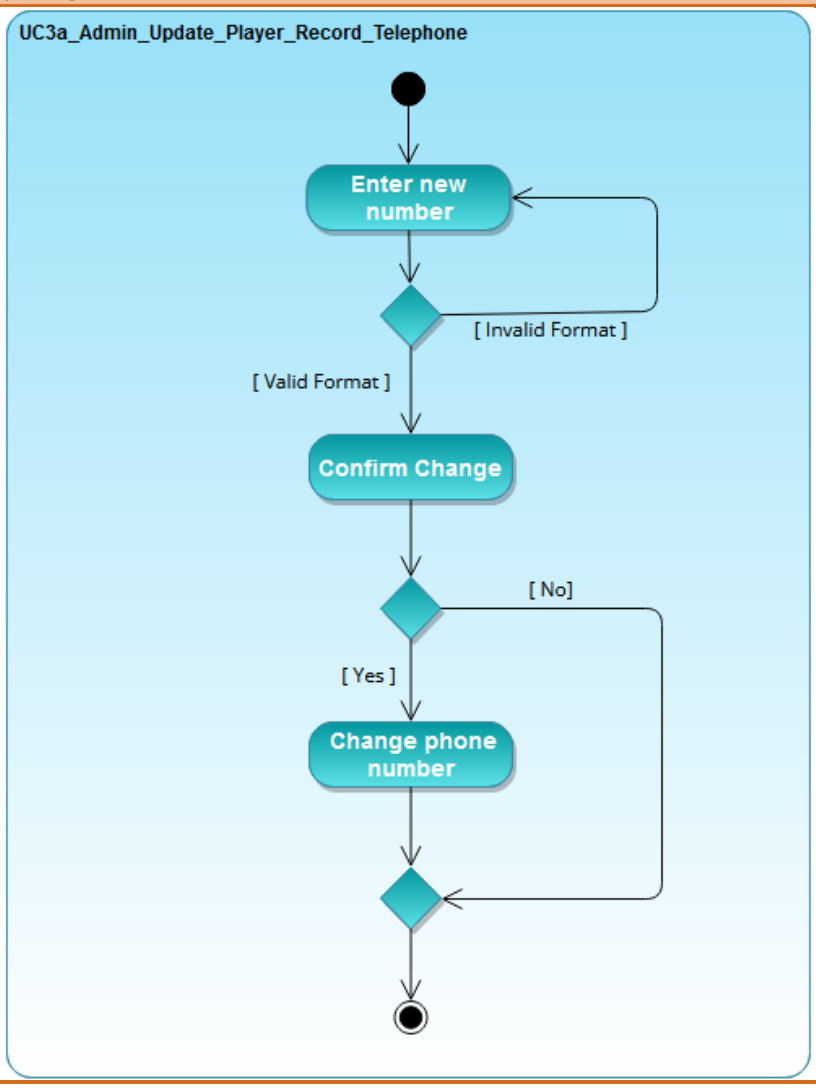
Trigger/Pre-condition(s):
A change of Telephone is requested.

Main Flow of Events:
1) Admin enters the new Telephone number. 2) Check if the value format is correct. 3) Confirm change 4) Replace the old Value by the new one. 5) End of Use Case.

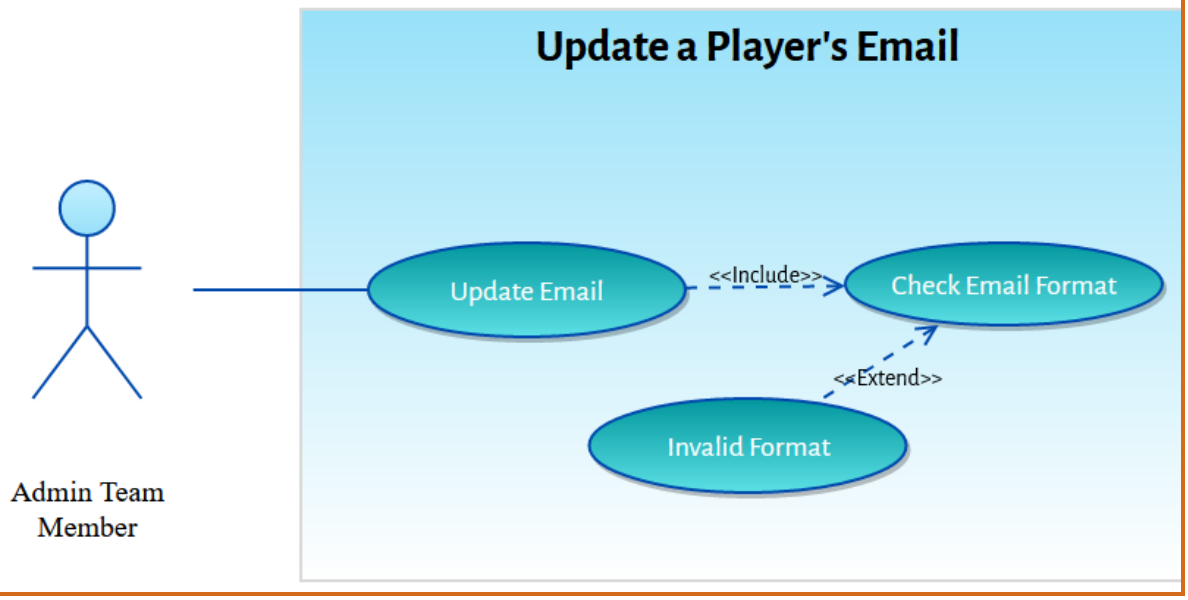
Alternate Flows:
2a. Invalid value entered. 1) Enter a new value. 2) Continue to next step. 3a. Cancel change 1) Exit Function

Post-conditions:
Telephone is updated

UC3a – Activity Diagram



Use Case Name:	UC3b_Admin_Update_Player_Record_Email



Initiating Actor(s):	Admin	**Receiving Actor(s):**	None

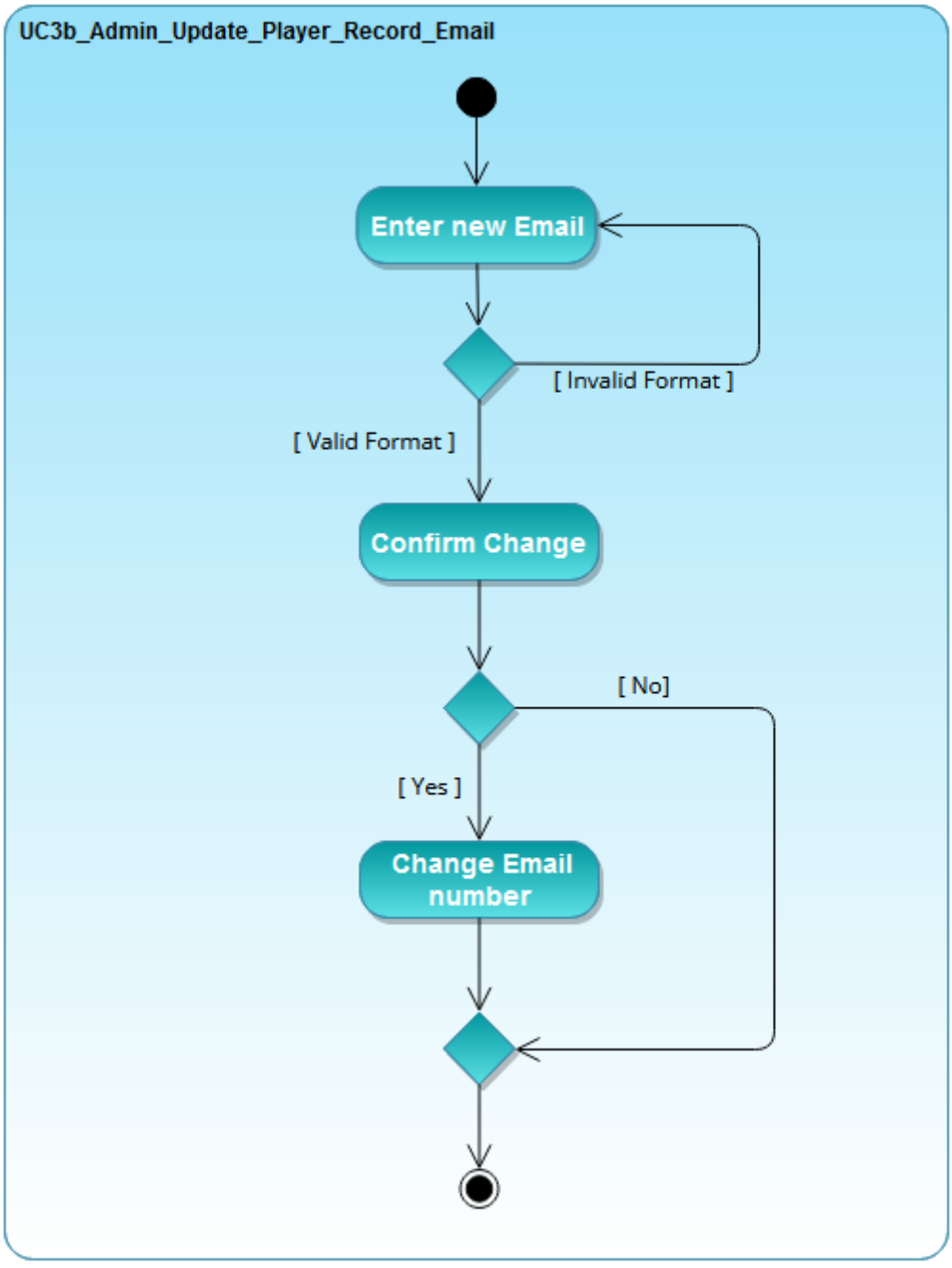
Trigger/Pre-condition(s):
A change of email is requested.

Main Flow of Events:
1) Admin enters the new email. 2) Check if the value format is correct. 3) Confirm change 4) Replace the old Value by the new one. 5) End of Use Case.

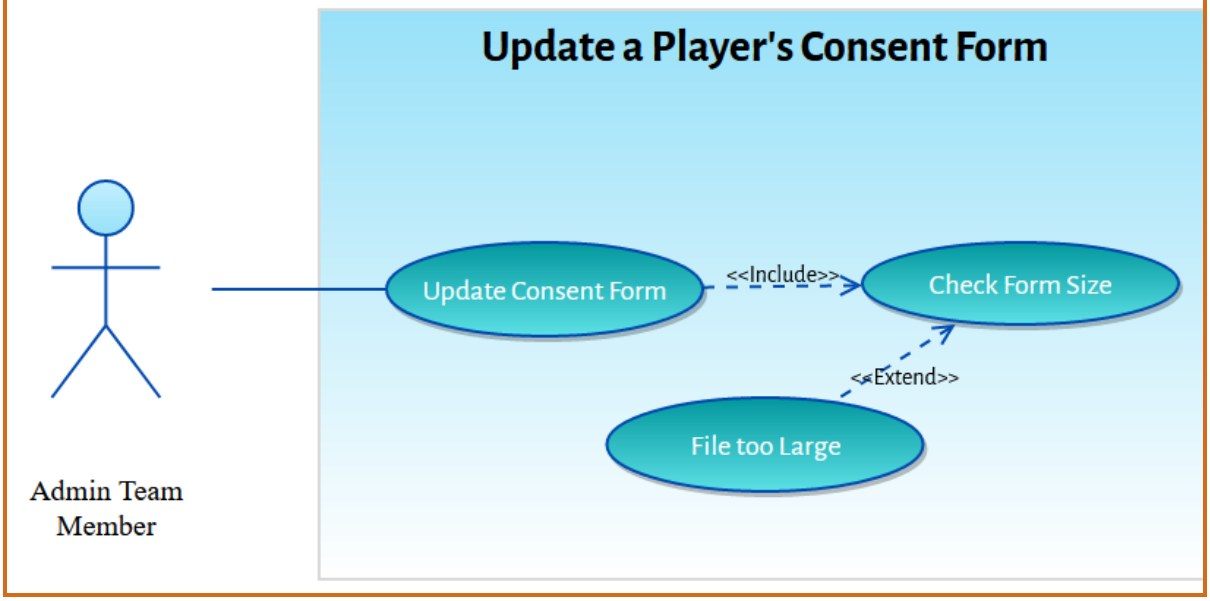
Alternate Flows:
2a. Invalid value entered. 1) Enter a new value. 2) Continue to next step. 3a. Cancel change 1) Exit Function

Post-conditions:
Email is updated

UC3b – Activity Diagram



Use Case Name:	UC3c_Admin_Update_Player_Record_Consent_Form



Initiating Actor(s):	Admin	Receiving Actor(s):	None

Trigger/Pre-condition(s):

An update of Consent Form is required. The admin has checked that the form has been properly documented and signed.

Main Flow of Events:

1) Admin selects the new Consent Form from the folder on the hard drive.
2) Systems checks the file size
3) Confirm the change
4) System uploads the file and save
4) End of Use Case.

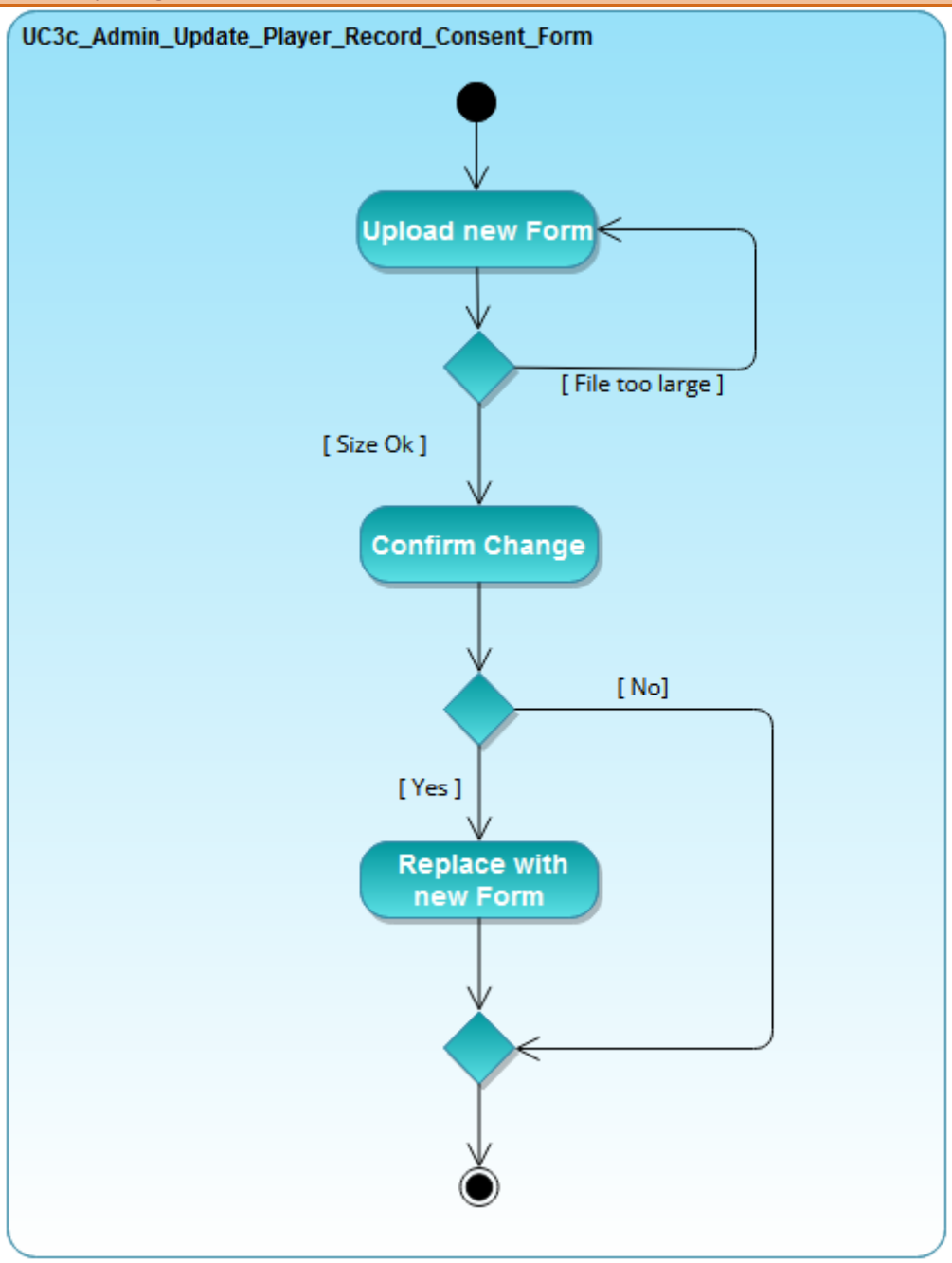
Alternate Flows:

2a. File too large
   1) Display error message.
   2) Admin edit and select the file again.
   3) Continue to next step.
3a. Cancel change
   1) Exit Function

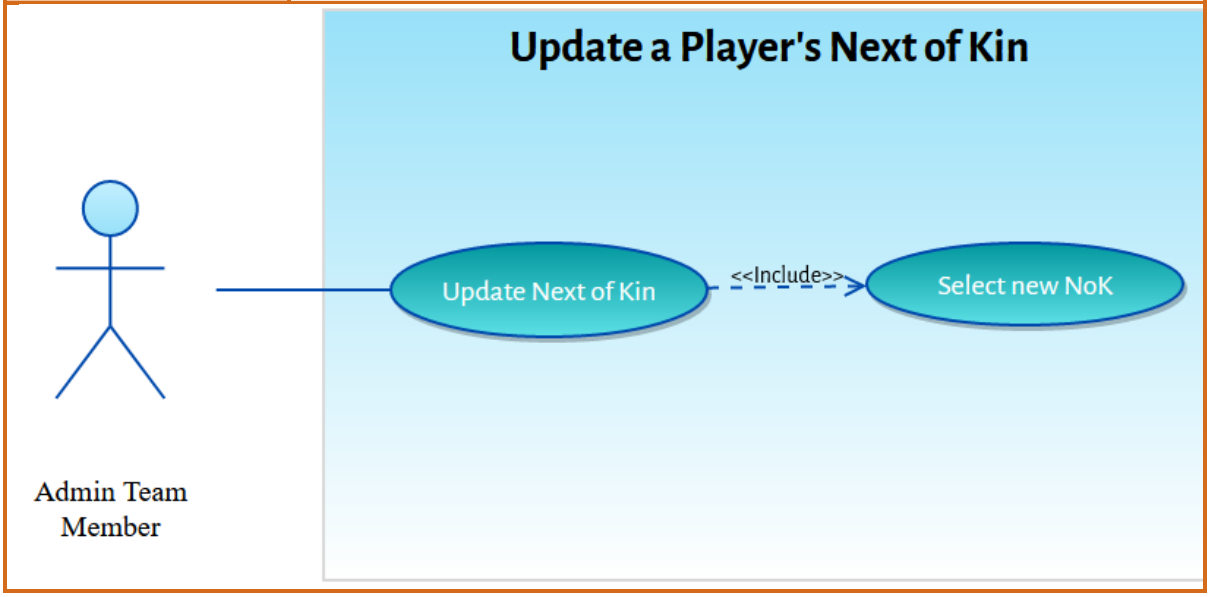
Post-conditions:

Consent Form is up to date.

UC3c – Activity Diagram



Use Case Name:	UC3d_Admin_Update_Player_Record_Next_of_Kin



Initiating Actor(s):	Admin	Receiving Actor(s):	None

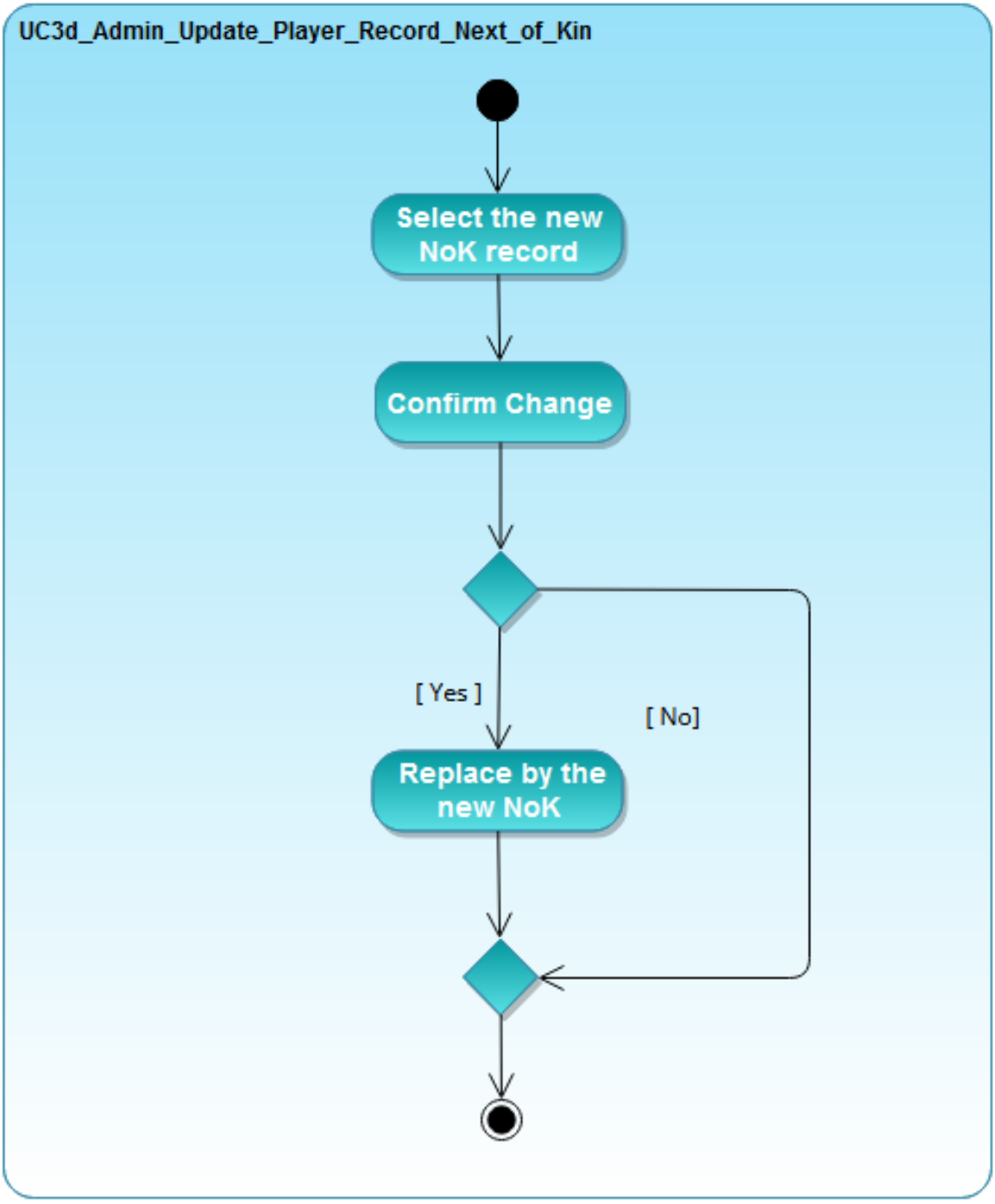
Trigger/Pre-condition(s):
Change of Next of Kin requested. The new NoK has been created

Main Flow of Events:
1) Admin selects the Player to update. 2) Admin selects the new Next of Kin. 3) Admin confirm the change. 4) End of Use Case.

Alternate Flows:
3a. Admin cancels the change. 1) Exit the Update NoK function.

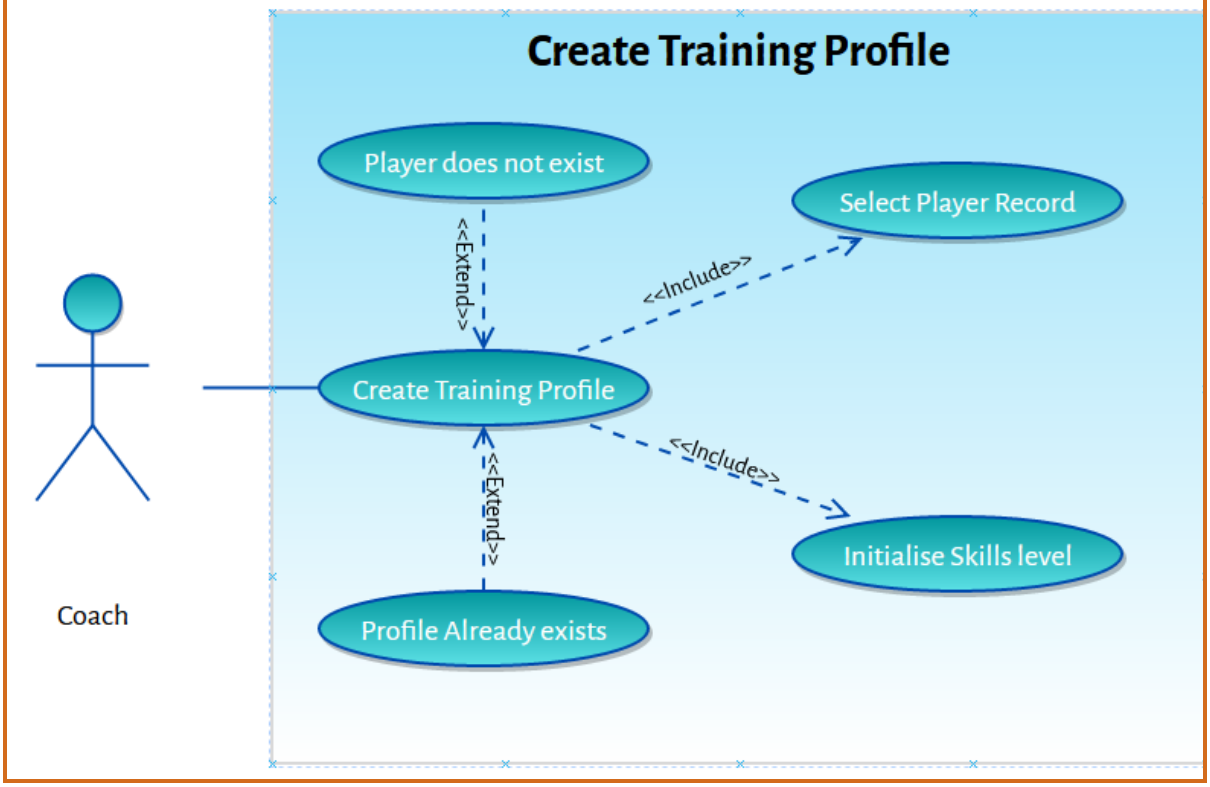
Post-conditions:
The Next of Kin has been updated.

UC3d – Activity Diagram



USE CASE 4: CREATE A TRAINING PROFILE

Use Case Name:	UC4_Coach_Create_Training_Profile



Initiating Actor(s):	Coach	Receiving Actor(s):	None

Trigger/Pre-condition(s):
A new Player has been created in the system after joining by the admin team. The relevant Coach has been notified.

Main Flow of Events:
1) Coach selects the new Player record. 2) Check if a training profile for the player already exists. 3) Coach adjusts the 5 core skills if needed. 4) Coach confirms creation 5) System saves the profile.

Alternate Flows:
1a. Player not found in the list. 1) Display error message to create the profile. 2) Exit Create Training profile function. 2a. Training Profile already in the system. 1) Display error message. 2) Exit Create Training profile function. 2b. Training Profile already in the system.

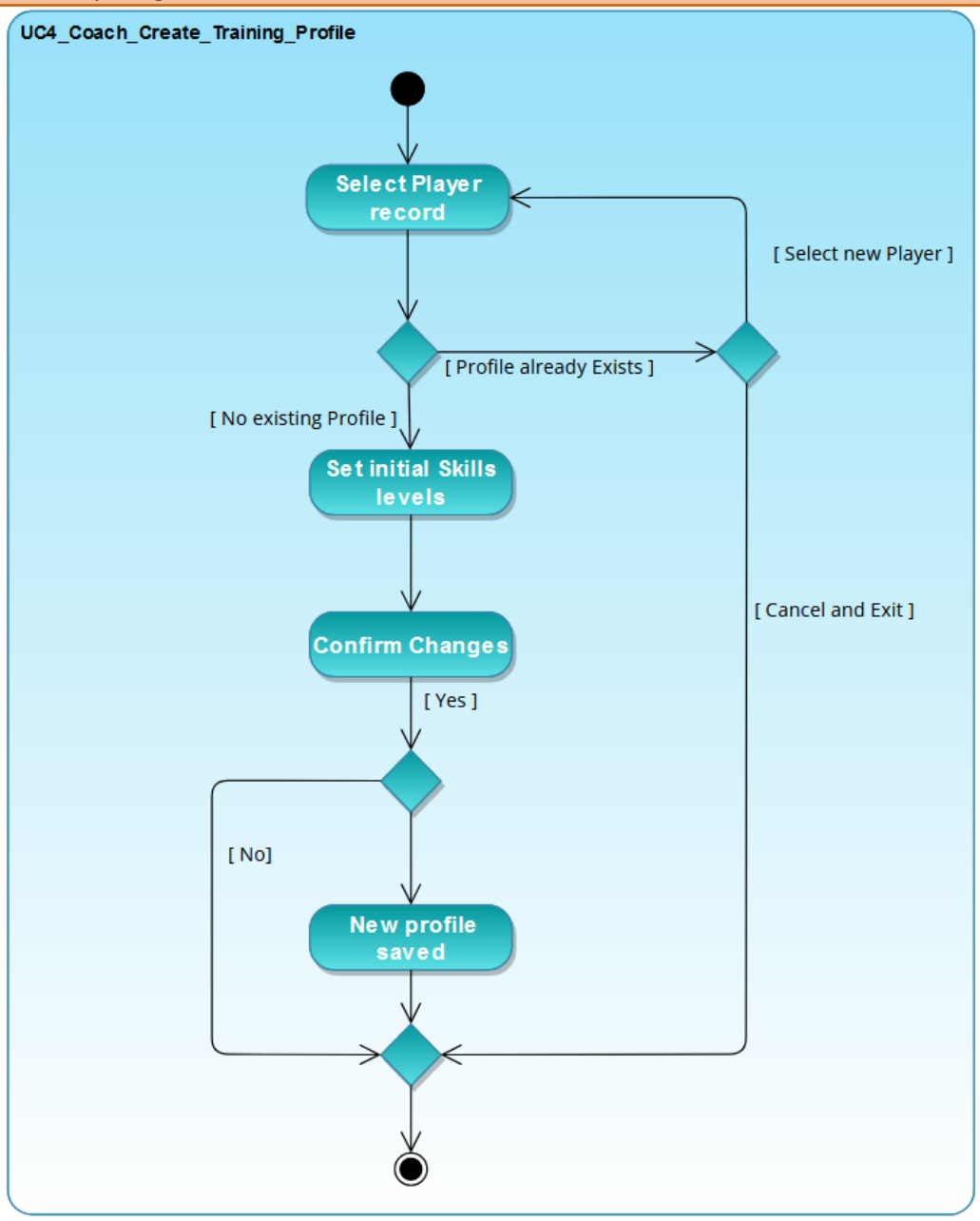
1) Select new Player record.
2) Continue to next step.

4a. Coach cancels.
1) Exit Create Training profile function.

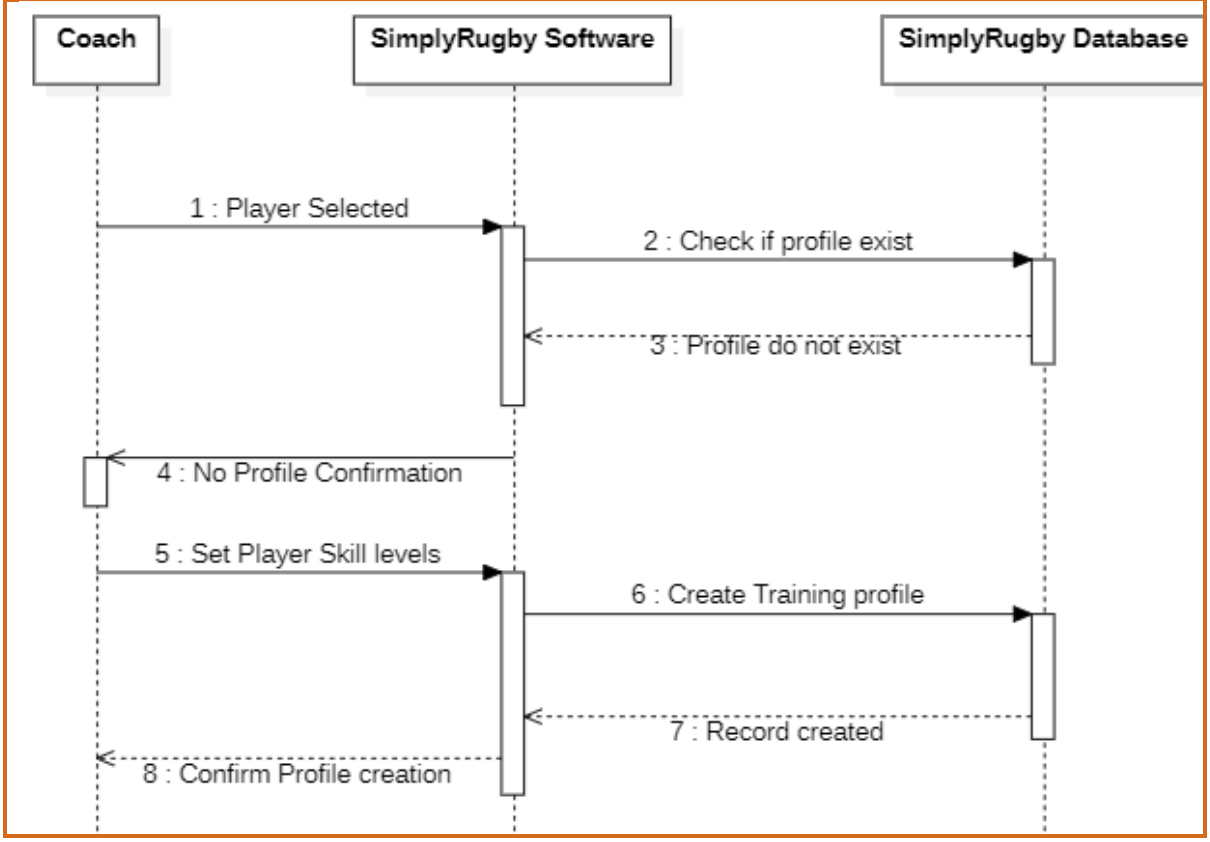
Post-conditions:

The Player's Training profile is created and saved in the system.

UC4 – Activity Diagram

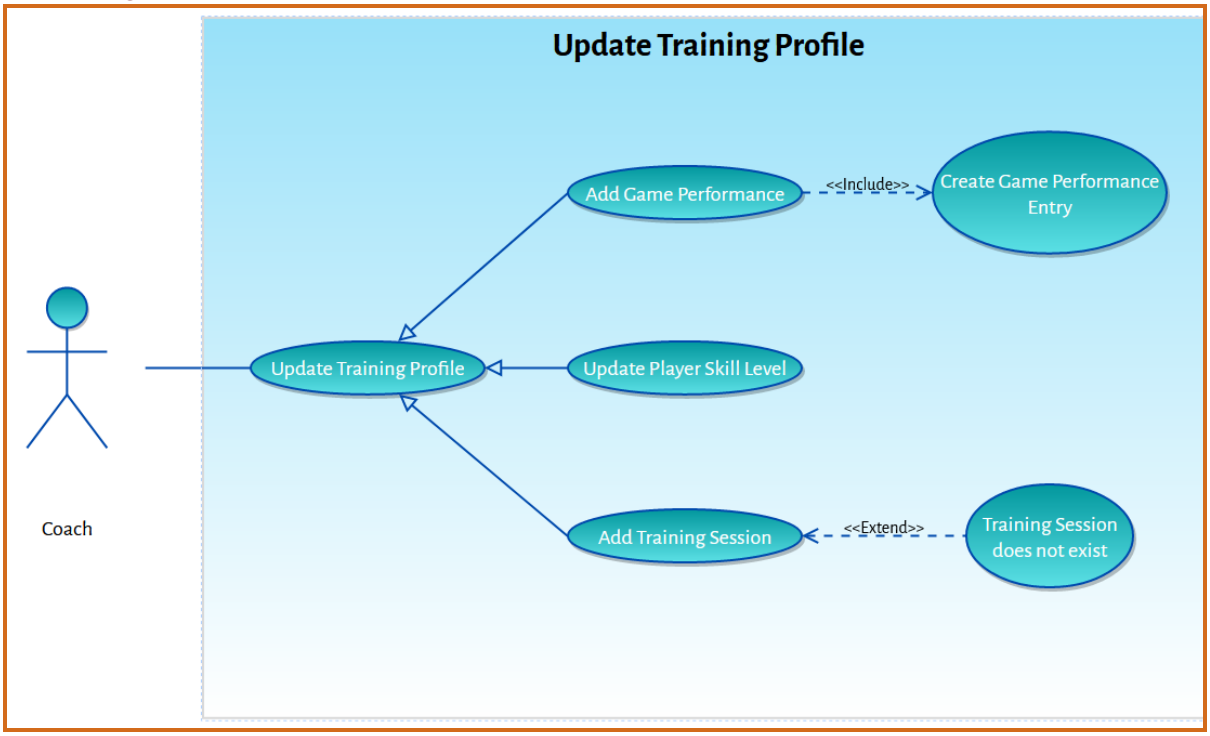


UC4 – Sequence Diagram

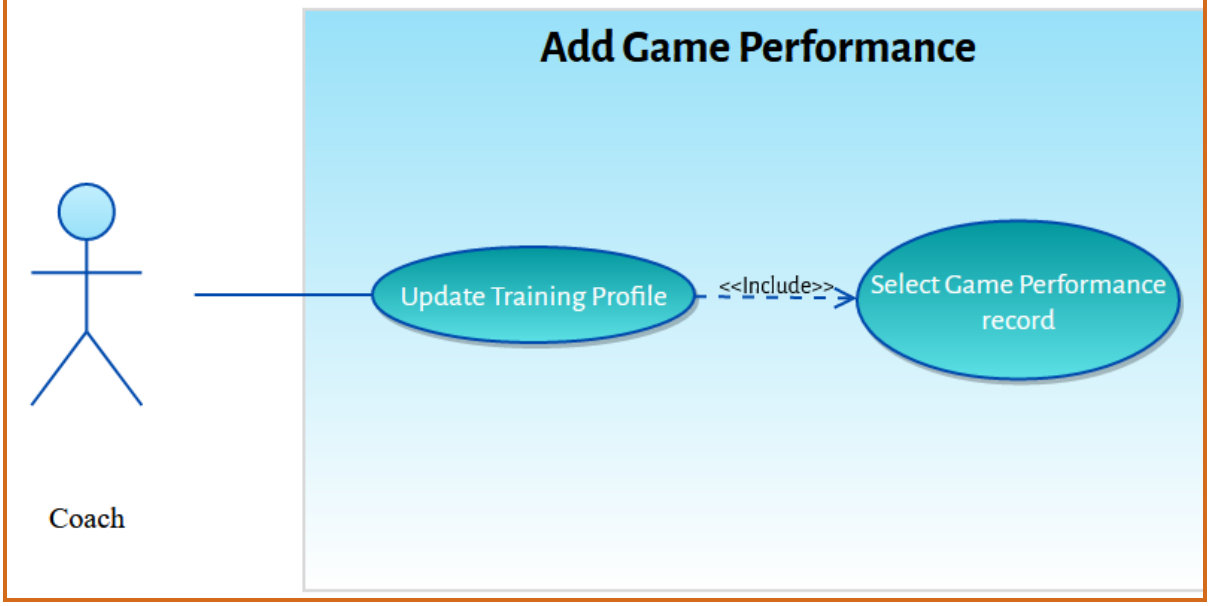


USE CASE 5: UPDATE A TRAINING PROFILE

UC5 Diagram



Use Case Name:	UC5a_Coach_Update_Training_Profile_Game_Performance



Initiating Actor(s):	Coach	Receiving Actor(s):	None

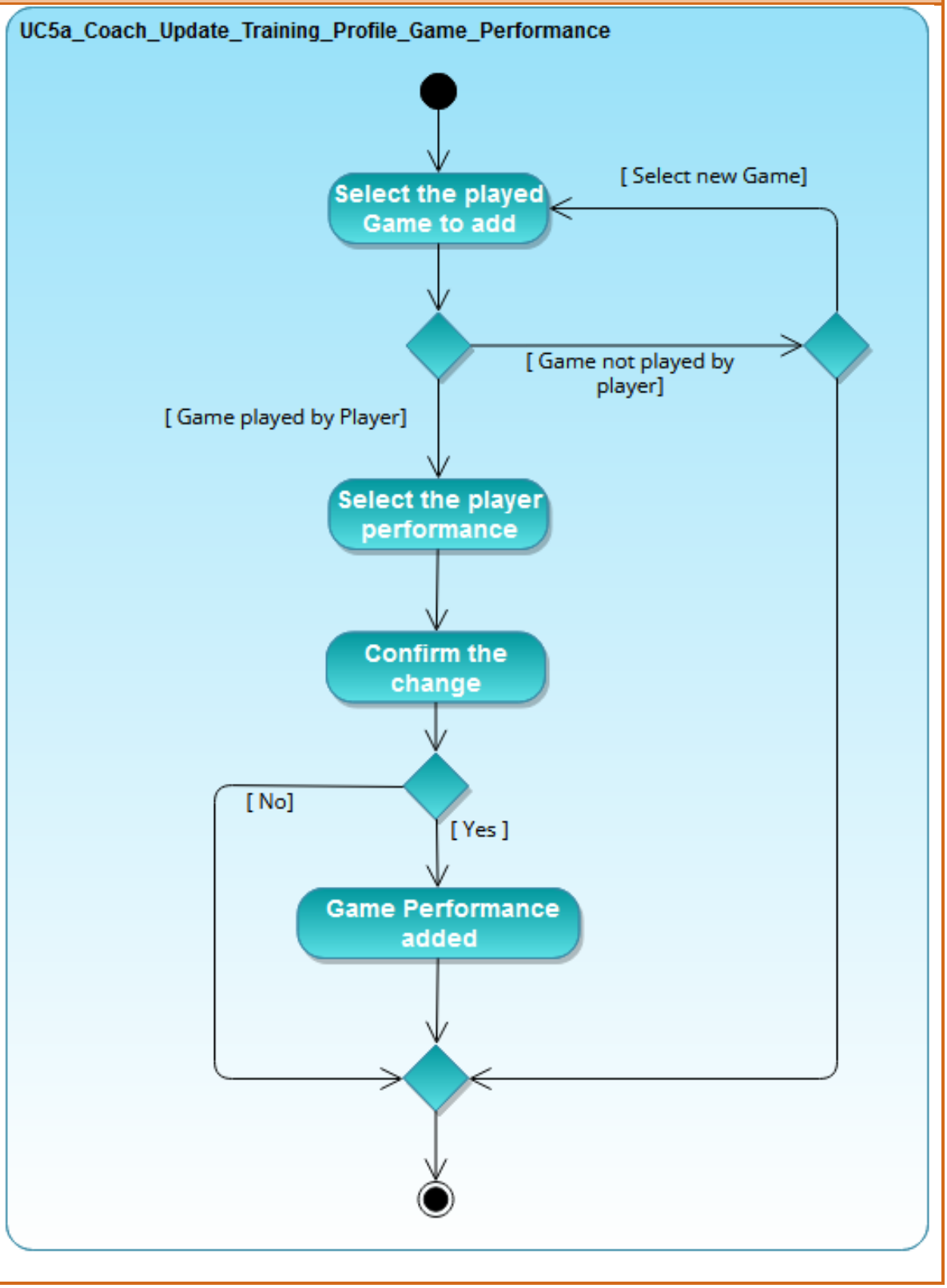
Trigger/Pre-condition(s):
The game to add to the training profile has been played by the Squad and documented by the Squad Secretary, the Coach has selected the correct Player Training profile

Main Flow of Events:
1) Coach selects a Game to add to the profile. 2) Check if the player played that game. 3) Coach selects the Player's performance from a dropdown list. 4) Coach confirms the change. 5) The system will create a Game Performance record for that player. 5) End of Use Case.

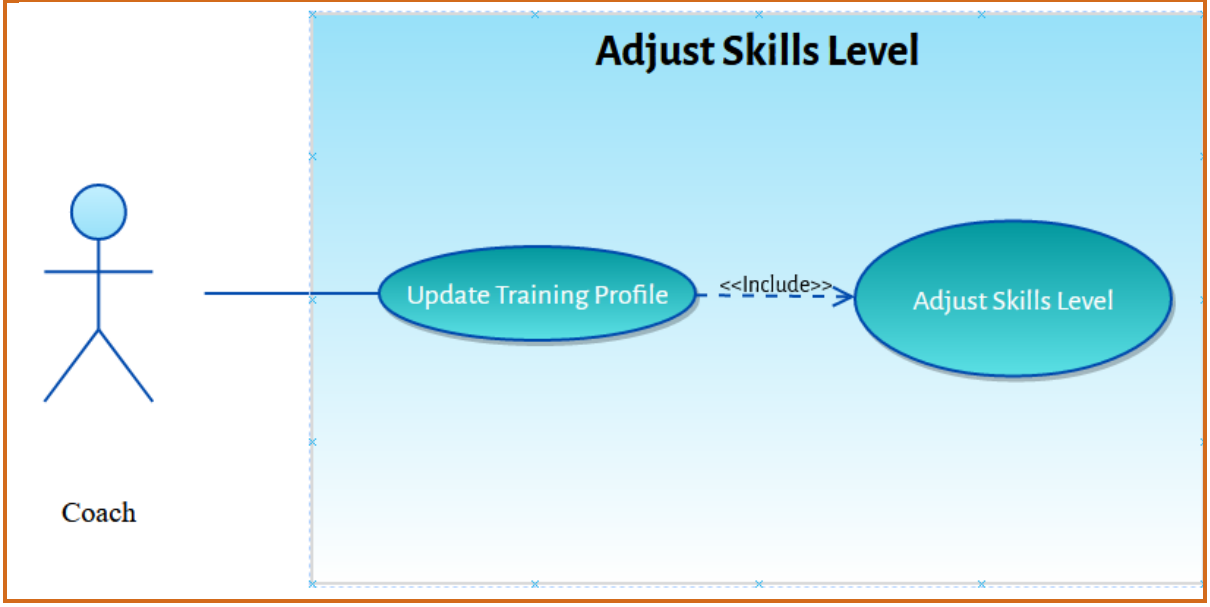
Alternate Flows:
2a. Player did not play that game. 1) Display error message. 2) Coach selects new Game. 3) Continue to next step. 2b. Player did not play that game. 1) Display error message. 2) Coach cancel function. 3) Exit from function.

Post-conditions:
Player's Game Performance added to his profile.

UC5a – Activity Diagram



Use Case Name:	UC5b_Coach_Update_Training_Profile_Skills_Levels



Initiating Actor(s):	Coach	Receiving Actor(s):	None

Trigger/Pre-condition(s):
Coach has determined that the player has improved or lost proficiency in some of the core skills.

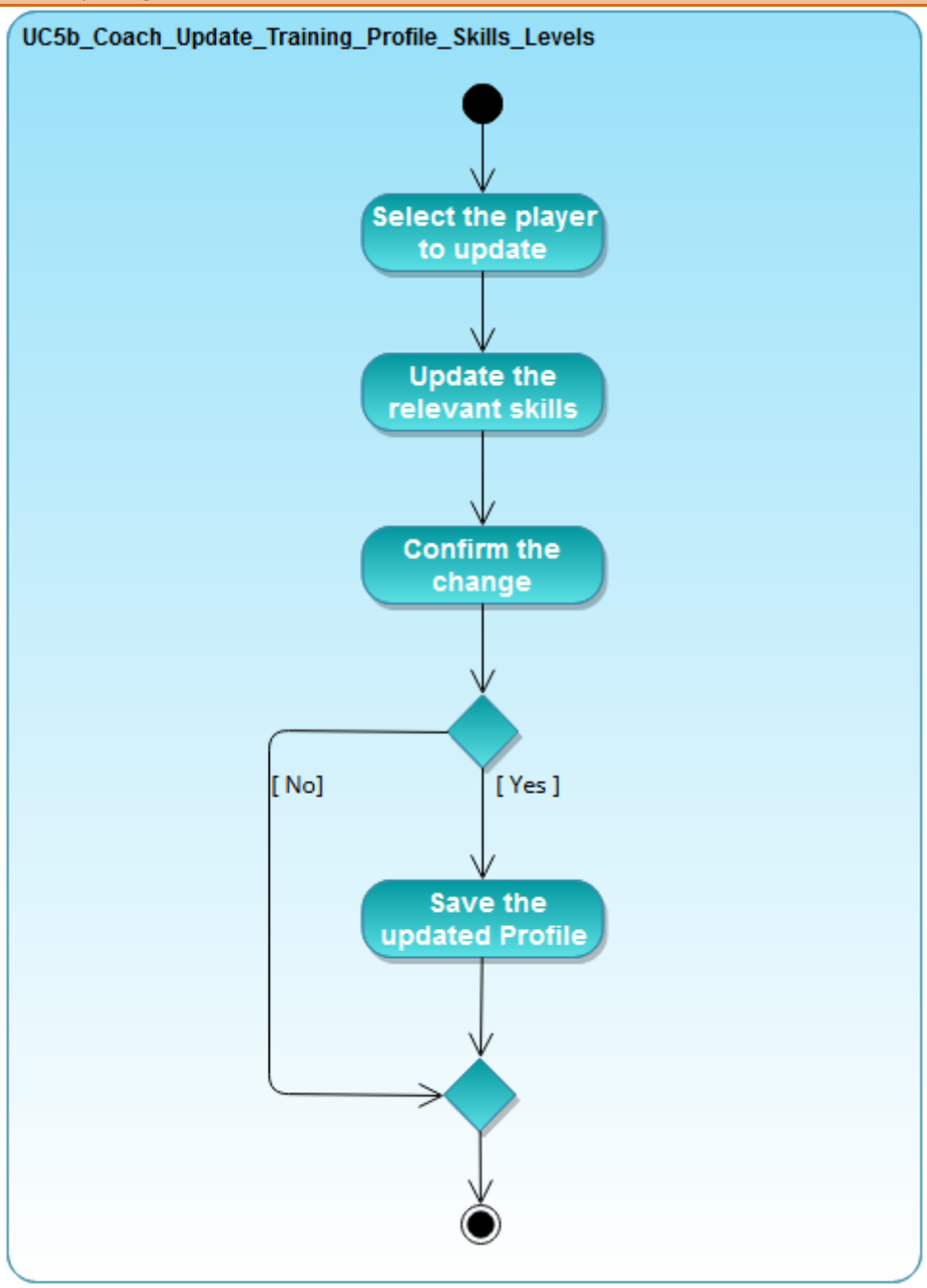
Main Flow of Events:
1) Coach selects Player to update. 2) Coach adjusts the relevant skill(s) level. 3) Coach confirms the changes. 4) End of Use Case

Alternate Flows:
3a. Coach cancel the changes. 1) Exit from Update Training Profile function.

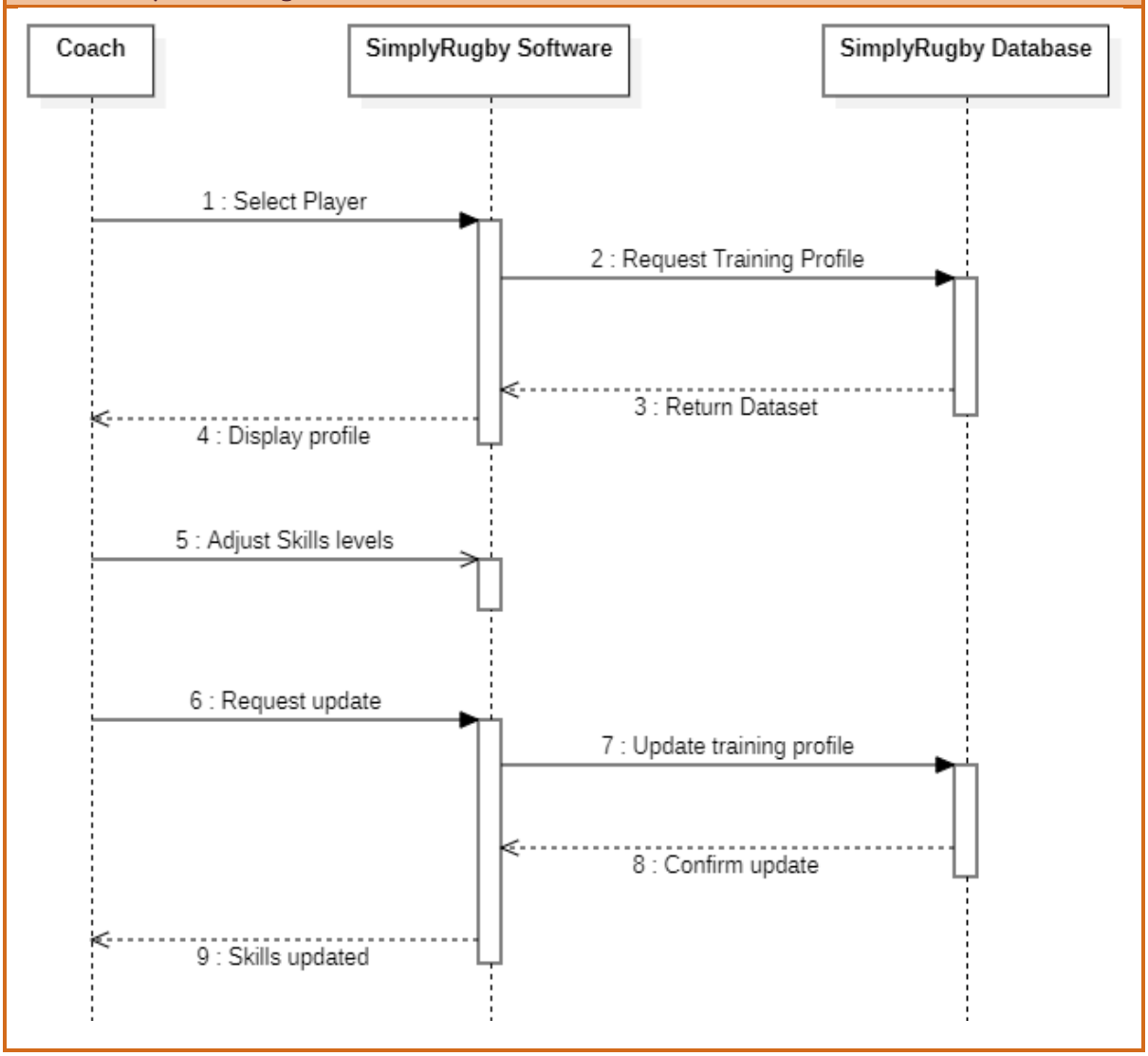
Assumptions(optional):
None

Post-conditions:
Player's skills level is updated.

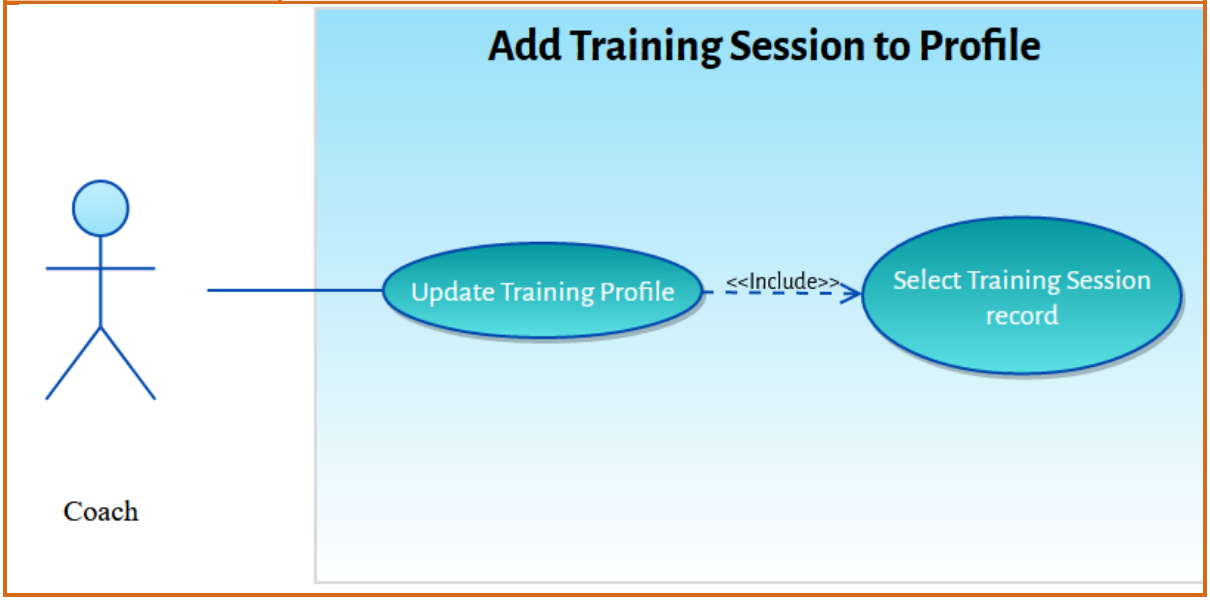
UC5b – Activity Diagram



UC5b – Sequence Diagram



Use Case Name:	UC5c_Coach_Update_Training_Profile_Training_Session



Initiating Actor(s):	Coach	Receiving Actor(s):	None

Trigger/Pre-condition(s):

The training session has been created in the system by the coach

Main Flow of Events:

1) Coach selects a Training Session to add to the profile.
2) Check if the player trained in that session.
3) The system will create a Training Log record for that player.
4) End of Use Case.

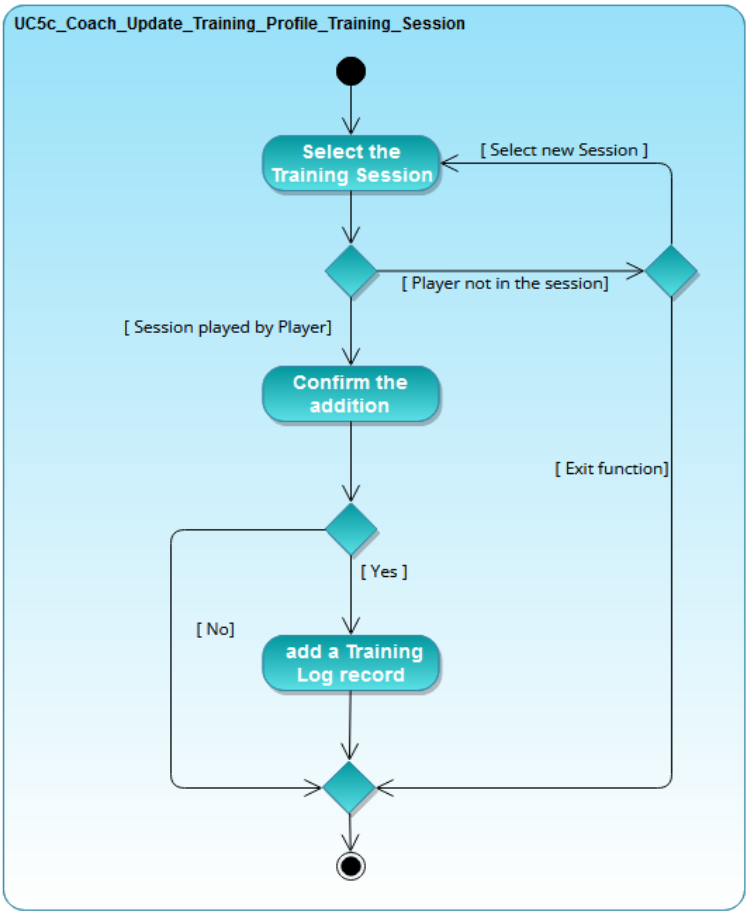
Alternate Flows:

3a. Player did not train in that session.
   1) Display error message.
   2) Coach selects another session.
   3) Continue to next step.
3b. Player did not train in that session.
   1) Display error message.
   2) Coach cancel function.
   3) Exit from Update Training Profile function.

Post-conditions:

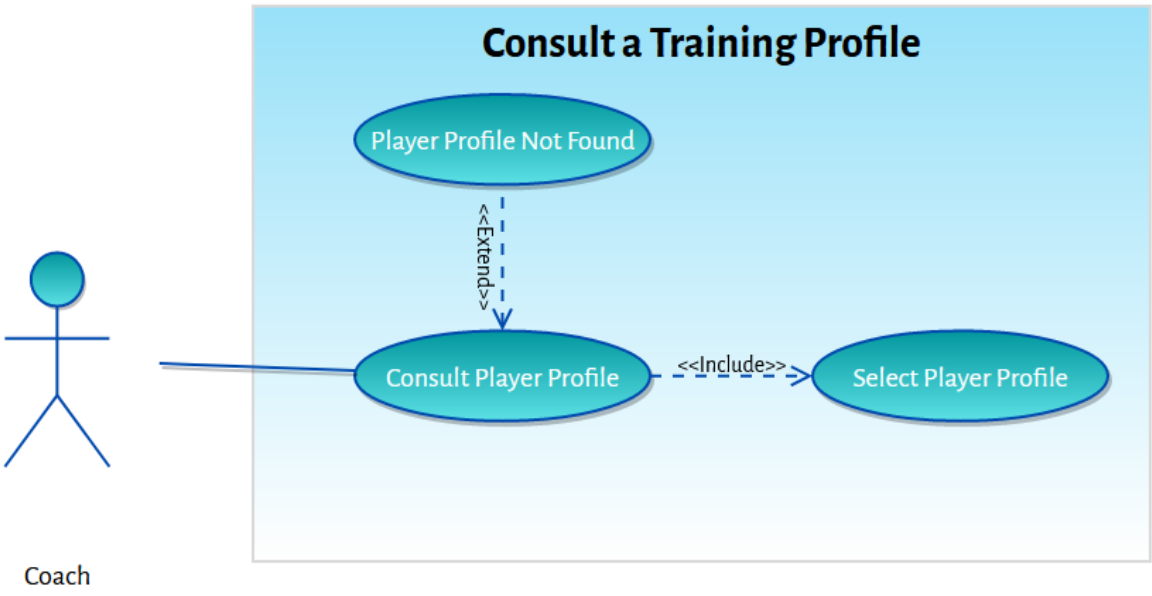
The training session has been added to the player profile.

UC5c – Activity Diagram



USE CASE 6: DISPLAY TRAINING PROFILE

Use Case Name:	UC6_Coach_Display_Training_Profile



Initiating Actor(s):	Coach	Receiving Actor(s):	None

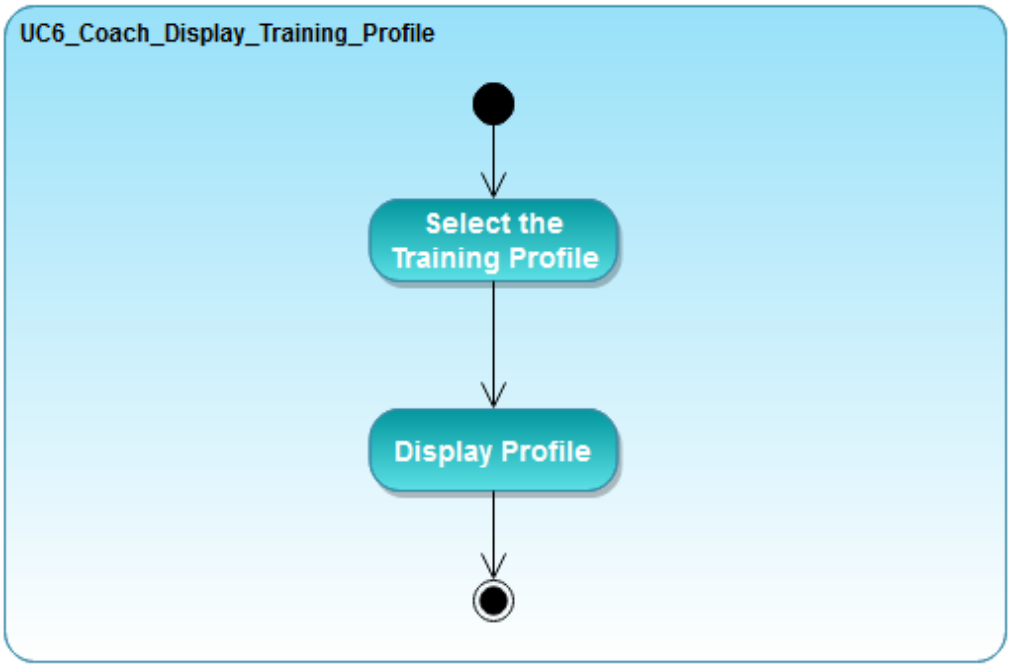
Trigger/Pre-condition(s):
Coach needs training info on a player.

Main Flow of Events:
1) Coach selects Player to show the profile of. 2) System displays the relevant Profile 3) End of Use Case

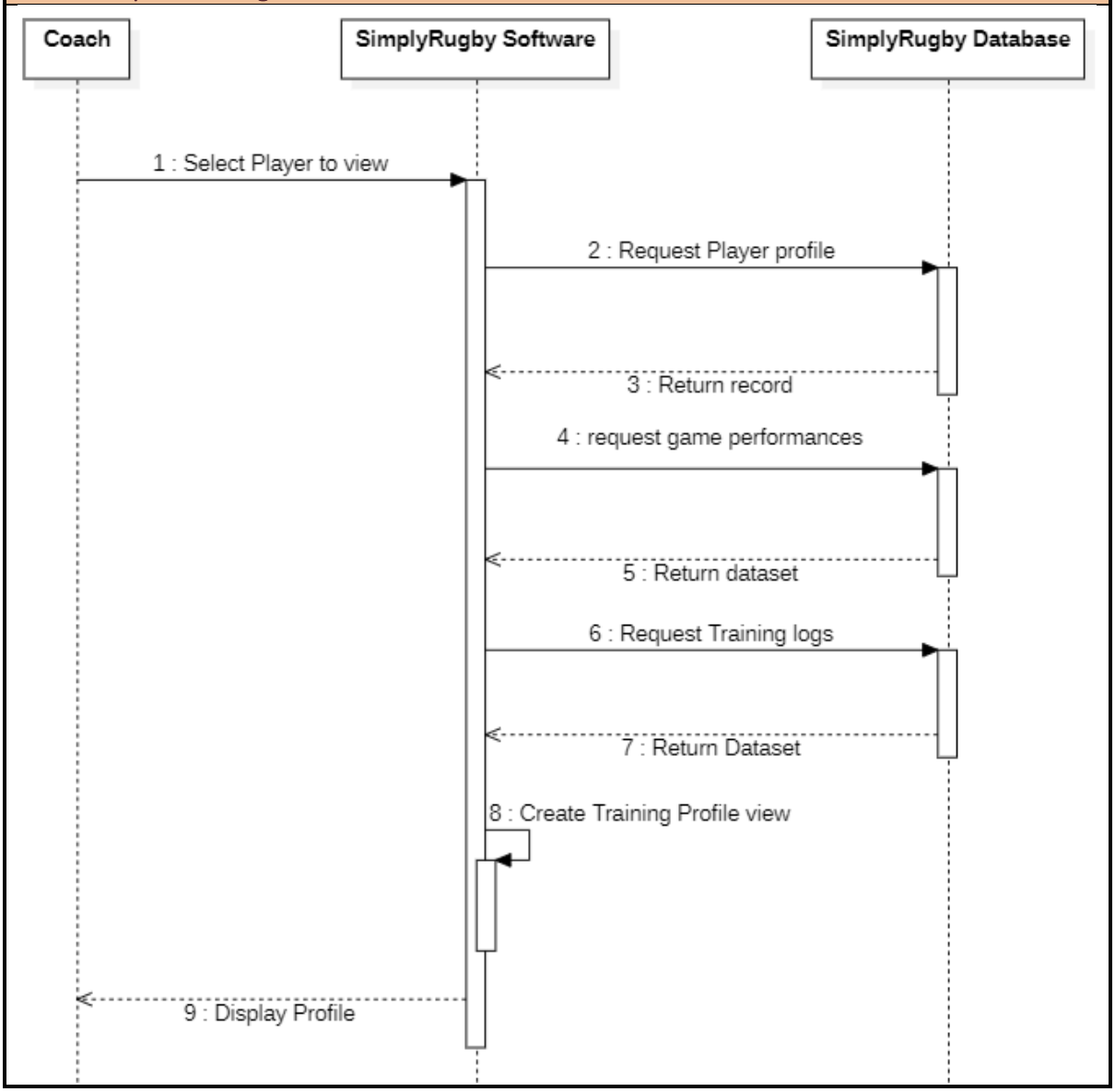
Alternate Flows:
1a. Player does not exist. 1) Exit from Display Training Profile function. 1b. Player does not exist. 1) Select another player from the list.

Post-conditions:
Relevant Profile is displayed.

UC6 – Activity Diagram

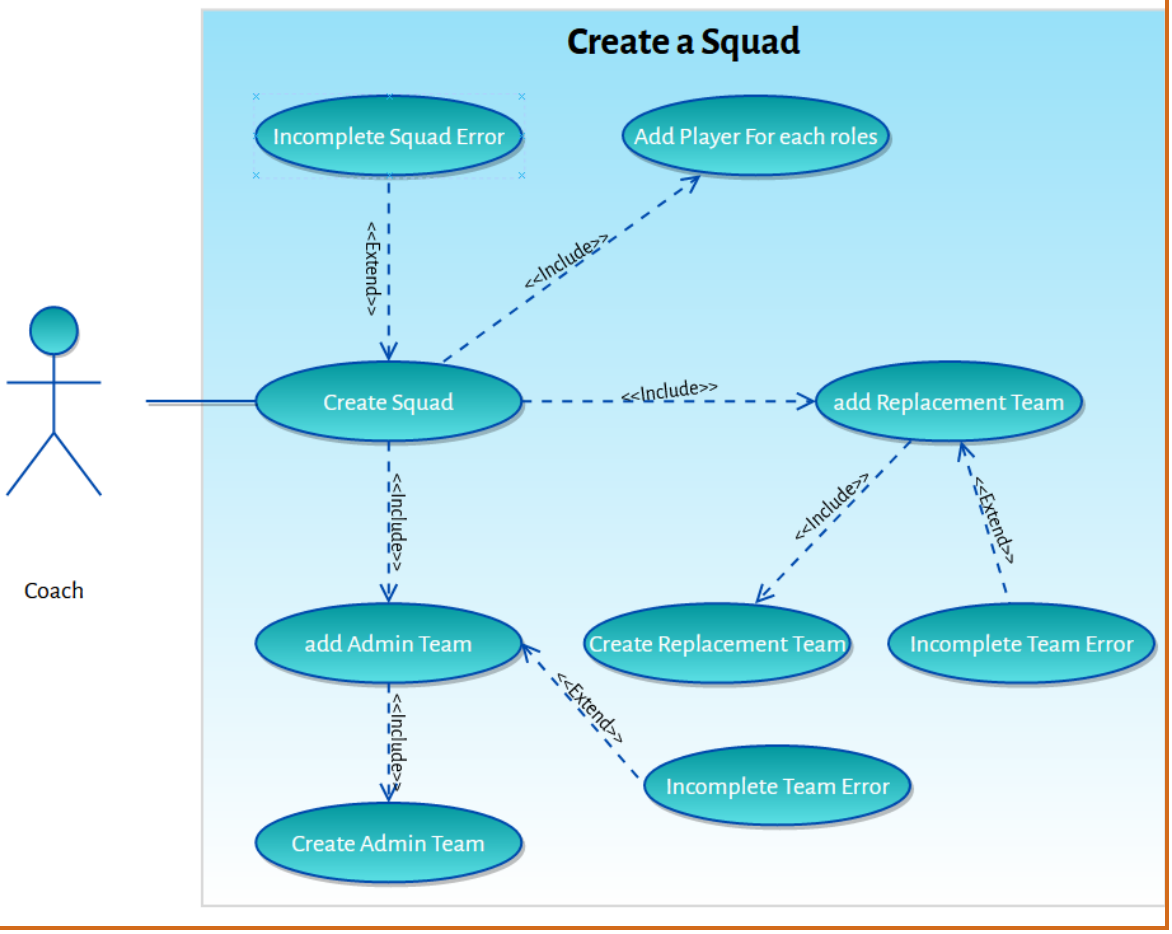


UC6 – Sequence Diagram



USE CASE 7: CREATE SQUAD

Use Case Name:	UC7_Coach_Create_Squad



Initiating Actor(s):	Coach	Receiving Actor(s):	None

Trigger/Pre-condition(s):
The club expends and need to create a new squad to accommodate or need to merge two incomplete squads into a new one. The Admin Team roster was passed to the coach by the club. Full headcounts available for team and sub-teams.

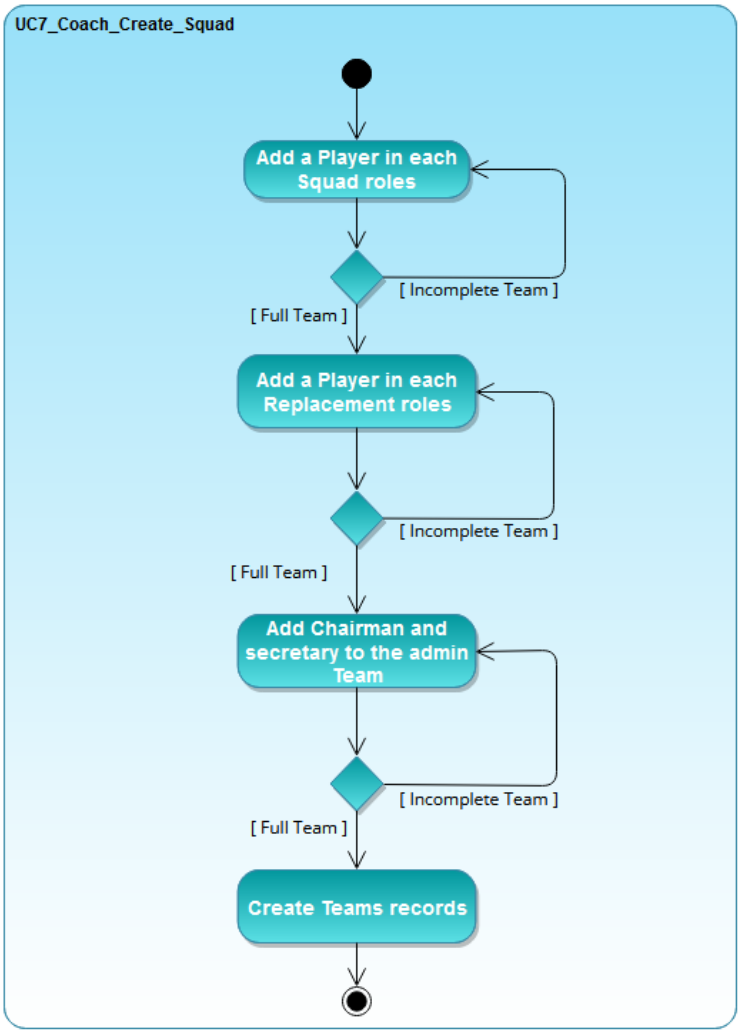
Main Flow of Events:
1) Coach assigns a player to each role in the squad. 2) Coach assigns a player to each slot in the replacement team. 3) Coach assigns a Chairman to the Admin team. 4) Coach assigns a Secretary to the Admin team. 5) Check if the admin team is full. 6) Check if the Replacement team is full. 7)Check if the Squad is full. 8) Create the relevant records in the system. 9) End of Use Case

Alternate Flows:
5a. Incomplete Admin Team 1) Select a non-playing member for the missing slot. 2) Continue to next step. 6a. Incomplete Replacement Team 1) Select a Player for the missing slot. 2) Continue to next step. 7a. Incomplete Replacement Team 1) Select a Player for the missing slot. 2) Continue to next step.

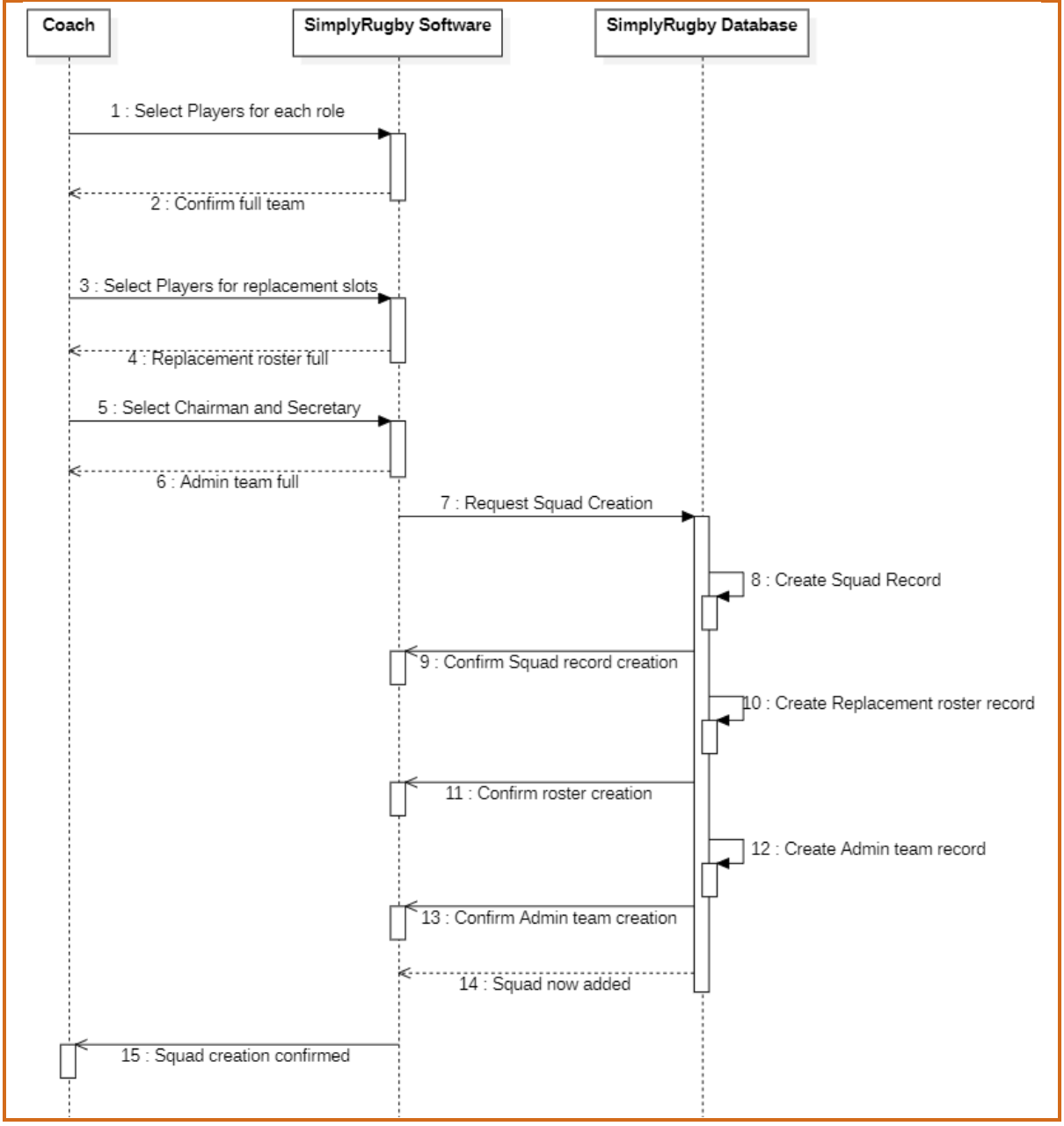
Assumptions(optional):
There are enough players and non-players created in the system to create a full squad.

Post-conditions:
The new Squad is created and saved in the system.

UC7 – Activity Diagram

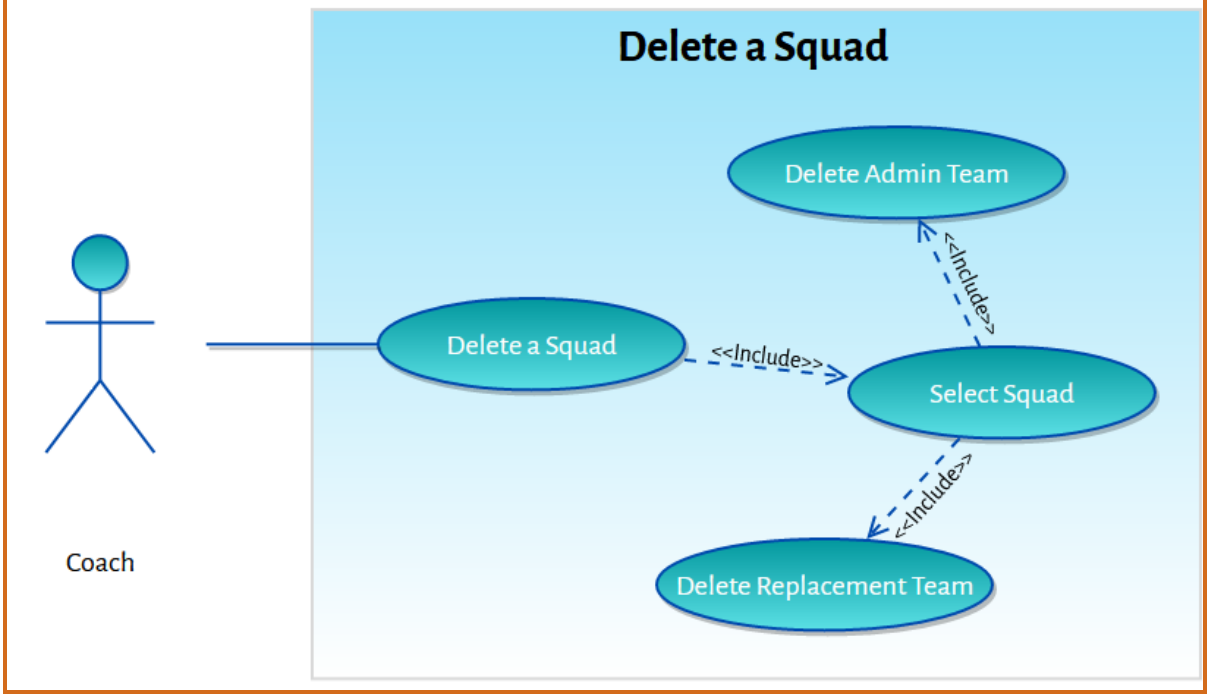


UC7 – Sequence Diagram



USE CASE 8: DELETE SQUAD

Use Case Name:	UC8_Coach_Delete_Squad



Initiating Actor(s):	Coach	Receiving Actor(s):	None

Trigger/Pre-condition(s):
The player base decreased and the club need to close a squad

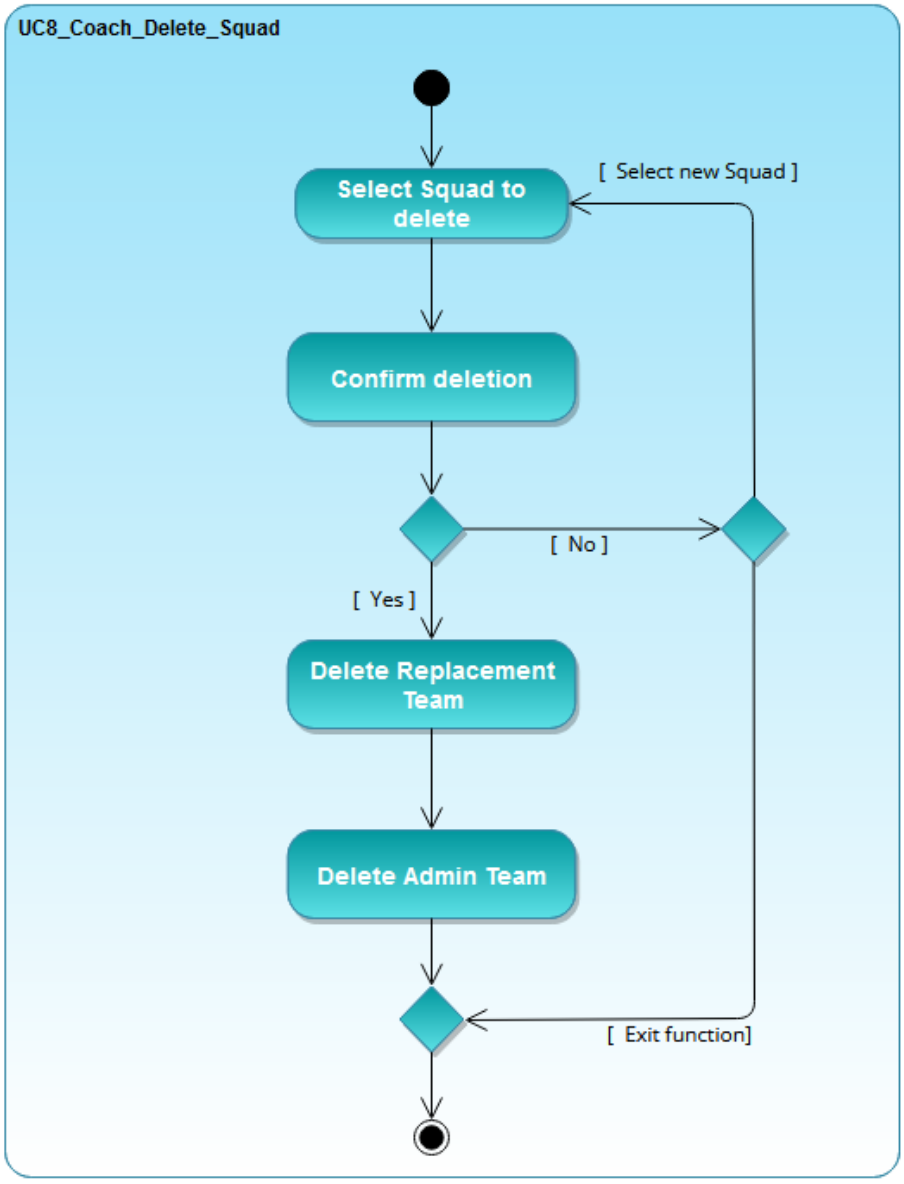
Main Flow of Events:
1) Coach selects a Squad to delete. 2) Coach confirms deletion. 3) Squad and assigned Replacement and Admin teams are deleted from the system. 4) End of Use Case

Alternate Flows:
2a. Coach cancels deletion 1) Coach selects another Squad to delete. 2) Continue to next step. 2a. Coach cancels deletion 1) Coach selects to exit function. 2) Exit Delete Squad function.

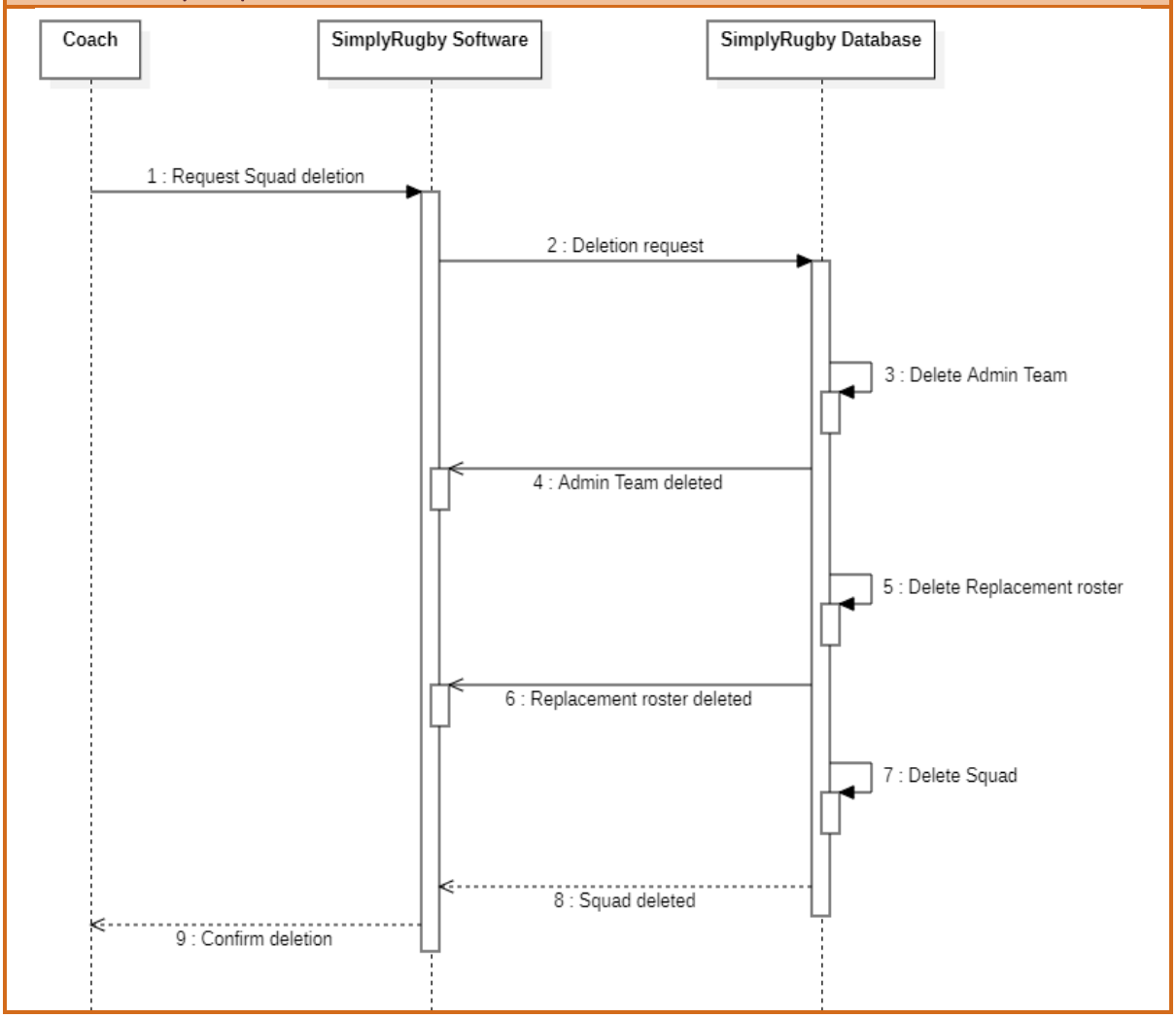
Assumptions(optional):
Non-playing members are not deleted so they can be reassigned. Same for players, as the leaving ones are deleted individually and the remaining ones could be re-assigned elsewhere.

Post-conditions:
The Squad is now deleted from the system.

UC8 – Activity Diagram

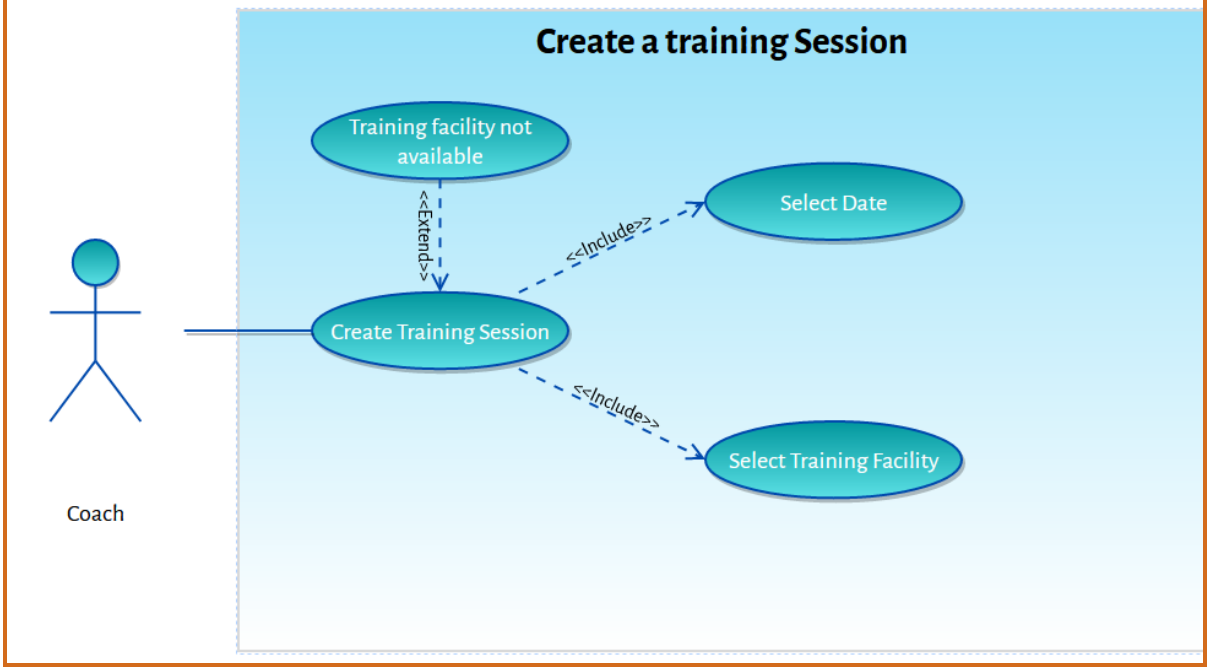


UC8 – Activity Sequence



USE CASE 9: CREATE TRAINING SESSION

Use Case Name:	UC9_Coach_Create_Training_Session



Initiating Actor(s):	Coach	Receiving Actor(s):	None

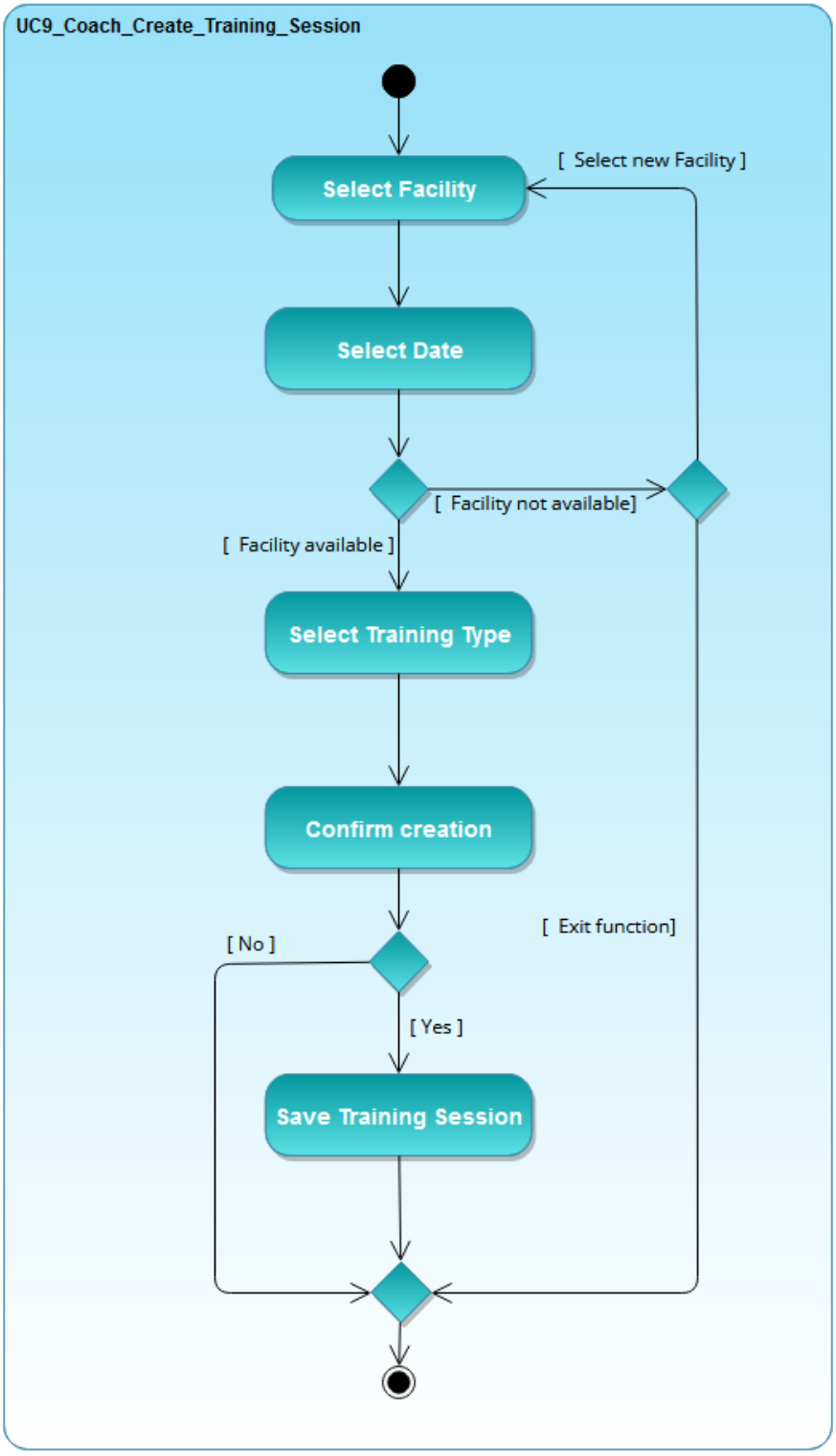
Trigger/Pre-condition(s):
Training sessions need to be recorded by Coaches, so they can be used in training Profiles

Main Flow of Events:
1) Coach selects a training facility. 2) Coach selects a date. 3) Check if the facility is available at that date. 4) Coach selects the training type. 5) Coach confirms the creation. 6) The training session is created. 7) End of Use Case.

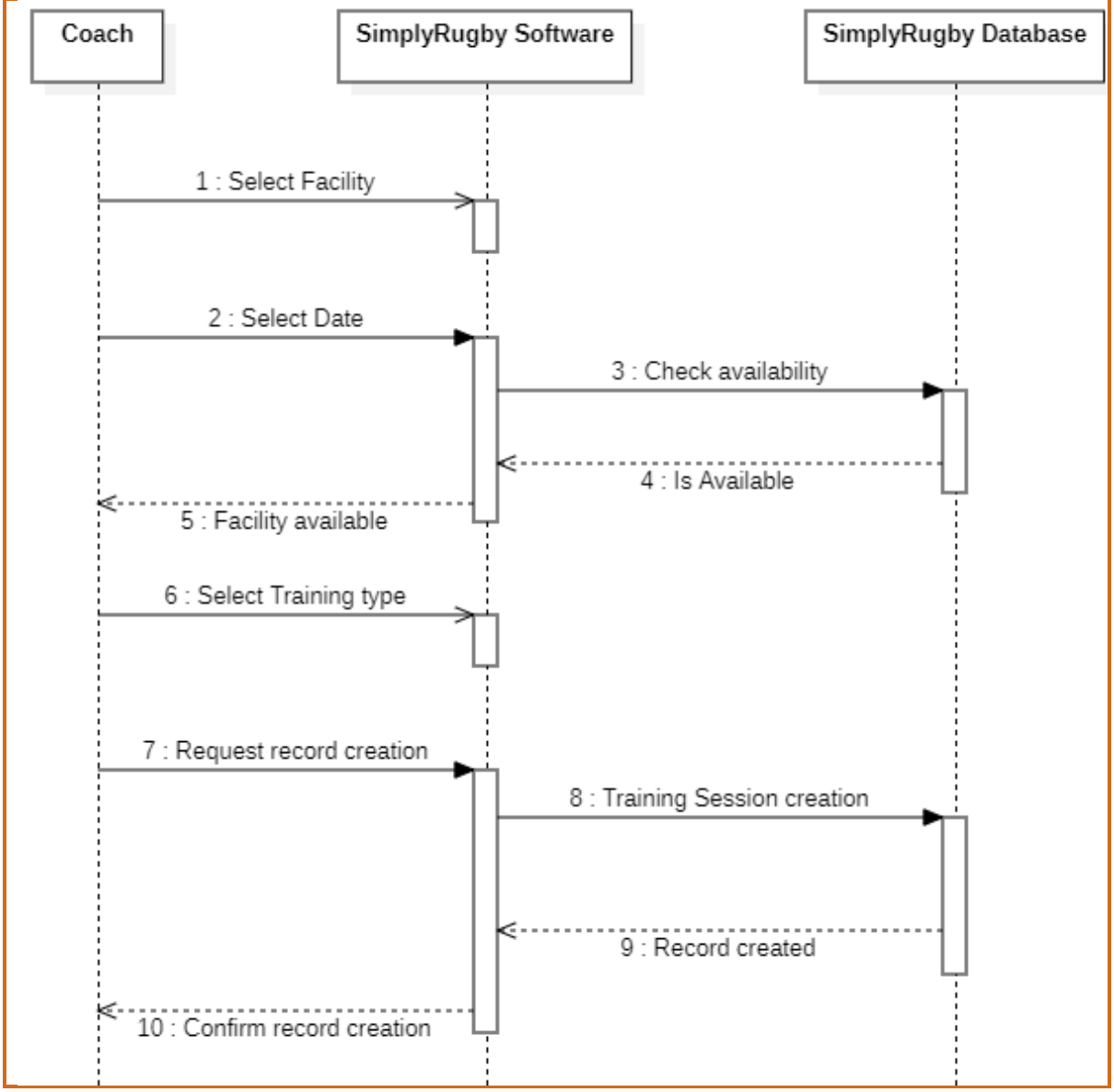
Alternate Flows:
3a. Facility not available at the selected date. 1) Coach selects another date. 2) Continue to next step. 3b. Facility not available at the selected date. 1) Coach selects another facility. 2) Continue to next step. 5a. Coach cancels record creation. 1) Exit Create Training Session function.

Post-conditions:
Session Training record created in the system.

UC9 – Activity Diagram

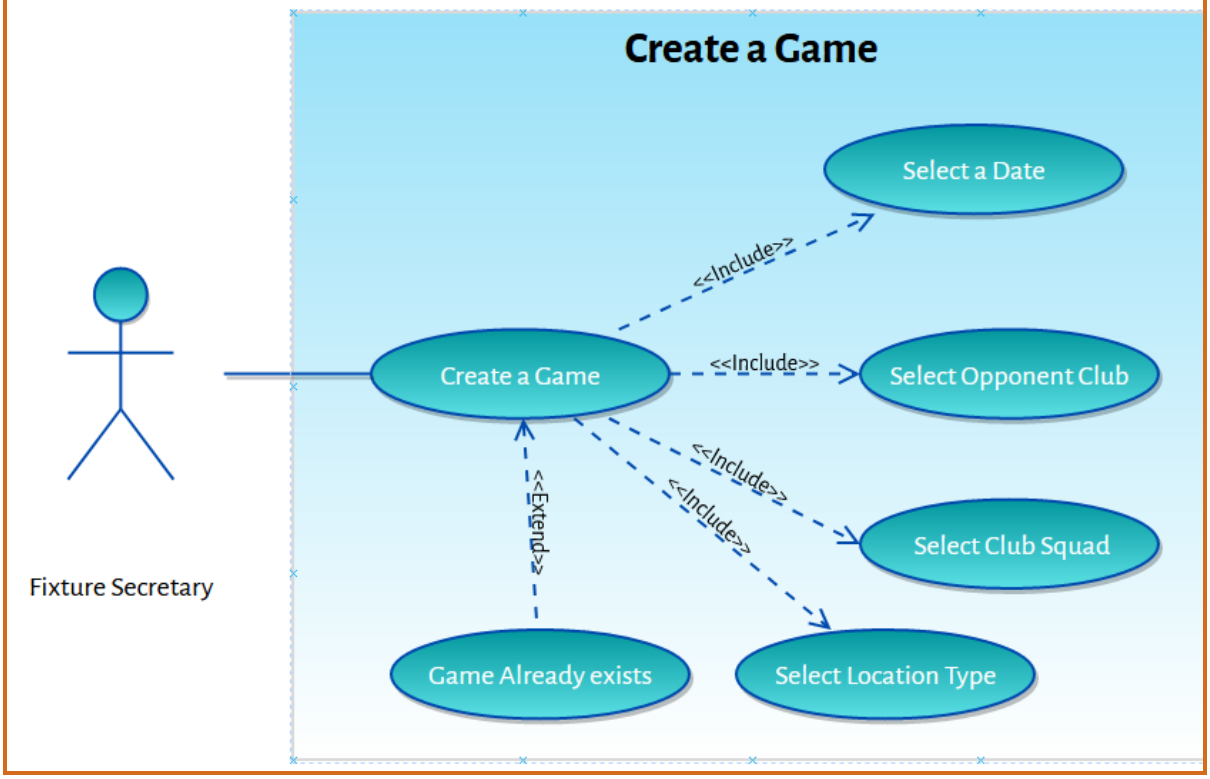


UC9 – Sequence Diagram



USE CASE 10: CREATE GAME

Use Case Name:	UC10_Secretary_Create_Game



Initiating Actor(s):	Secretary	Receiving Actor(s):	None

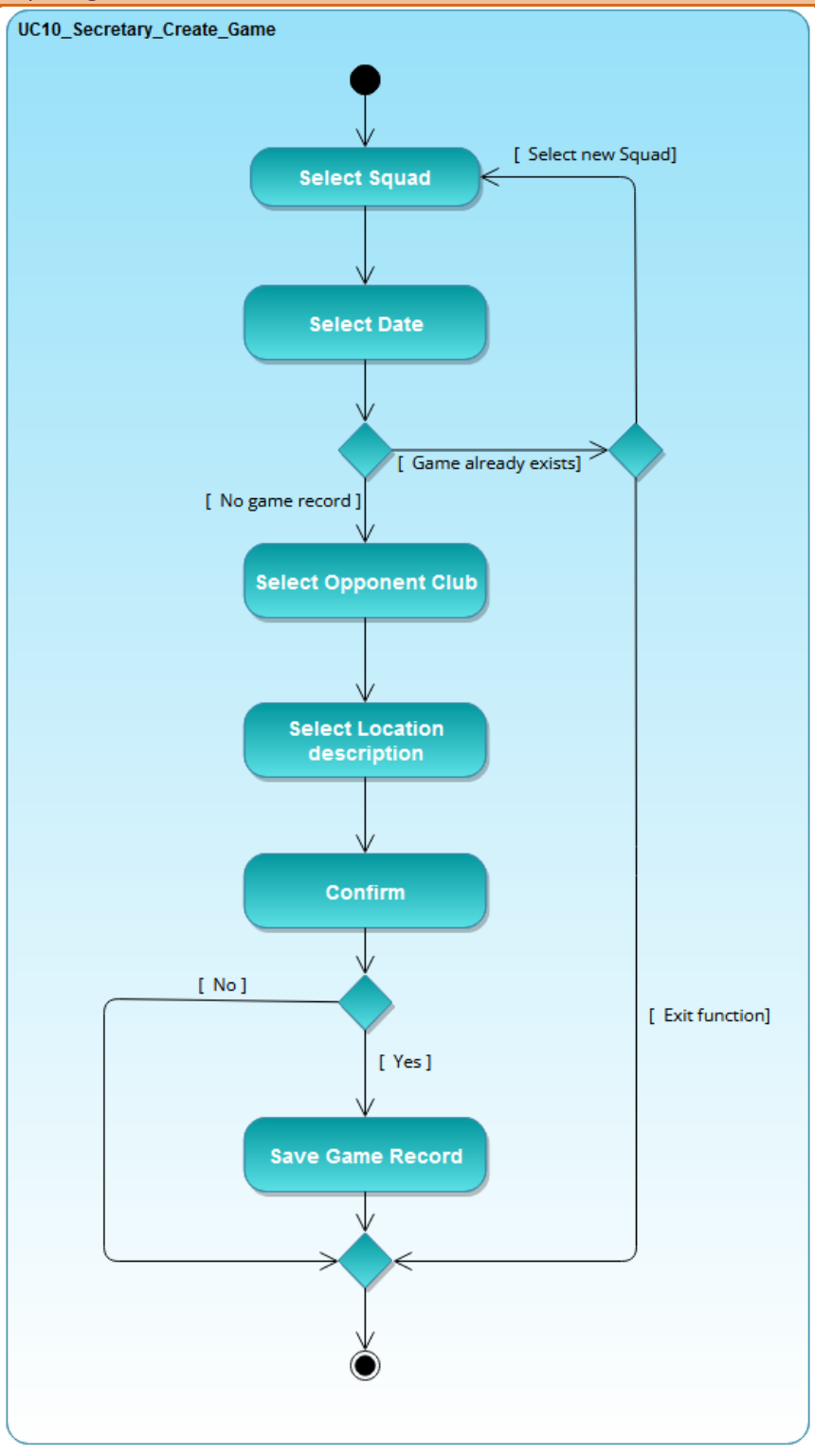
Trigger/Pre-condition(s):
Confirmed incoming Game at a date.

Main Flow of Events:
1) Secretary selects the club's Squad. 2) Secretary selects the date. 3) Check if the game already exists. 4) Secretary selects the opponent club. 5) Secretary selects location Type (Home or Away). 6) Game is saved. 7) End of Use Case.

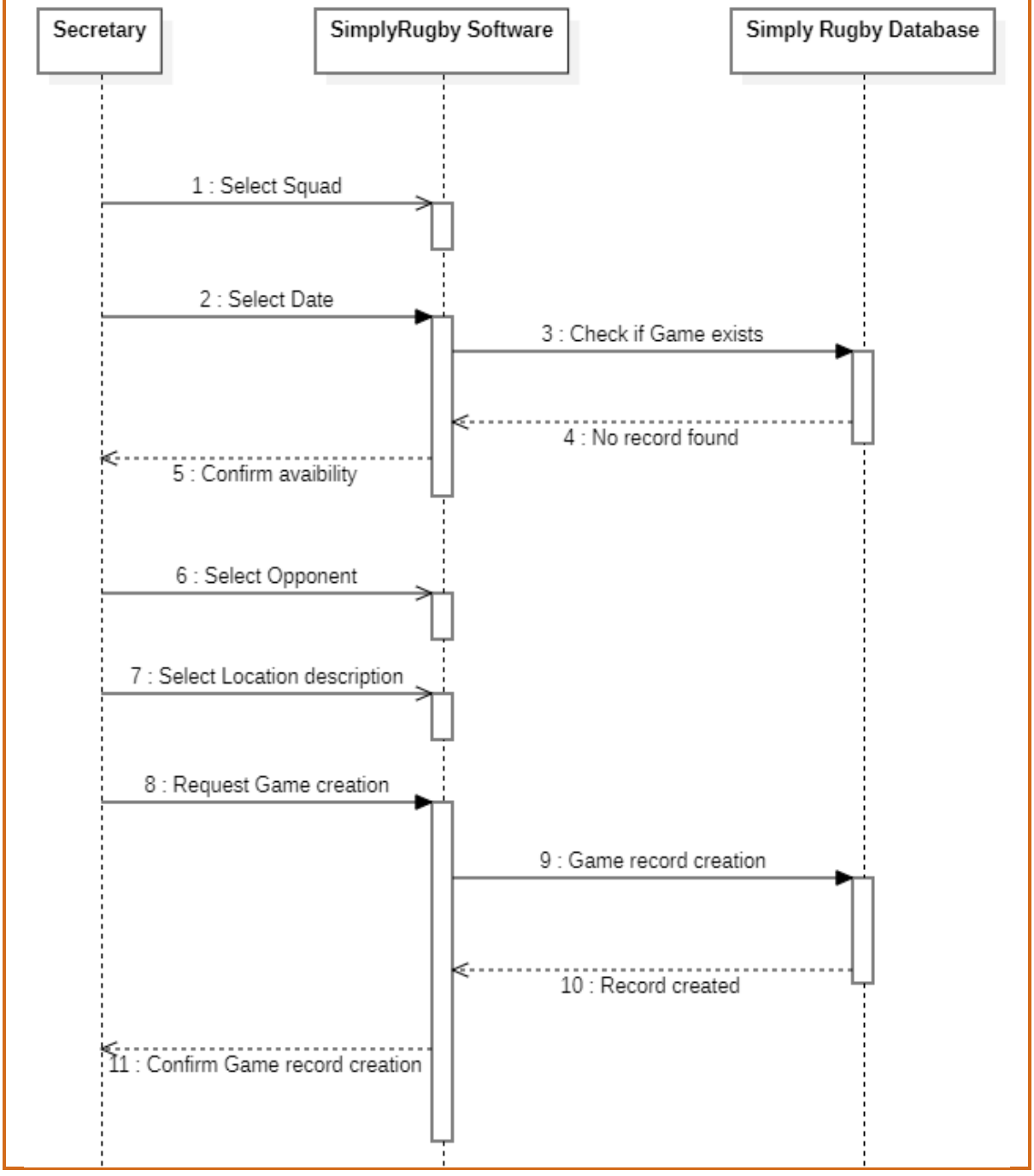
Alternate Flows:
3a. Squad already have a game planned at that date. 1) Secretary selects another date. 2) Continue to next step. 3a. Squad already have a game planned at that date. 1) Secretary exits function. 2) Exit Create Game Function.

Post-conditions:
Game record is created in the system.

UC10 – Activity Diagram

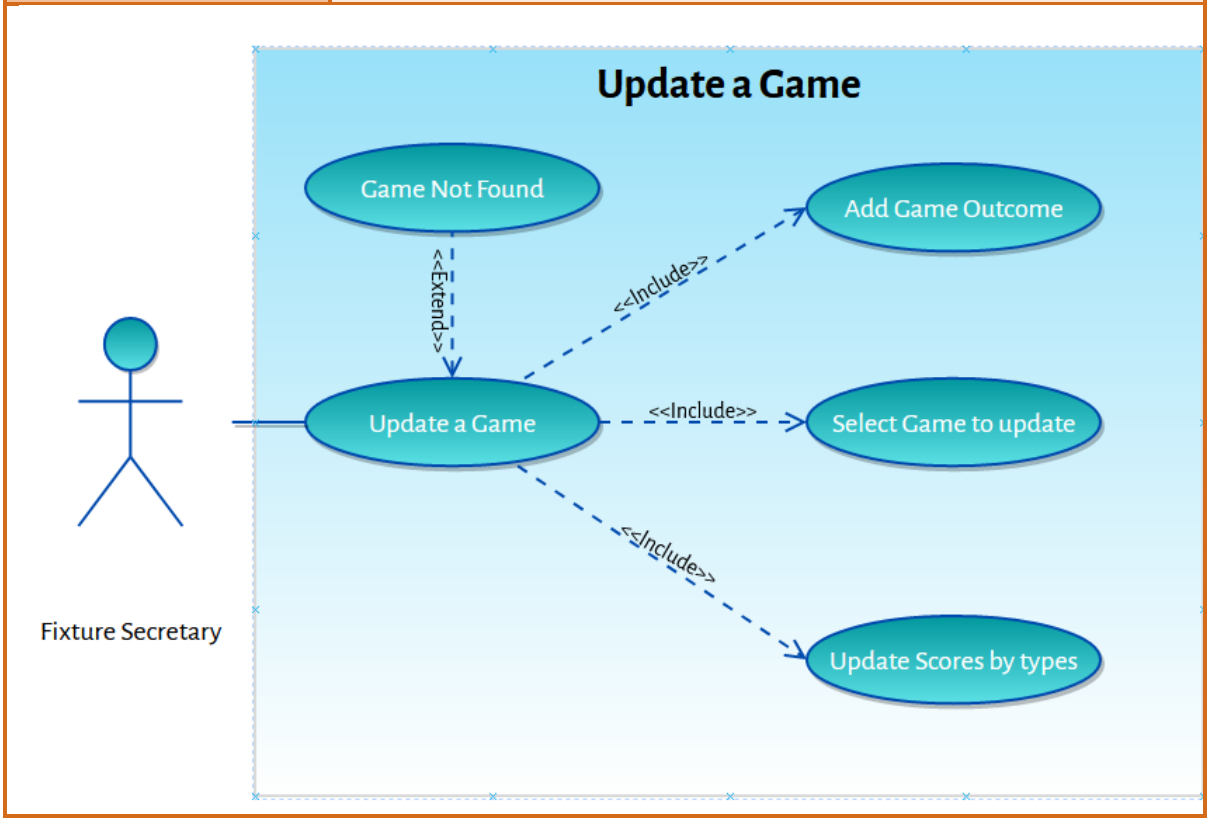


UC10 – Sequence Diagram



USE CASE 11: UPDATE A GAME

Use Case Name:	UC11_Secretary_Update_Game



Initiating Actor(s):	Secretary	Receiving Actor(s):	None

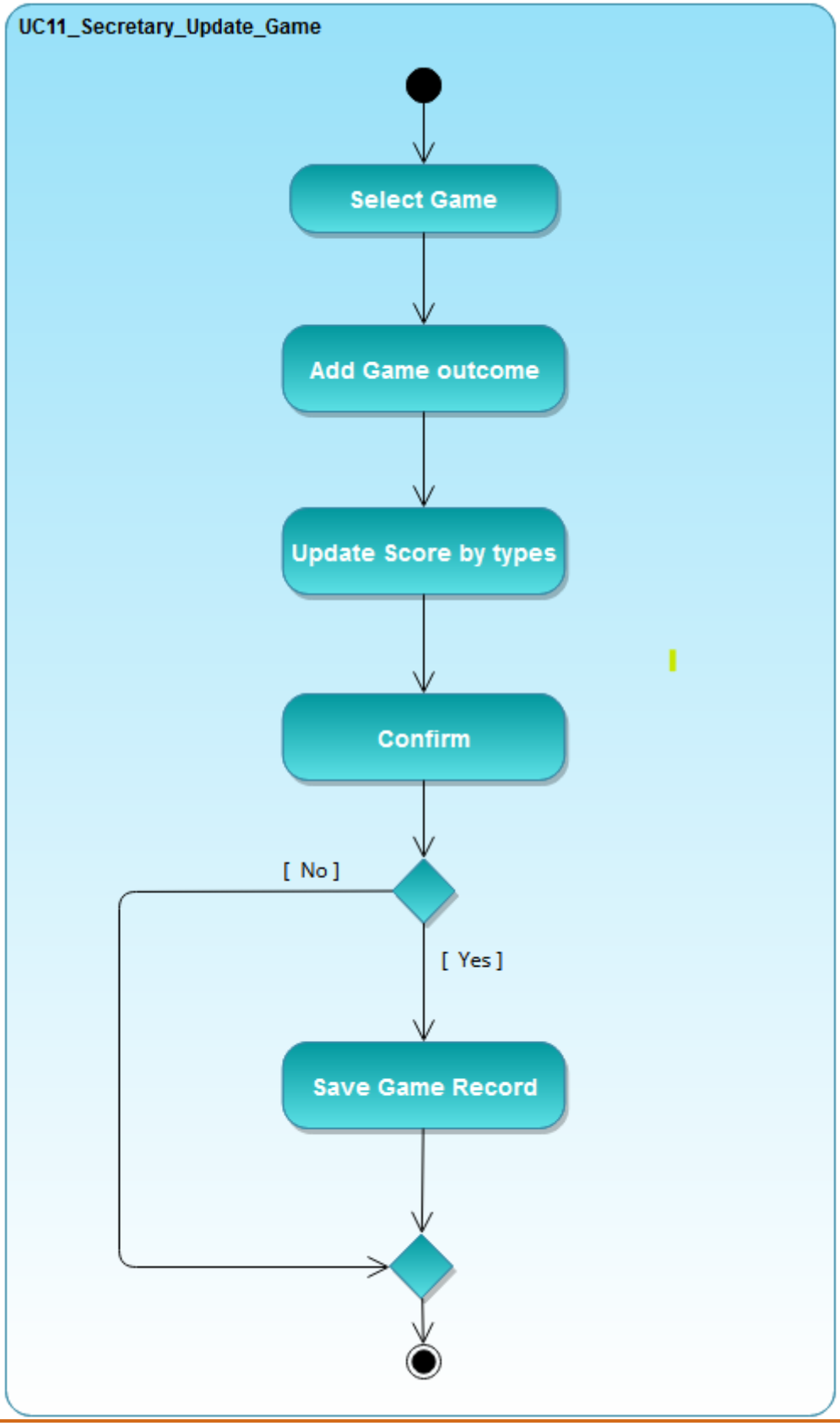
Trigger/Pre-condition(s):
The game has been played by the team.

Main Flow of Events:
1) Secretary searches for the game to update. 2) Secretary adds the game outcome from the list. 3) Secretary updates the scores by types. 4) Secretary confirms the update. 5) System updates the game record. 6) End of Use Case

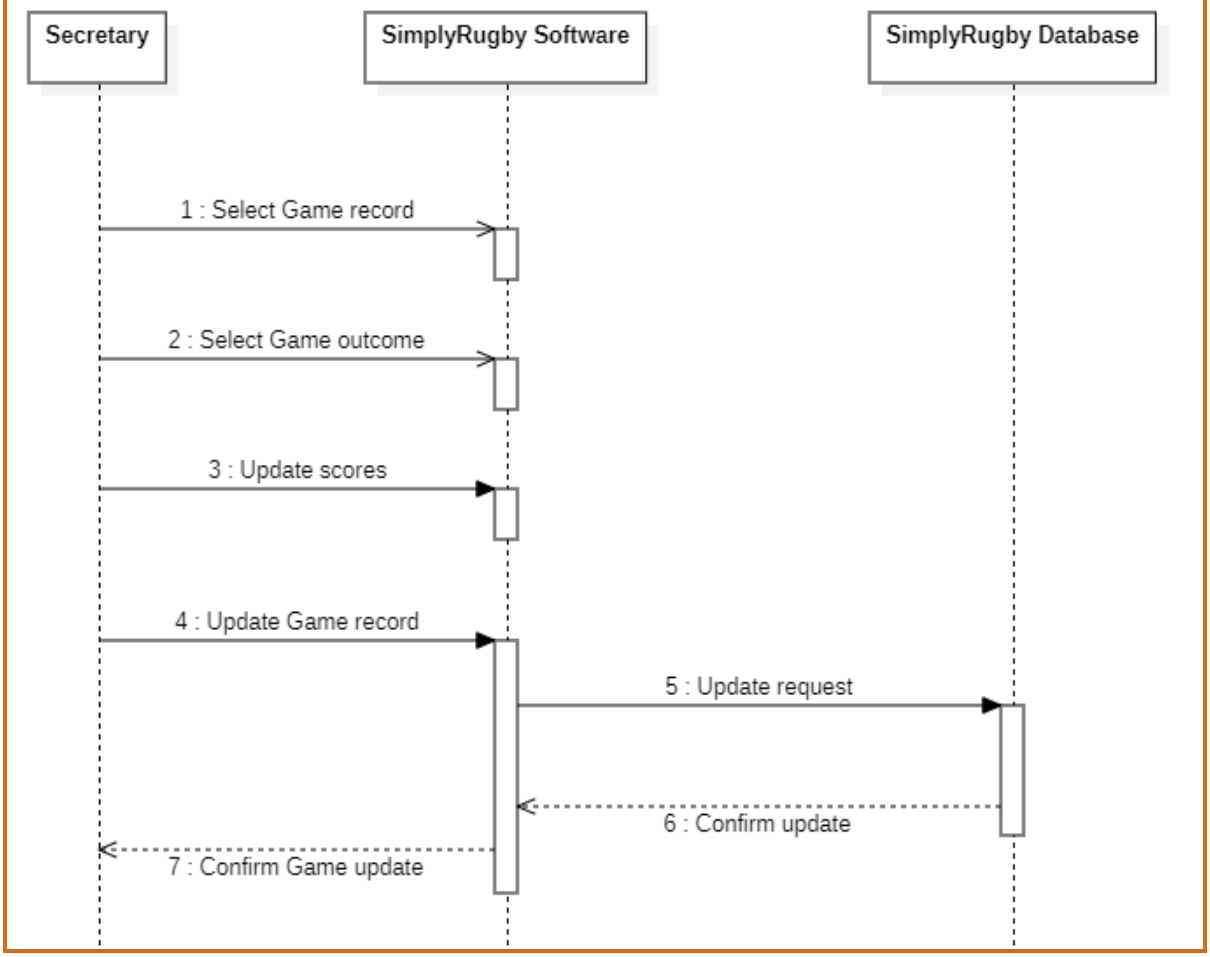
Alternate Flows:
5a. Secretary cancels the update. 1) Exit the Update Game function.

Post-conditions:
Game record is updated in the system.

UC11 – Activity Diagram



UC11 – Sequence Diagram

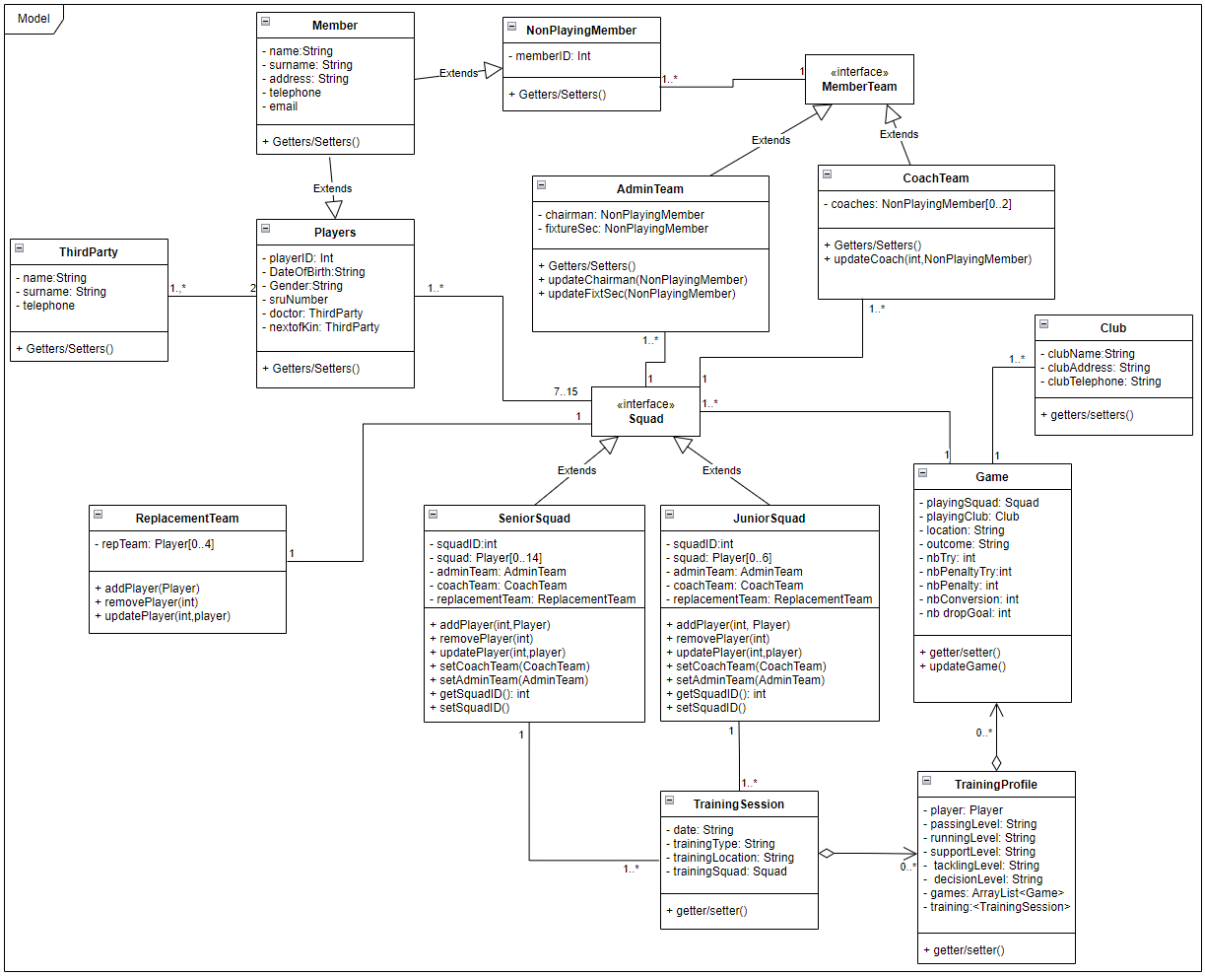


CLASS DIAGRAM

INTRODUCTION

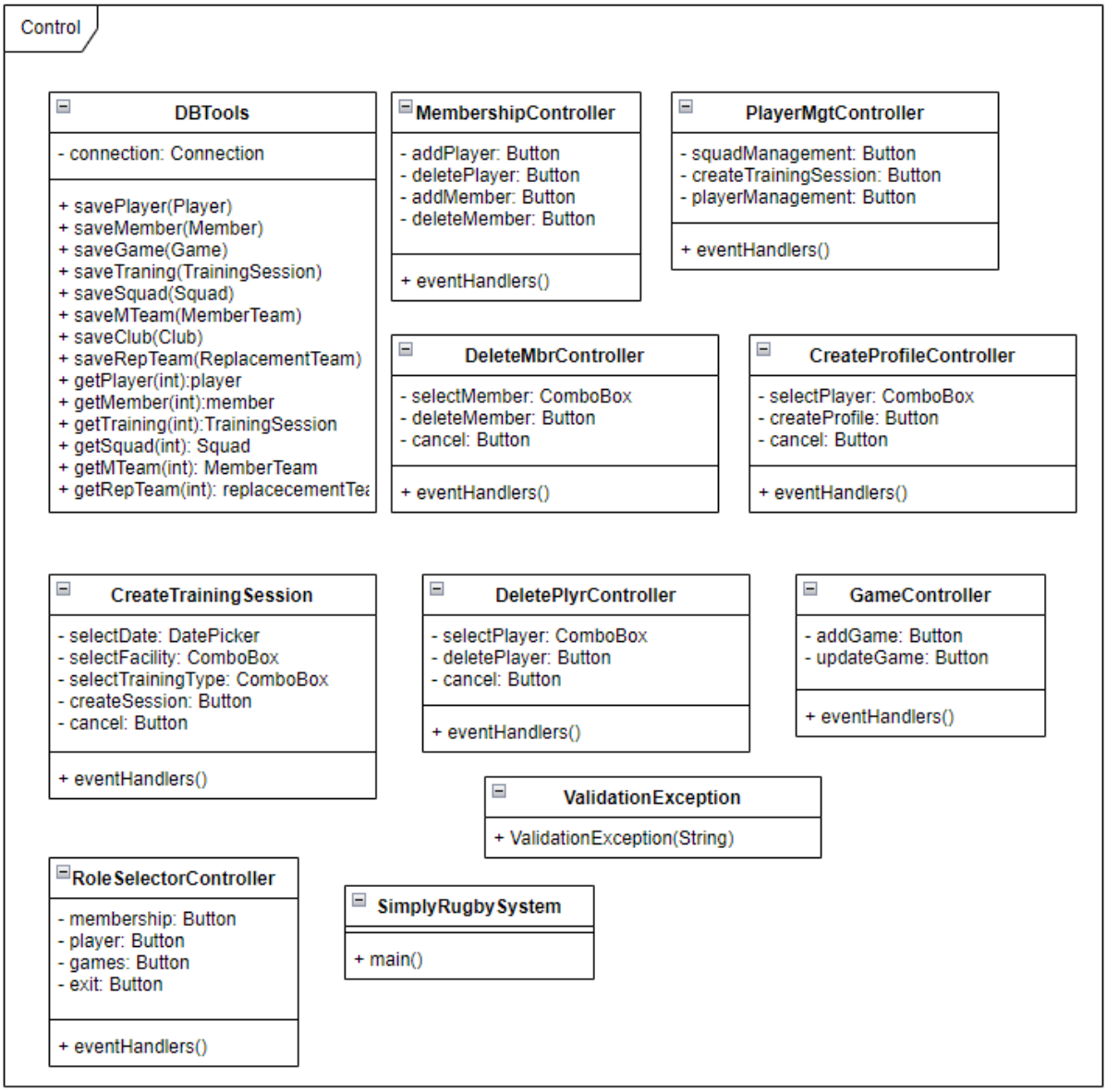
A class diagram is a tool used to model the objects handled by the application. It is usually done using the Unified Modelling Language, a widely used diagram notations.
For this project, some part of the class diagram will look like the previous ERD, this is because the software will use the data from the database and create objects from it, when needed those objects will be saved by being 'translated' in SQL queries. The application will make use of the MVC framework, so the class diagram has been divided into two parts: Model and Control. The Model show the classes like players, games, profiles and so on, used by the software to handle the application data. The Control diagram shows the 'system' classes: the classes used by the software to control itself.

MODEL DIAGRAM



(See Appendix K and attached Draw.io file for a better view)

CONTROL DIAGRAM



The control diagram shows the controllers of the main functions and popup windows (JavaFX scenes), internal panes will be handled by the event handlers. Some of it may change during the Implementation phase but this will be documented and shared in the Evaluation phase.

DATA DICTIONARY

In this section, the classes presented in the Class Diagram are detailed, presenting their roles and the projected variables and methods.

Most of the classes in the Model section are basically data structures with few methods beyond the usual getters and setters. The reason is that those objects will be saved or created in the database by one of the Control classes. This specific class will translate objects in SQL queries and vice-versa.

Unless stated otherwise, all variables/constants are local.

MODEL CLASSES

ThirdParty	This class will hold the data for Next of Kin and Doctors. Inheritance was not implemented as the data for both are the same and there is no value to setup child classes.
Attributes: ▪ **name:** String to hold the first name. ▪ **surname:** String to hold the surname. ▪ **Telephone:** String to store the telephone.	**Methods:** Only getters and setters for the variable will be implemented for this class. The save and load will be done in the Player class.

Member	This is a parent class to cover the two types of club members: players and non-playing members (Admins, secretaries, coaches, etc.). the class will have the shared data by its children.
Attributes: ▪ **name:** String to hold the first name. ▪ **surname:** String to hold the surname. ▪ **telephone:** String to store the telephone. ▪ **Email:** String to store an email	**Methods** Only getters and setters for this class. Specific methods will be set in the children's classes.

Player	Child of the Member class, this class is customised to collect additional data specific to the players on top of the one defined in the Member class. It also adds methods to get and insert data in the database.
Attributes ▪ **playerID:** Int to store the primary key of the corresponding entry in the DB. This key is generated by the DB, not the class. ▪ **dateofBirth:** String to store the DoB.	**Methods** Will have the usual getters and setters.

▪ **Gender:** String to hold the player's gender. ▪ **sruNumber:** Int to store the SCRUMS number, 7 digits number. ▪ **Doctor:** Will store a ThirdParty object as the player's doctor. ▪ **nextToKin:** Will store a ThirdParty object as the player's next of kin.	

NonPlayingMember	Child of the Member class, it adds methods to get and insert data in the database.
Attributes ▪ **memberID:** Int to store the primary key of the corresponding entry in the DB. This key is generated by the DB, not the class.	**Methods** Getters and Setters.

MemberTeam	Interface that will define the admin and coaches' teams. This also can be used to hold a collection of teams objects if needed.
Attributes **N/A**	**Methods** N/A

AdminTeam	Implements the MemberTeam interface. This class defines the admin team assigned to a squad.
Attributes ▪ **chairman:** stores a NonPlayingMember object for the Squad chairman. ▪ **fixtureSec:** stores a NonPlayingMember object for the Squad Fixture secretary.	**Methods** Getters and Setters. ▪ **updateChairman(NonPlayingMember):** will replace the existing chairman by the one passed as parameters. ▪ **updateFixtSec(NonPlayingMember):** replace the existing secretary by the one in parameters.

CoachTeam	Implements the MemberTeam interface. This class defines the coaches' team assigned to a squad.
Attributes ▪ **coaches:** Array of NonPlayingMember, to hold 3 coaches objects.	**Methods** Getters and Setters. ▪ **updateCoach(Int, NonPlayingMember):** will replace the existing coach by the one passed as parameters. The Int is the index of the coach to replace.

Club	Class to define and store the data of opponent clubs.
Attributes ▪ **clubName:** String to store a club's name. ▪ **clubAddress:** String to store the club's address. ▪ **clubTelephone:** String to store a club's telephone.	**Methods** Getters and Setters

SeniorSquad	This class describes a senior squad playing in rugby 15
Attributes ▪ **squadID:** Int to hold the squad's primary key from the database. ▪ **Squad:** array of Players to store the 15 players of the squad. The index corresponding to the role number. ▪ **adminTeam:** Variable to hold the assigned admin team. ▪ **coachTeam:** Variable to hold the assigned coach team. ▪ **replacementTeam:** store the replacement team assigned to the squad.	**Methods** ▪ **addPlayer(Int, Player):** add at the index Int(role) the Player passed in parameter. ▪ **removePlayer(int):** remove the player at the index passed in parameter from the array. ▪ **updatePlayer(int, Player):** replace the player at the index passed to the method by the player passes as parameter. ▪ **setCoachTeam(CoachTeam):** set the Squad's coach team by the team passed in parameter. ▪ **setAdminTeam(AdminTeam):** set the Squad's admin team by the team passed in parameter. ▪ **getSquadID():** returns the ID (primary key) of the squad. ▪ **setSquadID():** set the squad ID.

JuniorSquad	This class describes a junior squad playing in rugby 7
Attributes ▪ **squadID:** Int to hold the squad's primary key from the database. ▪ **Squad:** array of Players to store the 7 players of the squad. The index corresponding to the role number. ▪ **adminTeam:** Variable to hold the assigned admin team. ▪ **coachTeam:** Variable to hold the assigned coach team.	**Methods** ▪ **addPlayer(Int, Player):** add at the index Int(role) the Player passed in parameter. ▪ **removePlayer(int):** remove the player at the index passed in parameter from the array. ▪ **updatePlayer(int, Player):** replace the player at the index passed to

▪ **replacementTeam:** store the replacement team assigned to the squad.	the method by the player passes as parameter. ▪ **setCoachTeam(CoachTeam):** set the Squad's coach team by the team passed in parameter. ▪ **setAdminTeam(AdminTeam):** set the Squad's admin team by the team passed in parameter. ▪ **getSquadID():** returns the ID(primary key) of the squad. ▪ **setSquadID():** set the squad ID.

ReplacementTeam	This class will hold the 5 players assigned as replacement.
Attributes ▪ **repTeam:** Array to hold the replacement players.	**Methods** ▪ **addPlayer(Int, Player):** add at the index Int(role) the Player passed in parameter. ▪ **removePlayer(int):** remove the player at the index passed in parameter from the array. ▪ **updatePlayer(int, Player):** replace the player at the index passed to the method by the player passes as parameter.

Game	Class to create and store a Game object.
Attributes ▪ **playingSquad:** Store the club's playing squad. ▪ **playingClub:** Store the opposing club. ▪ **location:** String to hold the location type (home or Away). ▪ **nbTry:** int to store the number of Try scored. ▪ **nbPenaltyTry:** int to store the number of Penalty Try scored. ▪ **nbPenalty:** int to store the number of Penalty scored. ▪ **nbConversion:** int to store the number of Conversions scored. ▪ **nbDropGoal:** int to store the number of goals scored	**Methods** Getter and Setters. ▪ **updateGame():** method to update the Game record in the database with the after-match scores.

TrainingSession	Class to hold the data of a training session
Attributes ▪ **date:** String to store the date of the session. ▪ **trainingType:** String holding the session description. ▪ **trainingSquad:** Hold the squad present for the session.	**Methods** Getters and Setters.

TrainingProfile	This class describes the training profile of a players
Attributes ▪ **player:** Store the player's object. ▪ **passingLevel:** String describing the skill level, as defined in the Performance_level table in the database. ▪ **runningLevel:** Same as previous skill. ▪ **tacklingLevel:** Same as previous skill. ▪ **decisionLevel:** Same as previous skill. ▪ **games:** ArrayList that holds all the games played by the player's squad. ▪ **training:** ArrayList that will store the trainings played by the player.	**Methods** Getters and Setters.

CONTROL CLASSES

Those are the classes manipulating the structures defined in the Model diagrams. Those classes are in charges of the graphical interface and relation with the database.

<table>
<tr><td>DBTools</td><td colspan="2">This class will be a singleton class, static methods and forbid creation of objects. It will propose methods to connect to the database and interact with it for create, update, update and delete.</td></tr>
<tr><td colspan="2">Attributes
connection: Connection object that will connect to the system database.</td><td>Methods
savePlayer(Player): Translate the Player object in a SQL INSERT query to add a new record in the database. This will also insert the assigned Doctor and next of kin in the DB.
saveMember(NonPlayingMember): same as the previous method.
saveGame(Game): Saves the game in the database. If the record already exists, it will update it instead with the game scores.
saveTraining(TrainingSession): Saves a training session in the database.
saveSquad(Squad): This will save a squad in the database. It will also save the replacement, admin and coach team in the DB.
saveMTeam(MemberTeam): will insert an admin or coach team in the database. It will test what kind of object is passed to use the correct query.
saveClub(Club): method will insert a Club record in the DB.
saveRepTeam(ReplacementTeam): Will insert a replacement team in the database.
getPlayer(int): Search a player by its primary key, create and return a Player object from it.</td></tr>
</table>

	▪ **getMember(int):** Search a member by its primary key, create and return a Member object from it. ▪ **getTraining(int):** Search a training session by its primary key, create and return an object from it. ▪ **getSquad(int):** Search a Squad by its primary key, create and return a Squad object from it. Will also get the assigned replacement, coaches and admin teams. ▪ **getMTeam(int):** Search a member team by its primary key, create and return an Admin or Coaches team object from it. ▪ **getRepTeam(int):** Search a replacement team by its primary key, create and return a team object from it.

RoleSelectorController	This will handle the view to select between the 3 areas covered by the software: membership(admin), players(coaches) and the games(Fixture secretaries).
Attributes ▪ **membership:** Button to access the admin view. ▪ **Player:** Button to access the Player view. ▪ **Games:** Button to access the Game view.	**Methods** **eventHandlers() :** Those lambda functions will call the panels and functions inside the 3 main area of functionalities.

MembershipController	This controller handles the membership functions for the admins.
Attributes ▪ **addPlayer:** Button to display the add Player view. ▪ **deletePlayer:** Button to call the delete player windows. ▪ **addMember:** Button to display the add member view. ▪ **deleteMember:** Button the call the delete members windows.	**Methods** **eventHandlers() :** Those lambda functions will call the panels and functions managing the membership base.

PlayerMgtController	Controller class to manage the squad and players views/windows.
Attributes ▪ **squadManagement:** Button to display the Squad panel. ▪ **createTrainingSession:** Button to call the create session windows. ▪ **playerManagement:** Button to display the player management panels	**Methods** **eventHandlers() :** Lambda functions that will call the panels and functions managing the Player base.

ValidationException	Inherits from the Java Exception class, this will be used as a generic exception when the program does data validation, like regex, and so on.
Attributes **N/A**	**Methods** **validationException(String):** uses the super keyword to pass the String to the superclass to use its getMessage() method. Used to call an alert window for the error

DeleteMbrController	This is the controller for the window called to delete a member
Attributes ▪ **selectMember:** ComboBox used to select the member to delete, the list will be fetched from the database. ▪ **deleteMember:** Button used to delete the selected member. ▪ **cancel:** Button to cancel and close the window.	**Methods** **eventHandlers() :** Lambda functions that will manage the buttons and ComboBox.

CreateProfileController	Controller to manage the window called to create a player profile.
Attributes ▪ **selectPlayer:** ComboBox used to select the player for the profile, the list will be fetched from the database. ▪ **createProfile:** Button used to create the profile. ▪ **cancel:** Button to cancel and close the window.	**Methods** **eventHandlers() :** Lambda functions that will manage the buttons, ComboBox and proceed with the profile creation.

DeletePlyrController	Controller to manage the window called to delete a player record.
Attributes ▪ **selectPlayer:** ComboBox used to select the member to delete, the list will be fetched from the database. ▪ **deletePlayer:** Button used to delete the selected member. ▪ **cancel:** Button to cancel and close the window.	**Methods** **eventHandlers() :** Lambda functions that will manage the buttons, ComboBox and proceed with the deletion.

CreateTrainingSession	This controller is used to manage the window used to create a training session.
Attributes ▪ **selectDate:** JavaFX DatePicker to select a date. ▪ **selectFacility:** ComboBox to select a club's facility. ▪ **selectTrainingType:** ComboBox to select the type of training session. ▪ **createSession:** Button to create the session. ▪ **cancel:** Button to cancel and close.	**Methods** **eventHandlers() :** Lambda functions that will manage the buttons, ComboBox, DatePicker and proceed with the creation.

GameController	Class to control the Add/Update game panels.
Attributes ▪ **addGame:** Button to call the add game internal pane. ▪ **updateGame:** Button to call the internal update game pane.	**Methods** **eventHandlers() :** Lambda functions that will manage the buttons, Add and Updates panes.

SimplyRugbySystem	Entry call of the application.
Attributes **N/A**	**Methods** ▪ **main():** Entry point of the program, will launch the first window.

USER INTERFACE DESIGN

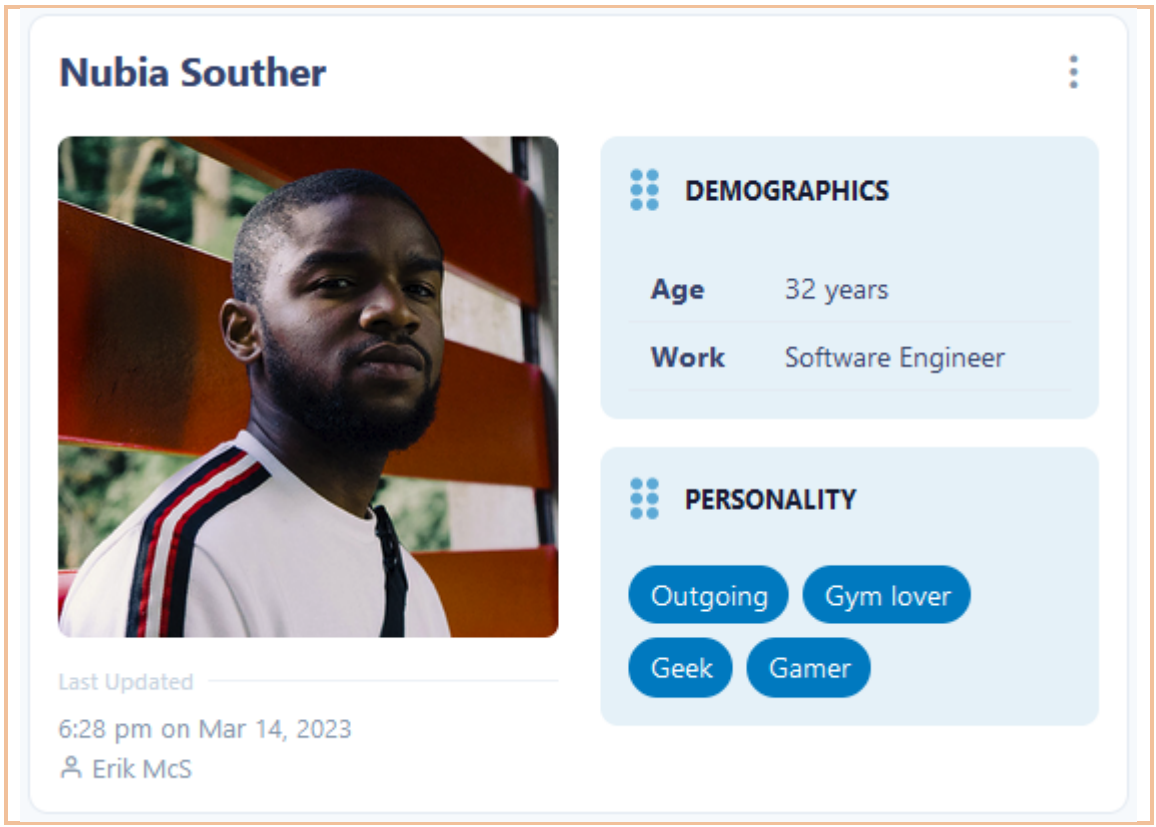
USER ANALYSIS

Designing a user interface follows steps like a SDLC, there is an analysis phase used to understand who are going to use the software, how they work and what they are expecting from it. This can be achieved by using questionnaires, interviews, brainstorming meetings, etc. Then comes the design, development and testing of the interface.

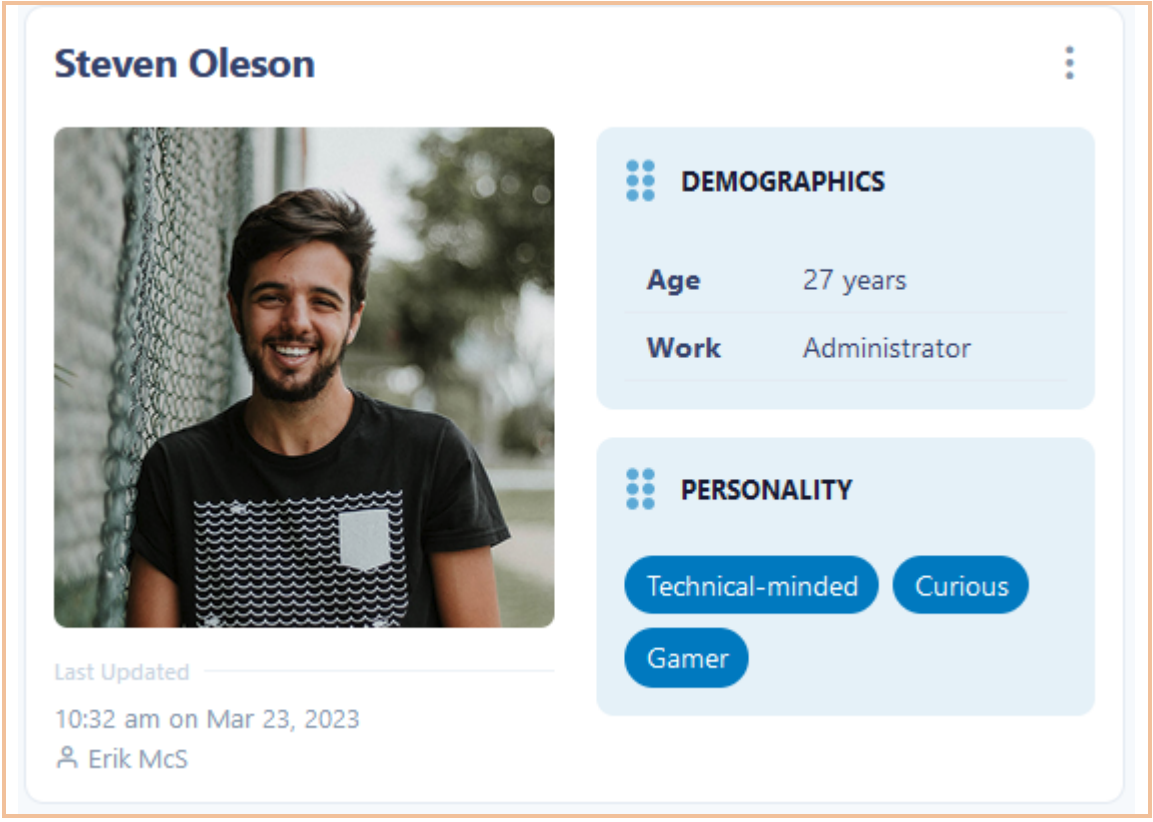
For this project, we will assume that we met some of the coaches, secretaries and members of the admin team to check what methods they used to keep track of the club data.
This allowed us to come with 3 personas (user profiles described in a card format) to describe what will be the average user of the system.

User Personas

Coach: Nubia Souther



Administrator: Steven Oleson



Fixture Secretary: Shanna Segal

(See full cards in Appendix J)

The following main points are assumed coming from the meetings with users:

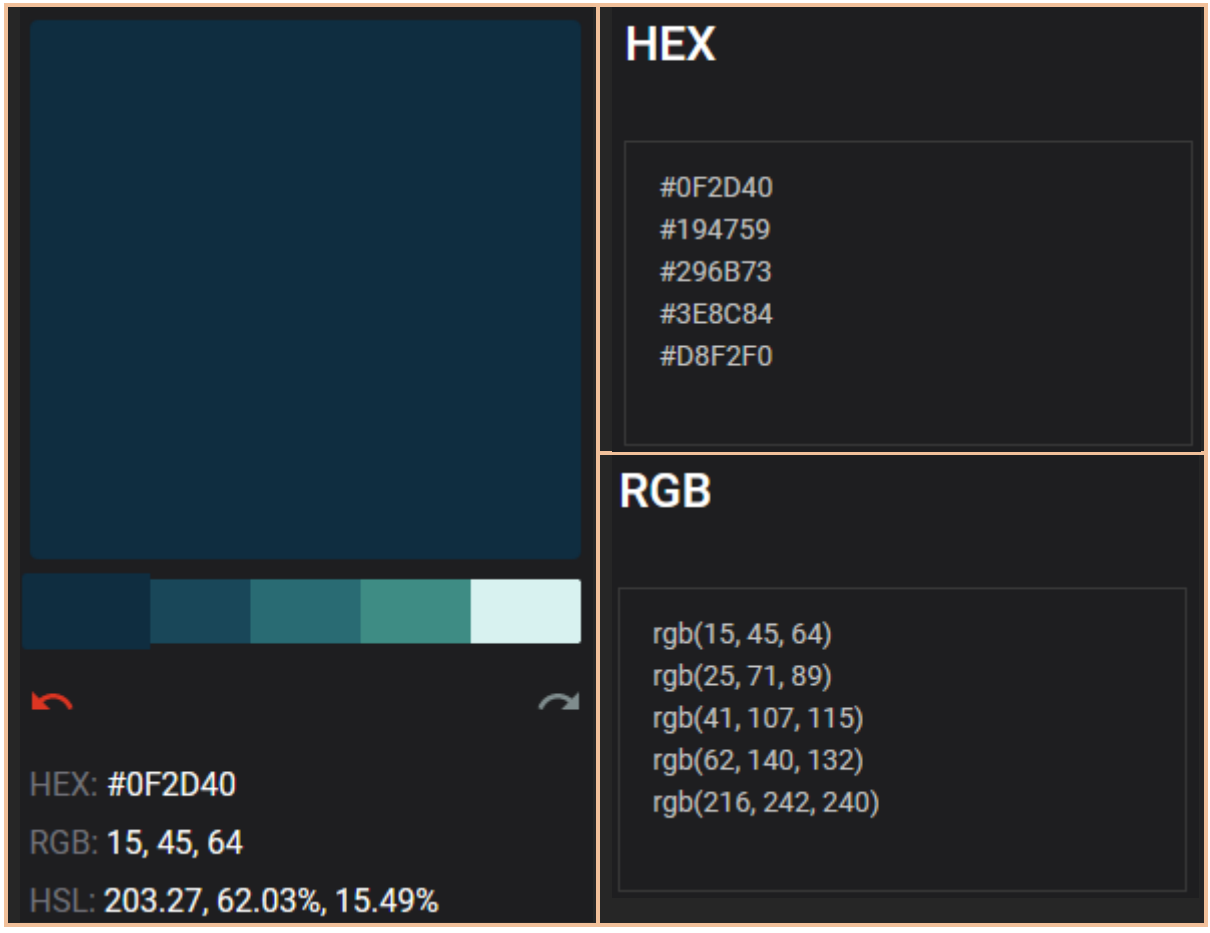
- All agreed that the paper system is past its time and that a modern solution is needed.
- Almost all want a friendly, colorful and visual user interface, except the one Unix fan in the admin team.
- Most have no preference between a web or application-based solution.
- The system must be intuitive and the interface groups tasks and actions in role groups.
- It should be reliable and fast since the point of it is to make their life easier.
- They would expect to be trained with the new system.
- And that the application covers all their needs for managing their activity.

It was decided that the solution would be proposed as an application with a graphical interface. It will try as much as possible use the club's colours and logo.
The application will group functions by roles, as defined in the analysis and Use Cases.

GRAPHICAL USER INTERFACE DESIGN

The Club's Logo

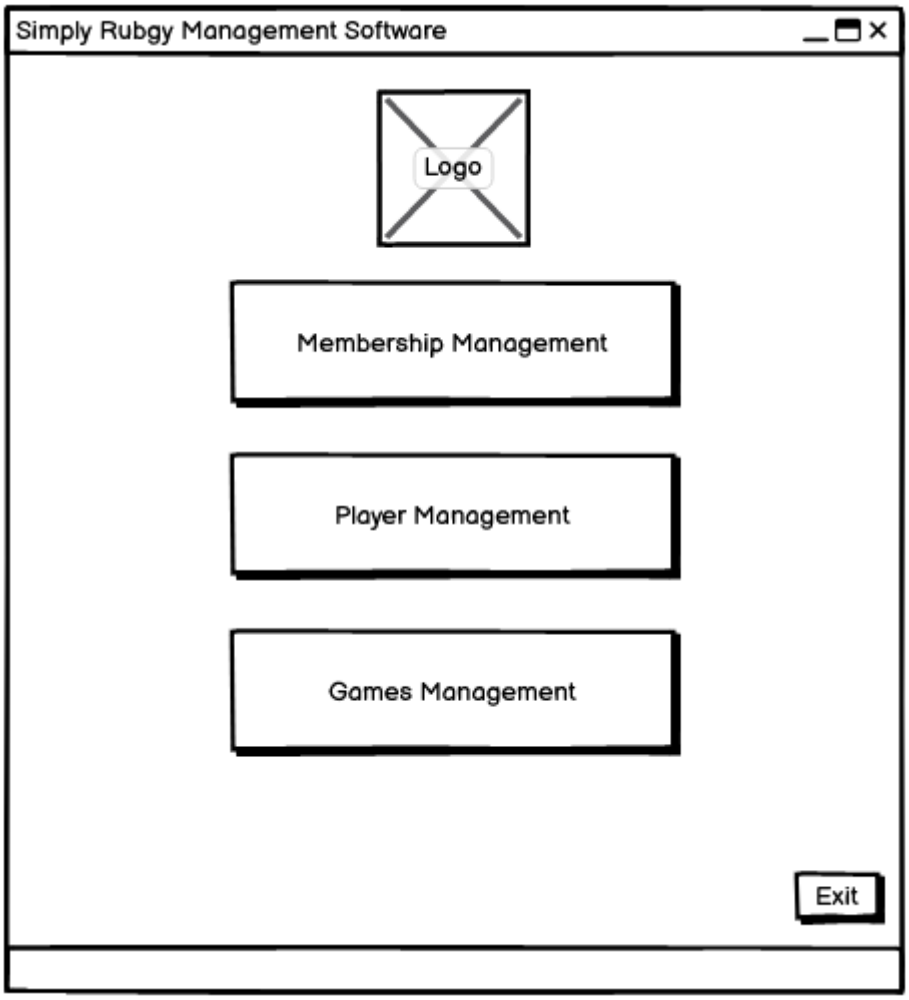
Proposed Colour Palette



APPLICATION WIREFRAMES

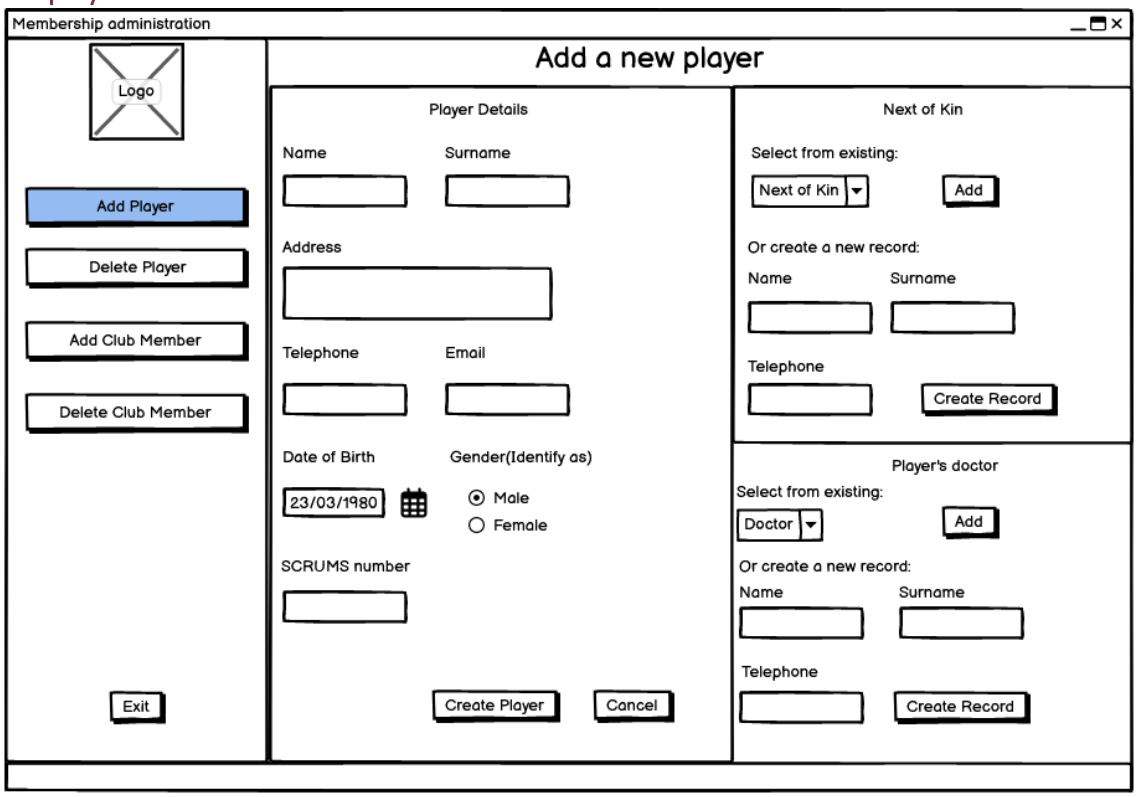
At this point, the overall look and structure of the application is designed, this will be done by using wireframes to show the different 'views' that will be developed with JavaFX.

Main menu window

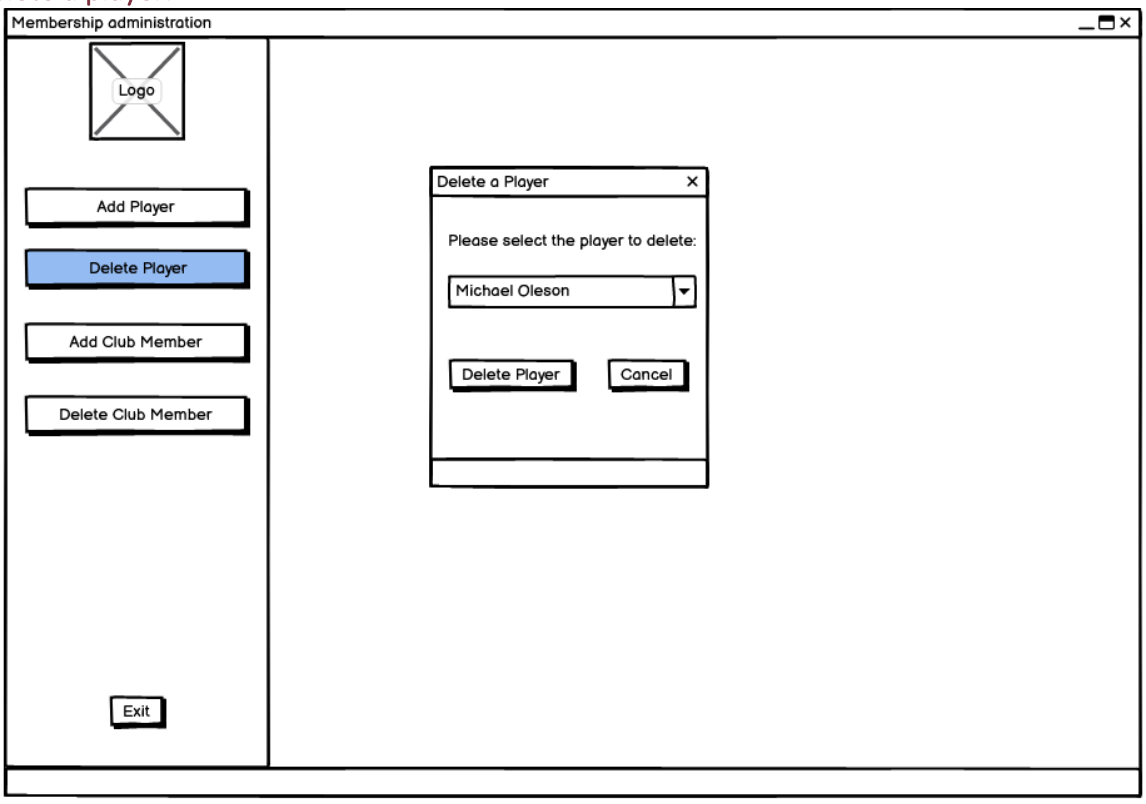


Membership Administration

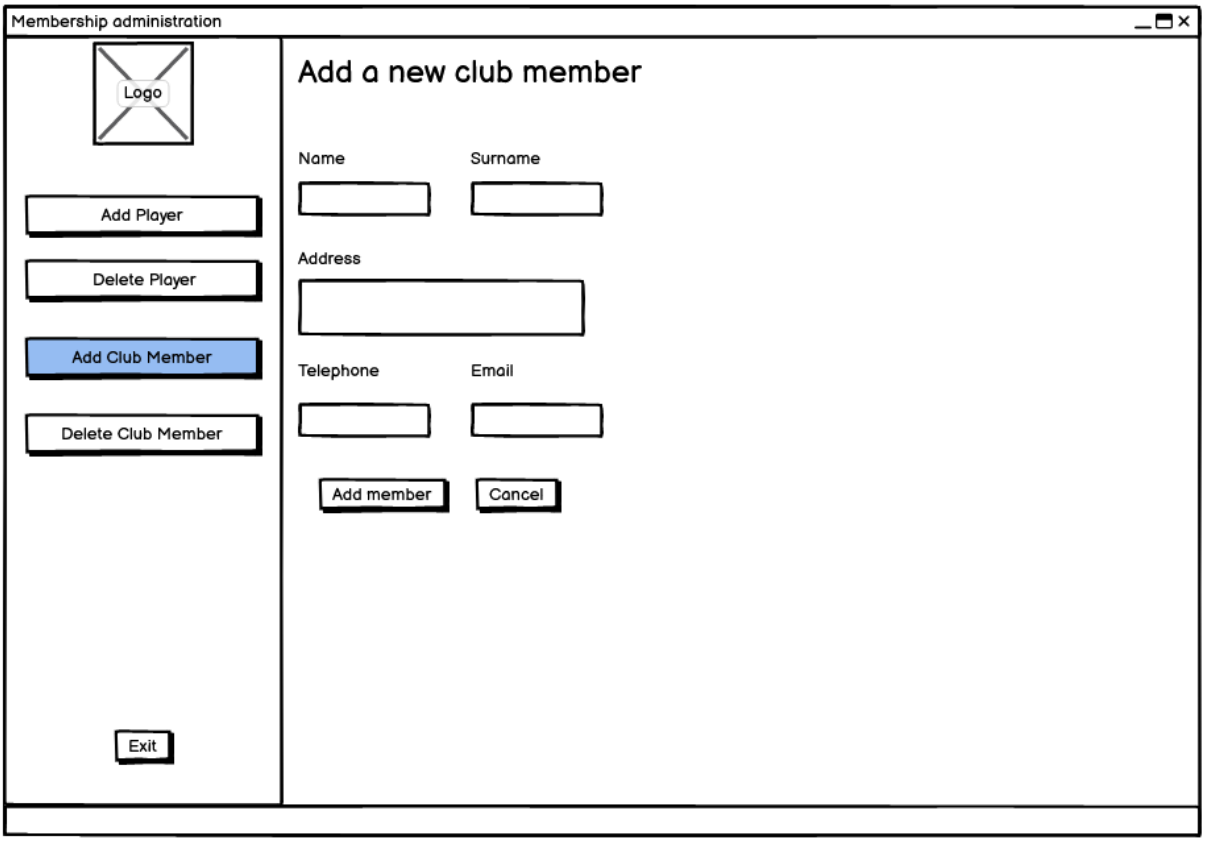
Add a player.



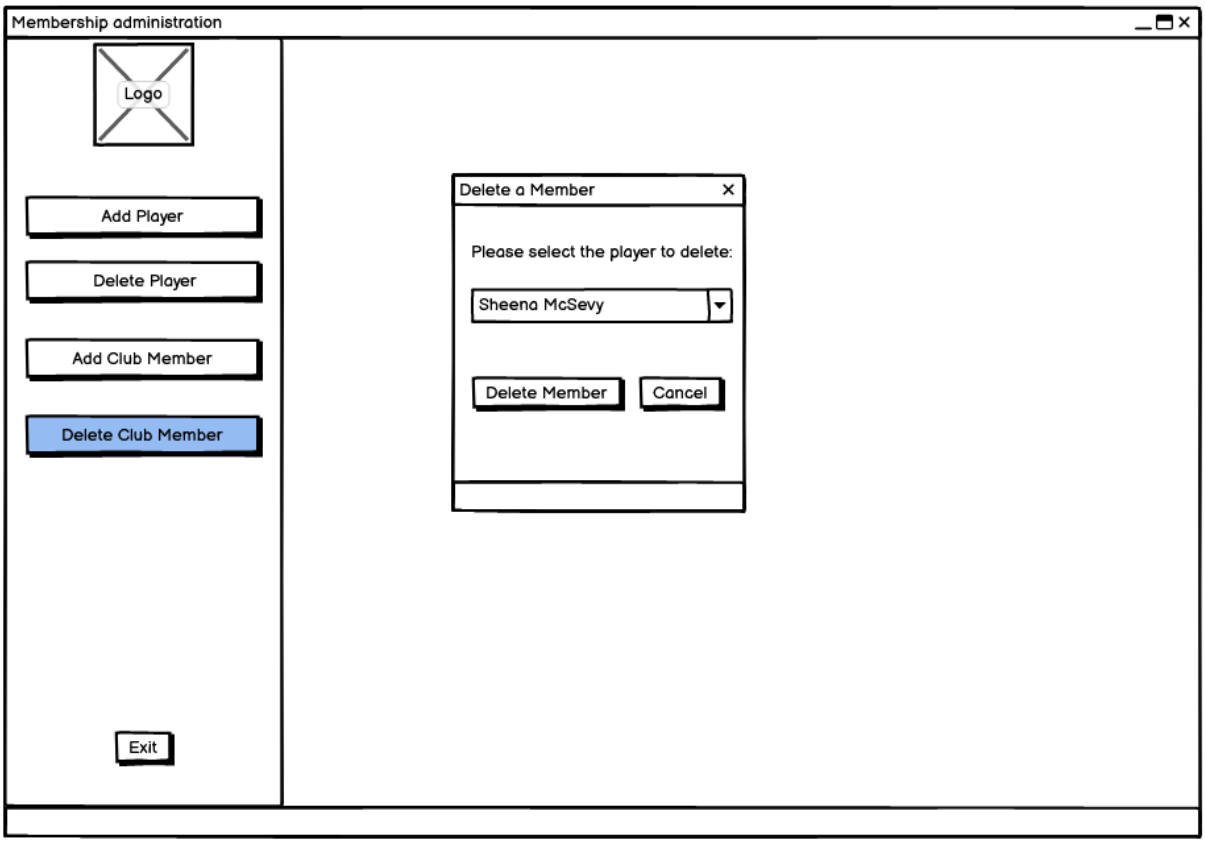
Delete a player.



Add a club member.

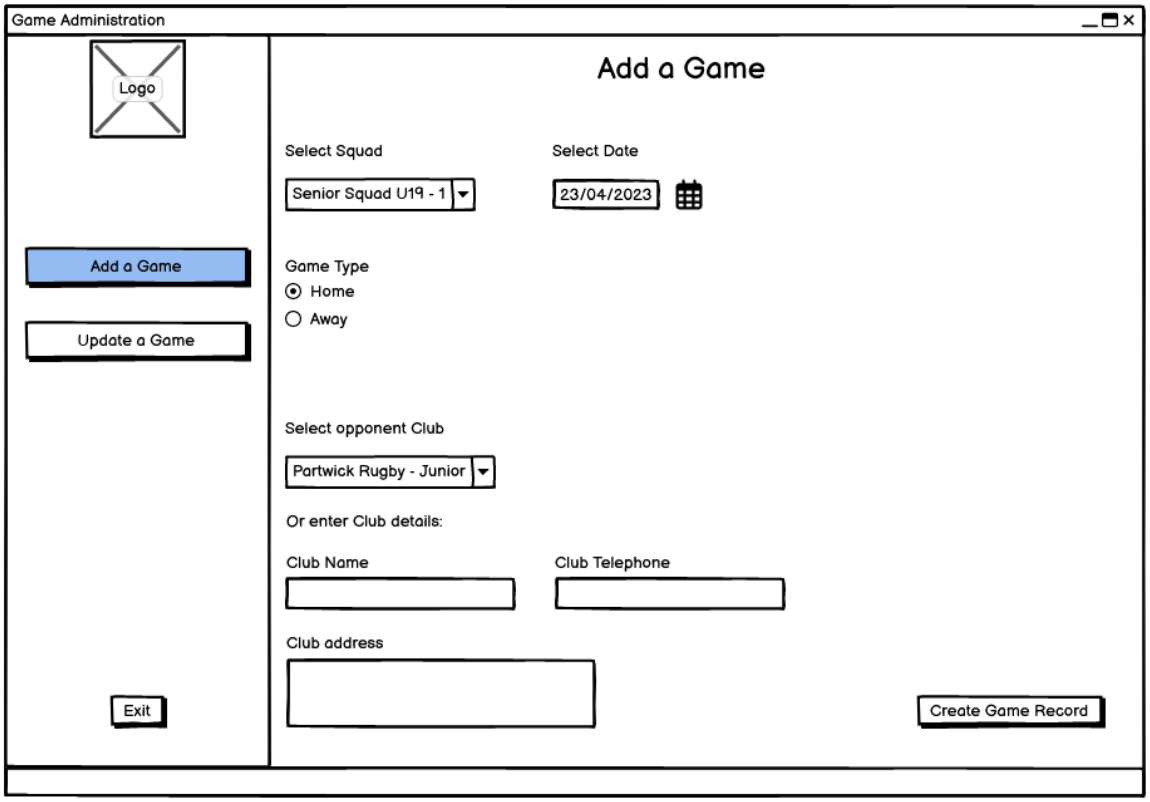


Delete a member.

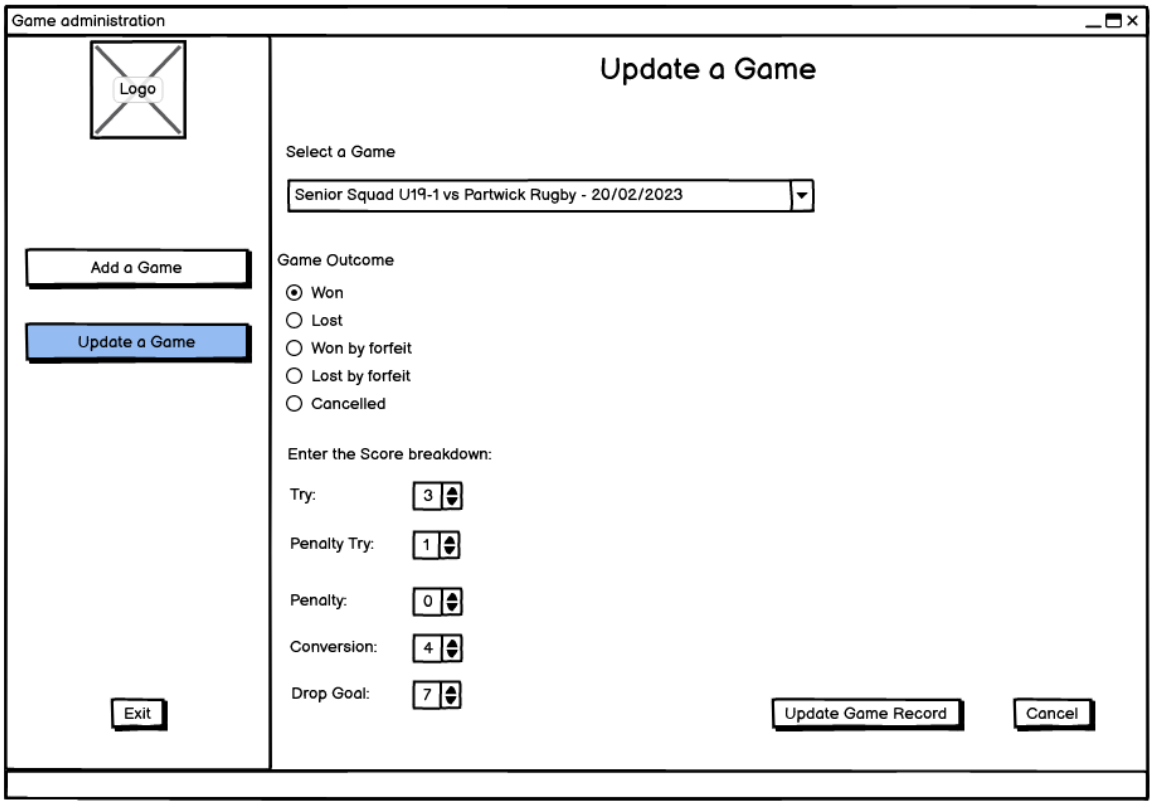


Games Administration

Add a game.

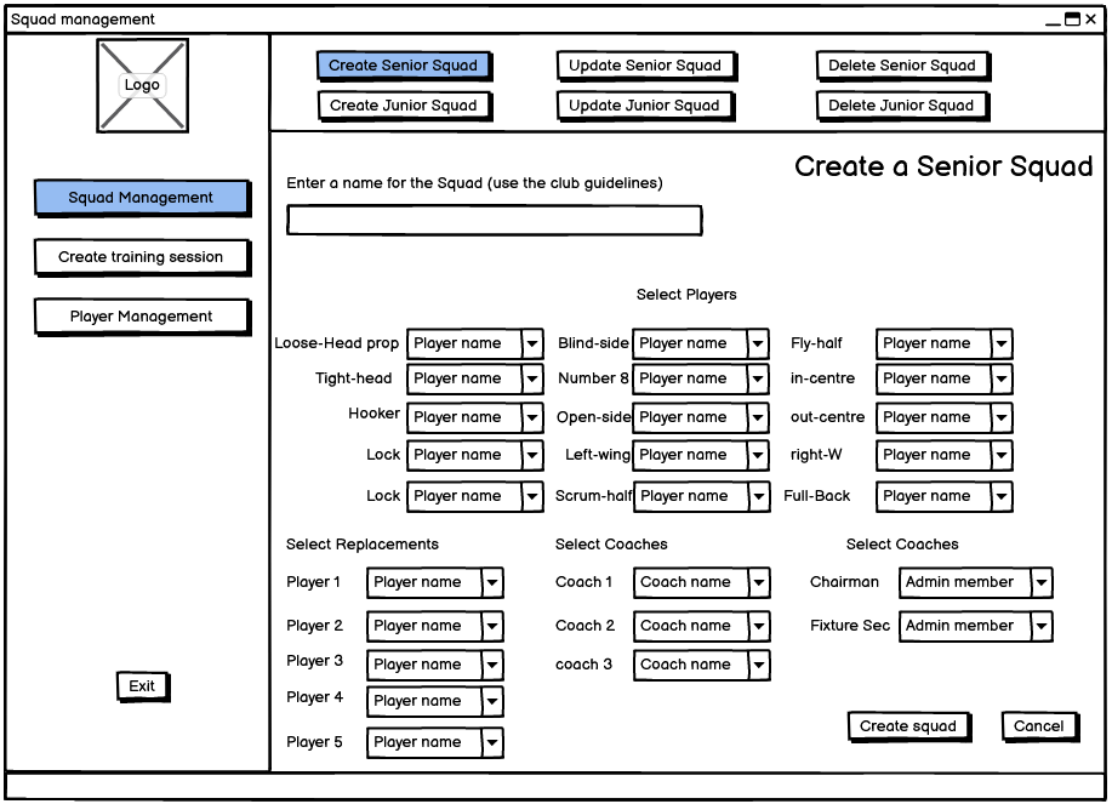


Update a game.

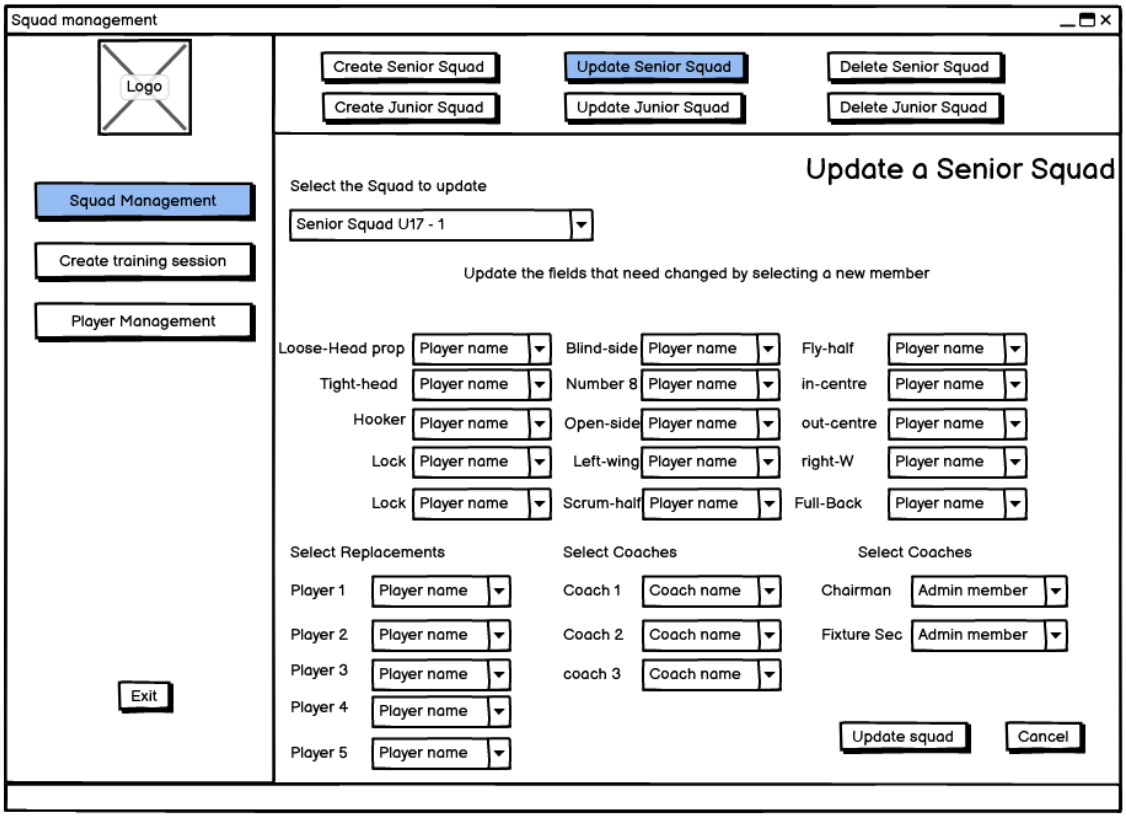


Squad Administration

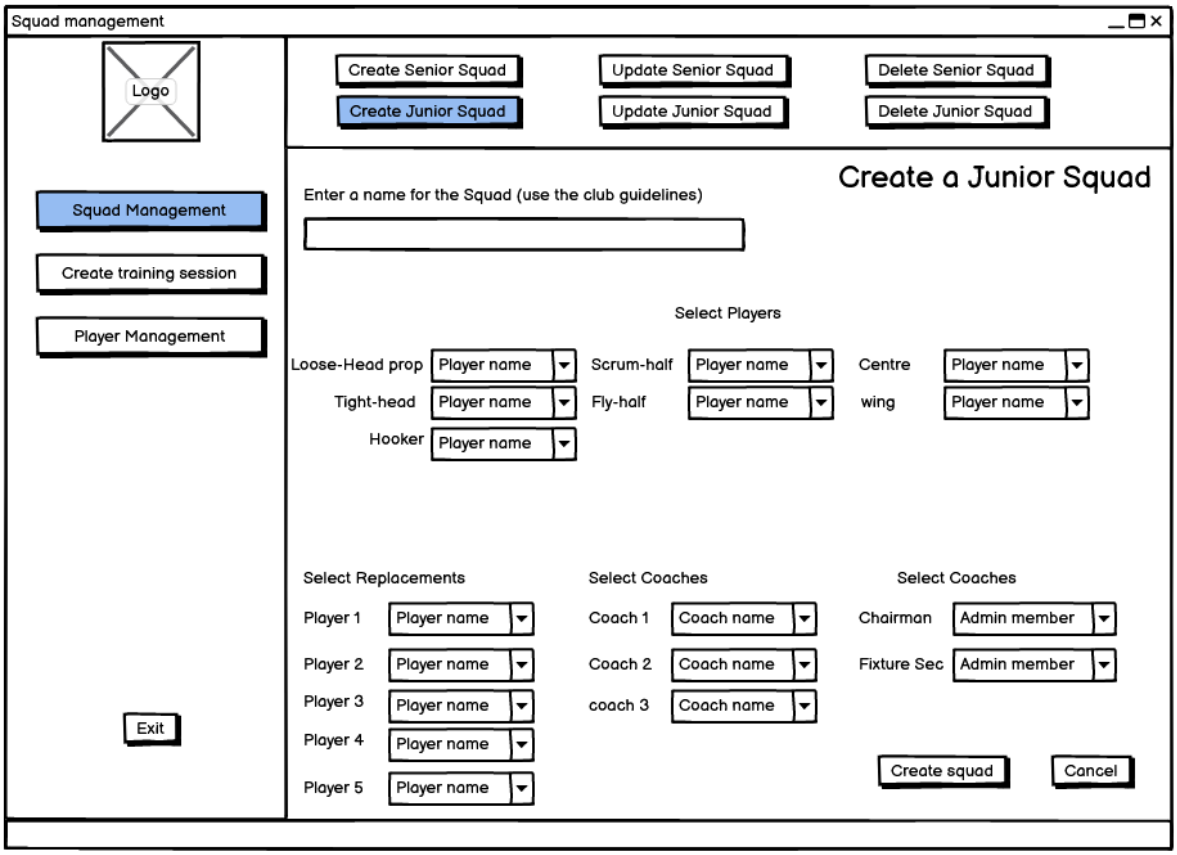
Create a Senior Squad.



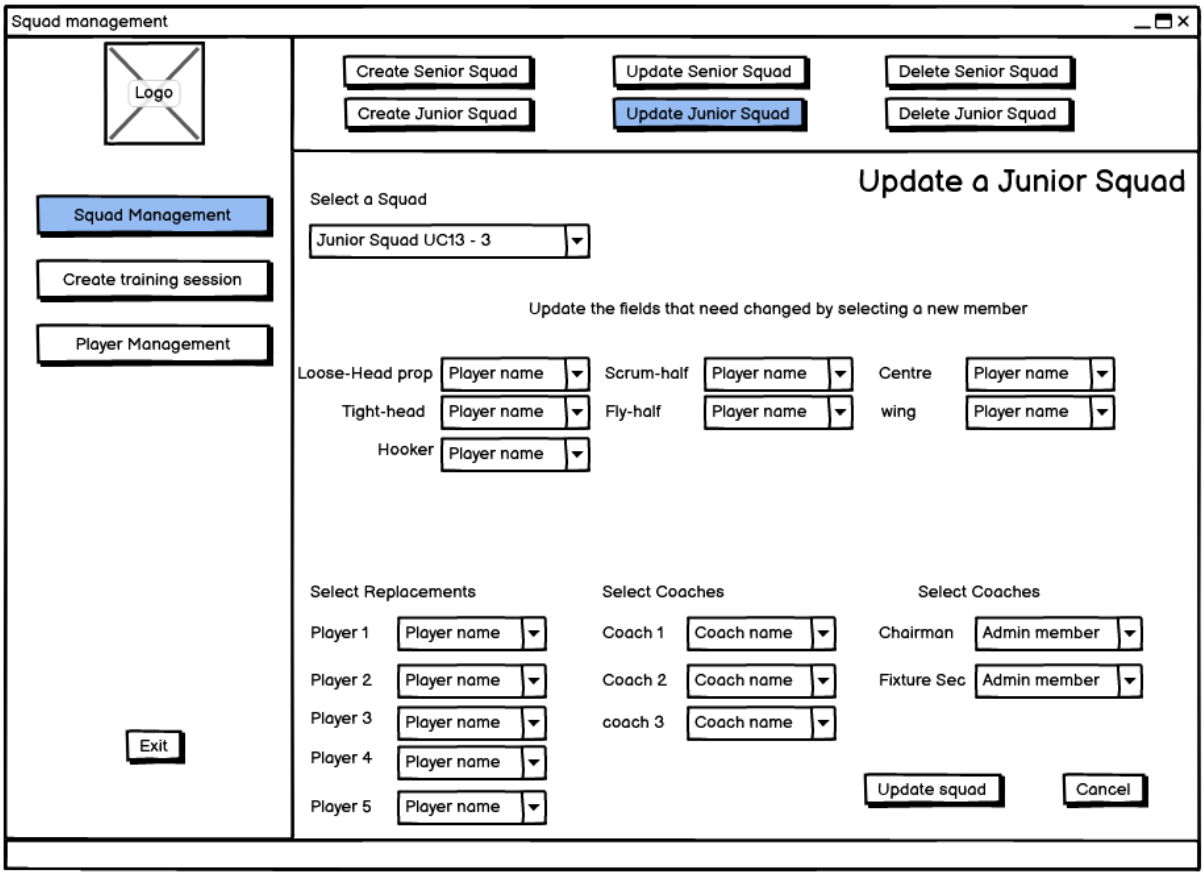
Update a Senior Squad.



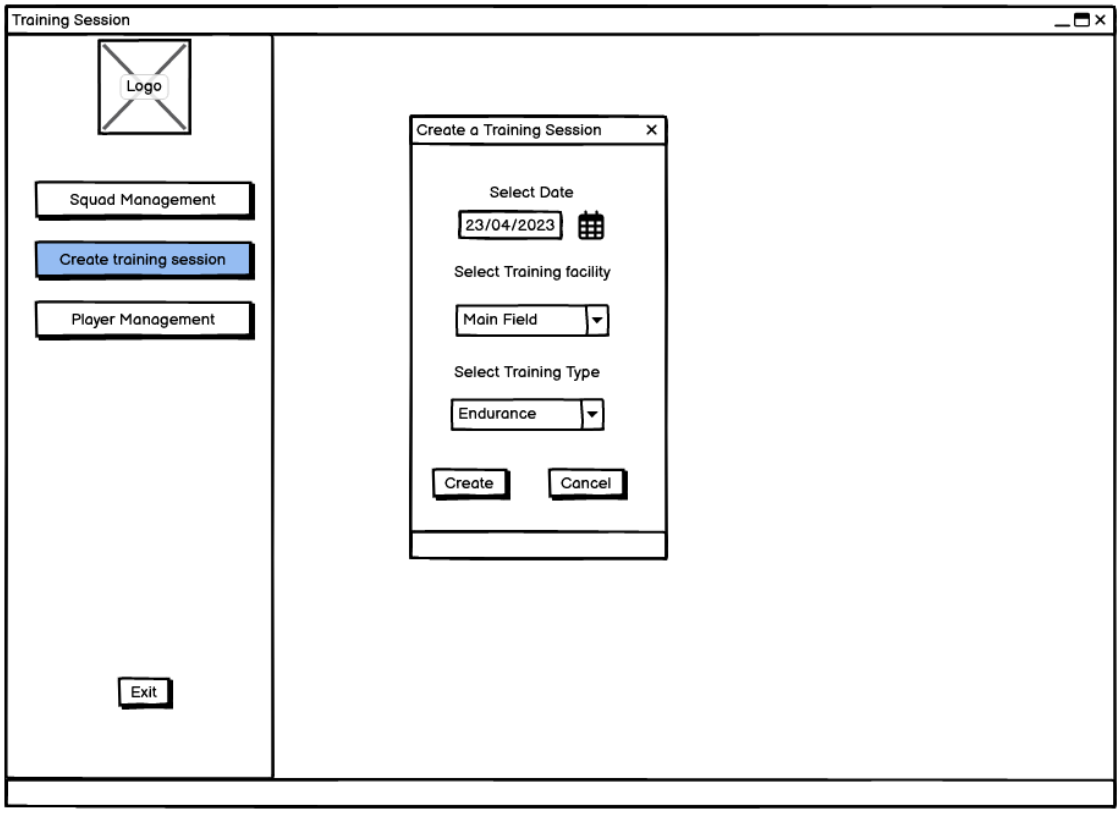
Create a Junior Squad.



Update a Junior Squad.

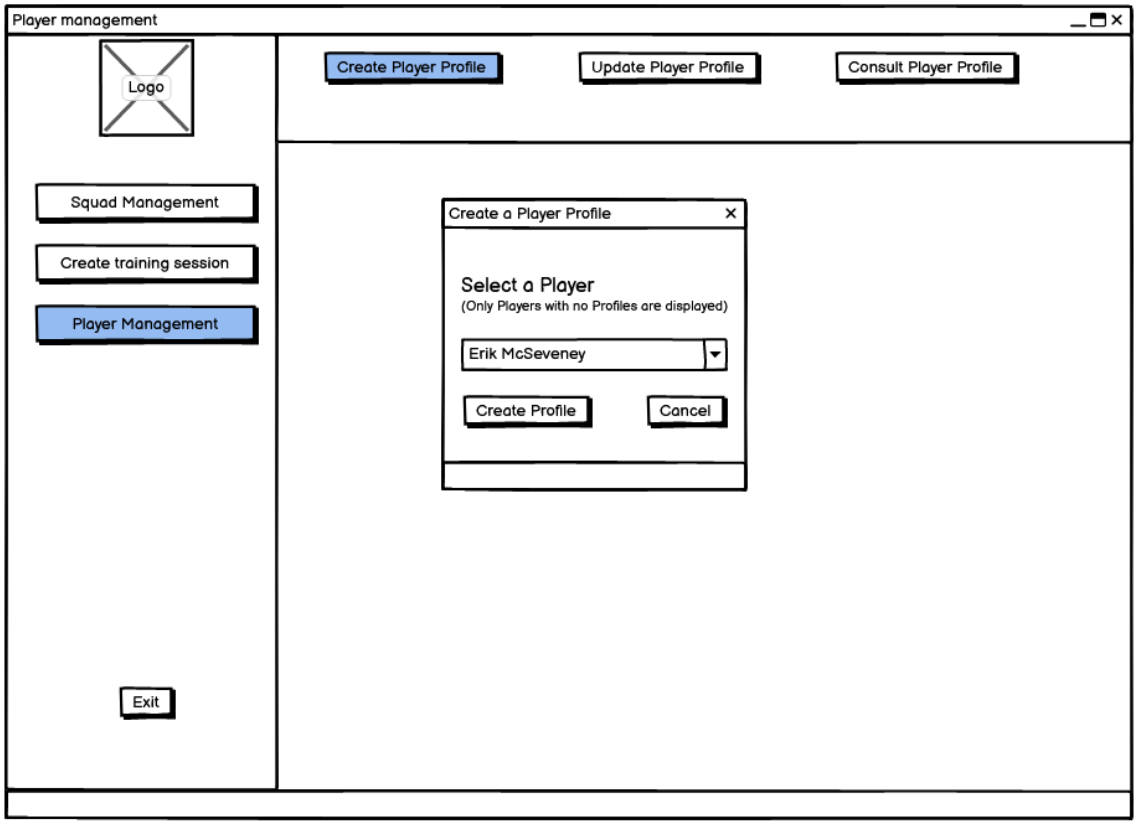


Create a Training Session.

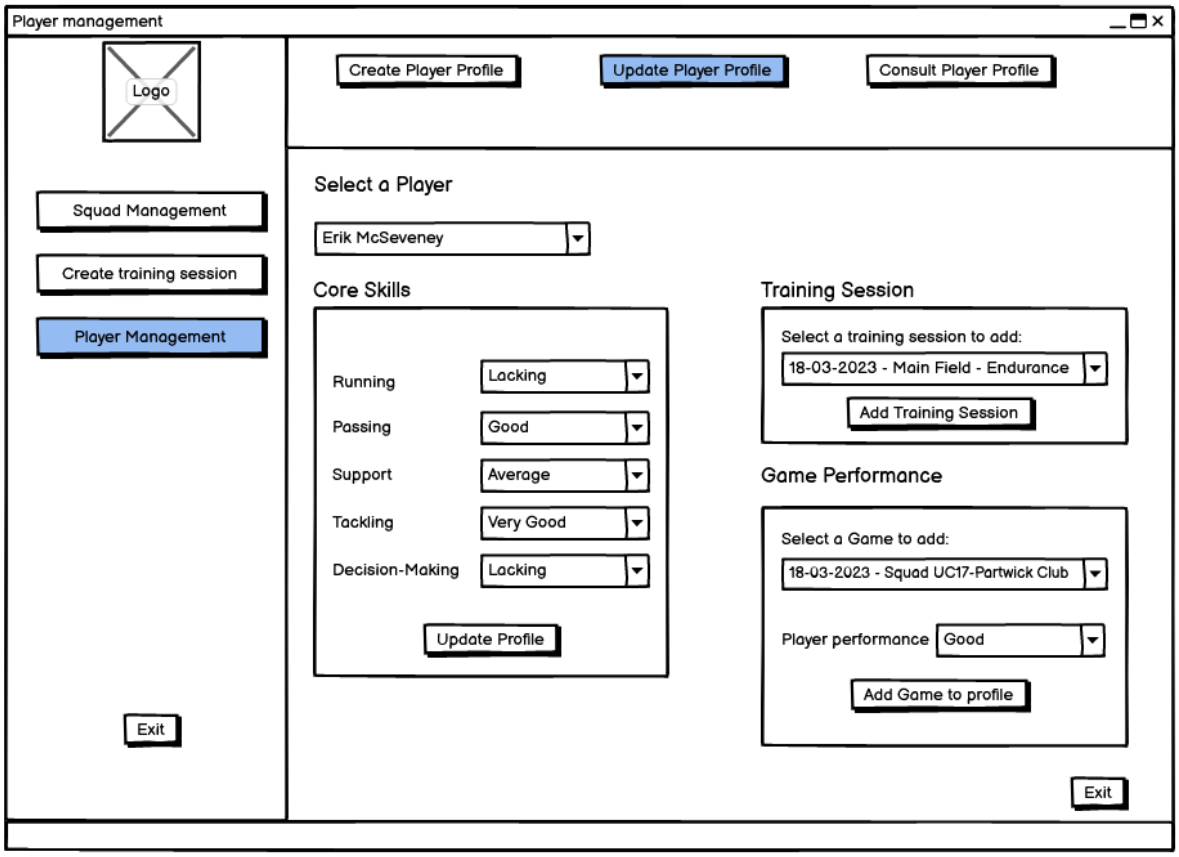


Player Administration

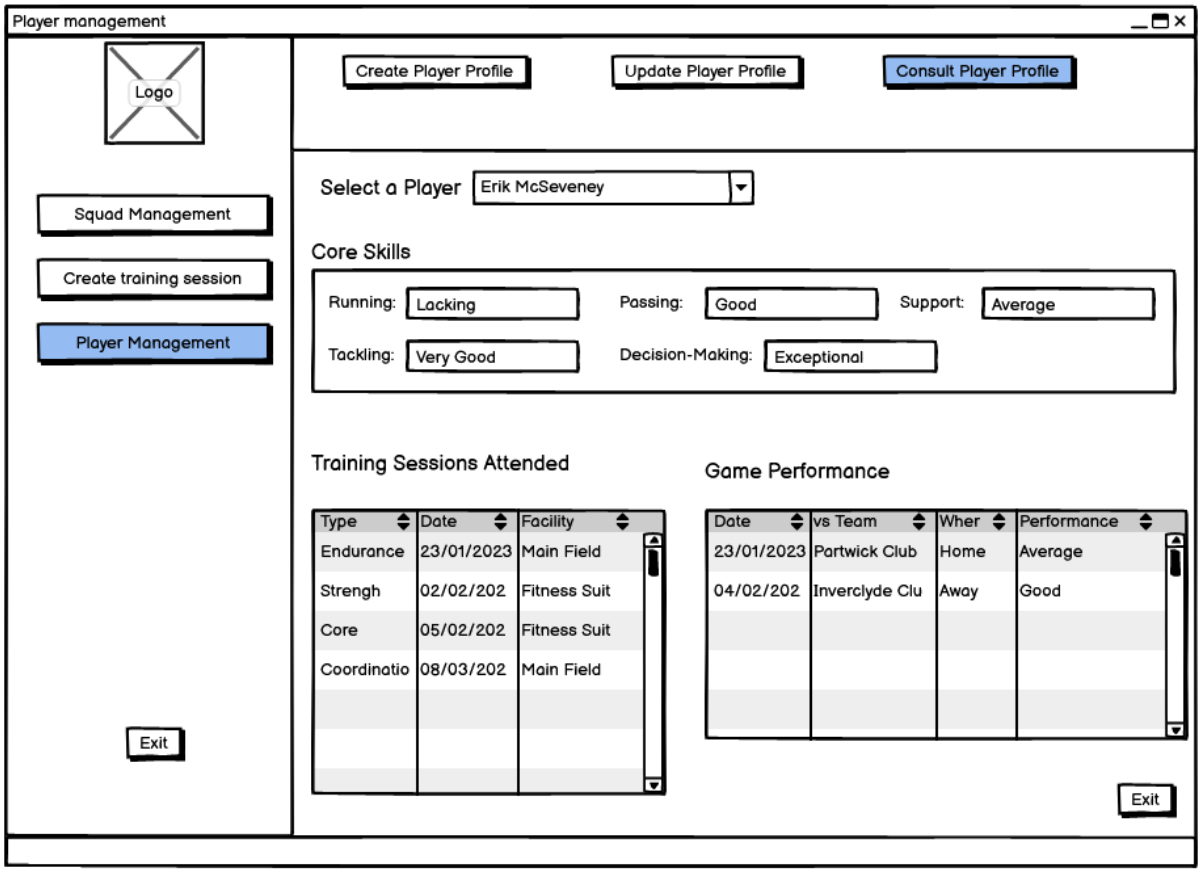
Create a player Profile.



Update a Player Profile.



Consult a Player Profile.



REFERENCES

Anand, A. (2022). *User Personas - A Complete Guide for UX Designers*. [online] UserBit. Available at: https://userbit.com/content/blog/ux-personas-definitive-guide [Accessed 23 Mar. 2023].

Cottrell, D. (2016). *Rugby Coaching Tips to Develop the Five Core Skills*. [online] Rugby Coach Weekly. Available at: https://www.rugbycoachweekly.net/rugby-coaching/team-management/develop-the-five-core-skills/.

docs.oracle.com. (2010). *JavaFX CSS Reference Guide*. [online] Available at: https://docs.oracle.com/javafx/2/api/javafx/scene/doc-files/cssref.html#intro [Accessed 21 Mar. 2023].

Jenkov, J. (2021). *JavaFX Overview*. [online] jenkov.com. Available at: https://jenkov.com/tutorials/javafx/overview.html [Accessed 21 Mar. 2023].

Maguire, P. (2018). *Guide to the GDPR for Sports Clubs*. [online] www.wrighthassall.co.uk. Available at: https://www.wrighthassall.co.uk/knowledge-base/guide-to-the-gdpr-for-sports-clubs [Accessed 21 Mar. 2023].

Performyard.com. (2021). *Employee Performance Rating Scales in 2022: Examples & Definitions*. [online] Available at: https://www.performyard.com/articles/performance-review-ratings-scales-examples [Accessed 21 Mar. 2023].

SQLite Tutorial. (2022). *SQLite Tutorial - an Easy Way to Master SQLite Fast*. [online] Available at: https://www.sqlitetutorial.net/ [Accessed 21 Mar. 2023].

SRU (2022). *Age Grade Law Variations (AGLVs)*. [online] Scottish Rugby. Available at: https://scottishrugby.org/age-grade-law-variations-aglvs/ [Accessed 20 Mar. 2023].

Stack Overflow (2022). *Stack Overflow - Where Developers Learn, Share, & Build Careers*. [online] Stack Overflow. Available at: https://stackoverflow.com/ [Accessed 21 Mar. 2023].

Steven (2019). *Rugby Positions Explained for Beginners: The Full Guide from 1-15*. [online]

Ruck. Available at: https://www.ruck.co.uk/rugby-positions-roles-beginners/ [Accessed 21 Mar. 2023].

Thomas, D. (2021). *Rugby Training: How to Catapult Your Performance in 8 Weeks (2017 Update)*. [online] Rugby Warfare. Available at: https://rugbywarfare.com/rugby-training-guide/ [Accessed 21 Mar. 2023].

W3schools (2019). *Java Tutorial*. [online] W3schools.com. Available at: https://www.w3schools.com/java/default.asp [Accessed 21 Mar. 2023].

Winter, J. (2021). *Substitutions in Rugby (Explained for Beginners) – Rugby Dome*. [online] Rugby Dome. Available at: https://rugbydome.com/substitutions-in-rugby/ [Accessed 21 Mar. 2023].

worldrugby.org (2023). *Overview | About World Rugby*. [online] www.world.rugby. Available at: https://www.world.rugby/organisation/about-us/overview [Accessed 20 Mar. 2023].

Wright, M. (2022). *Rugby positions explained: Names, numbers and what they do*. [online] Six Nations Rugby. Available at: https://www.sixnationsrugby.com/2022/02/02/rugby-positions-explained-names-numbers-and-what-they-do/ [Accessed 20 Mar. 2023].

APPENDICES

APPENDIX A – PROJECT BRIEF

Simply Rugby is a Scottish based rugby club who have squads playing at all the age grades currently supported by the Scottish Rugby Union (SRU). These include teams playing at mini, midi and senior levels. Each junior team is run by a group of coaches who are normally parents of players in the squad. They are looking for a new computerised system that will allow their coaches to keep track of:

- player details for each squad
- player skill development
- game details
- training session details

The membership secretary also wants to be able to store details of all members both for legal reasons and to allow relevant information to be sent to all members.

They will use the computerised system to keep track of their members and their player training profiles.

Further details relating to the proposed system can be obtained from the club chairman, who will be available for a discussion upon arrangement of an appointment.

APPENDIX B – PROJECT ADDITIONAL DOCUMENTATION

Simply Rugby is a Scottish based rugby club who have teams playing at all the age grades currently supported by the Scottish Rugby Union (SRU). These include squads playing at junior and senior levels. Each squad is run by a group of coaches who are normally parents of players in teams.

The club has a large membership consisting of non-playing members, senior playing members and junior playing members. All playing members must be registered with the SRU to ensure that they are adequately covered by the SRU's insurance schemes. Junior members must also have a consent form signed every season by one of their legal guardians.

A coach or parent in every age group keeps track of all the player details in that age group including doctor and next of kin/guardian details. Each section (junior and senior) also has a chairman and fixture secretary who is responsible for organising fixtures.

The coaches keep records of player profiles for their squads which they use to help organise appropriate training sessions. These profiles include details of the training that players have undertaken and an estimate of the level of skills acquired in various aspects of rugby. They also record training session activities and game details including how each player performed in the games as well as the scoring details.

They also use these profiles to help judge which players should be put forward for pathway development trials and to help ensure that they plan appropriate development activities to allow each player to achieve their potential. This is particularly important in rugby where

different positions require different skill sets. The most appropriate position for a player may not manifest itself until after many years of playing at different positions.

APPENDIX C – CLIENT INTERVIEW

General/Club

Do the club already store the member details in a similar system (paper, excel, etc.)?
At the moment, the coaches / parents have been responsible for storing details relating to their own groups so various methods have been used.

If so, is the club compliant with the GDPR/Data Protection Act 18? (Privacy notice, consent forms, processes for requests, etc.)
We currently require junior members to have signed consent forms.

If compliant, do this related documentation needs to be tracked in member profiles?
We would like to keep records to show who has had consent forms signed.

Do membership fees need to be tracked?
No, membership fees are currently tracked by a different system.

If so, are they age grade dependent, what is the frequency for payment, can you pass us fees table/details?
N/A

We are assuming that Coaches need to be identified when filling games or training sessions, do the club need to also track their details?
Yes, we would like coach / parent details to be stored.

If so, please refer to the Coaches section (Ignore otherwise)

Do the club have access/owns one or more training facilities? If so, could we get a list and type? (Ex: Fields, gym Facilities, etc.)
No, we currently only have one training facility which has a playing field, gym, swimming pool and cafe.

Who is responsible to create/update/delete members profiles? do they need those functions integrated in the system or a direct access to the database?
This would be the responsibility of our admin team. We would leave the design details to you in relation to how they are able to implement this.

Are you aware of the system's future user's computer literacy levels, if any specific training other that on the proposed solution would be needed?
Our admin team have used computers before and are comfortable with excel etc however training on how to use the system would be helpful.

Do the club own a computer that will host the proposed solution? If not, would you like us to integrate it in the project?
We do own a computer however we would be happy to upgrade this if you thought it would be appropriate to do so.

Players

Do players need a recurring health-check (yearly, for ex) to play?
Yes, players currently undertake a health check however this is handled by the SRU within the registration process.

If so, does this need to be tracked in the new system in a player profile?
Just need to show that they are currently registered with the SRU.

Beside the common details stored about a person (Names, telephone, address, email, date of birth), is there any other detail the club need/want to track? (The age grade is likely to be tracked with the Squad details at this point).
Please refer to additional information

Coaches (if needed)

All Junior coaches are required by the SRU for 2022/23 to: be registered on the SCRUMS system, completed the RugbyRight training and be a member of the Protecting Vulnerable Groups scheme. does the club need to track those requirements?
No.
If so, in 2024, the SRU will also requires a recognized Coach qualification. Does the new system need to be ready for the change?
At the moment, there are no plans to have this added to the system.

Games

Does the new system need to track the game venues? and if so, its type, details? (Home, away)
We only need track if the games were played home or away however if possible the away team's location / details would be helpful.

A game involves another club's squad, do you need to keep those clubs' details, or squad details in the system/DB?
No there are no plans to track any other clubs' details.

Training Session

Do you need to keep a register of players present?
It would be useful to know if the player attended the training session.

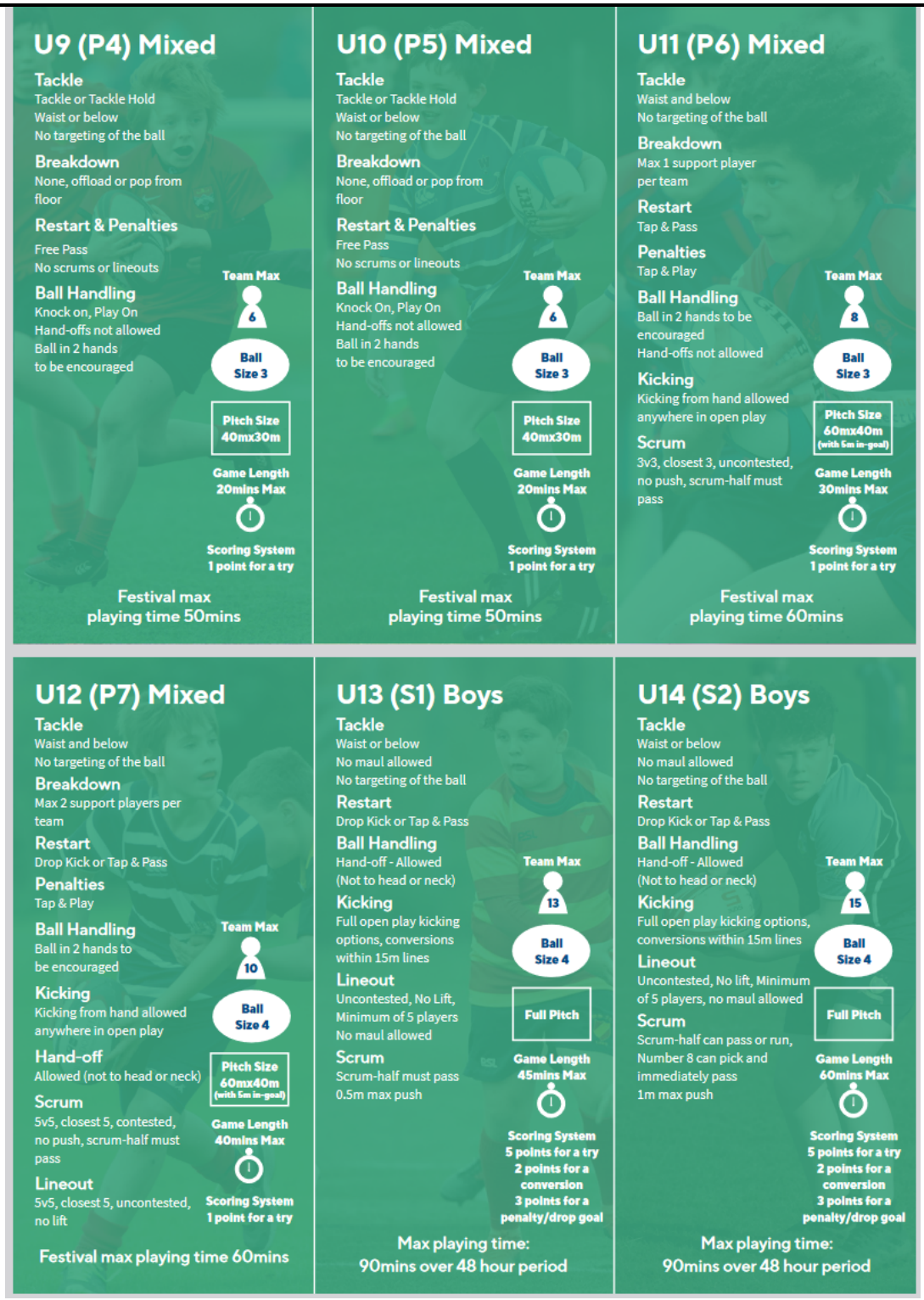
Do you need to track which asset was used for the session? (Field, gym, etc.)
No need to track exactly what asset was used just a description of the training session activities that were undertaken.

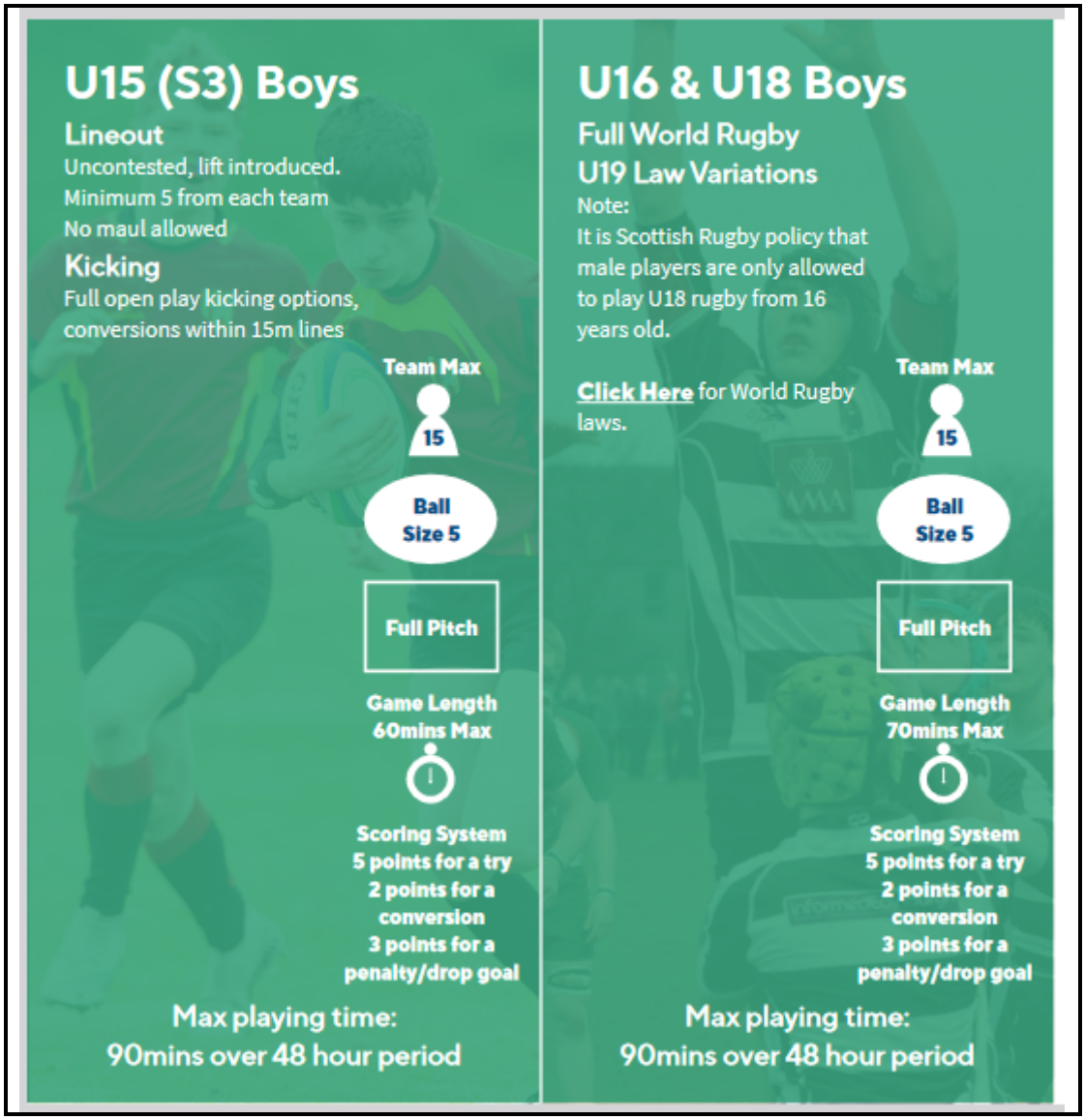
Broadly, it is agreed that there are 5 main skills in rugby: Passing, Running, Support play, Tackling and Decision making. Which is part of trainings and can be used for player profiles too.
Are there any other skills that you would like to keep track?
Not at the moment.

Training profile

this is a very open requirement, could you detail what do you which to track in this profile? Do you already use a system to track those profiles? are you able to share a template for example?
Please refer to additional information.

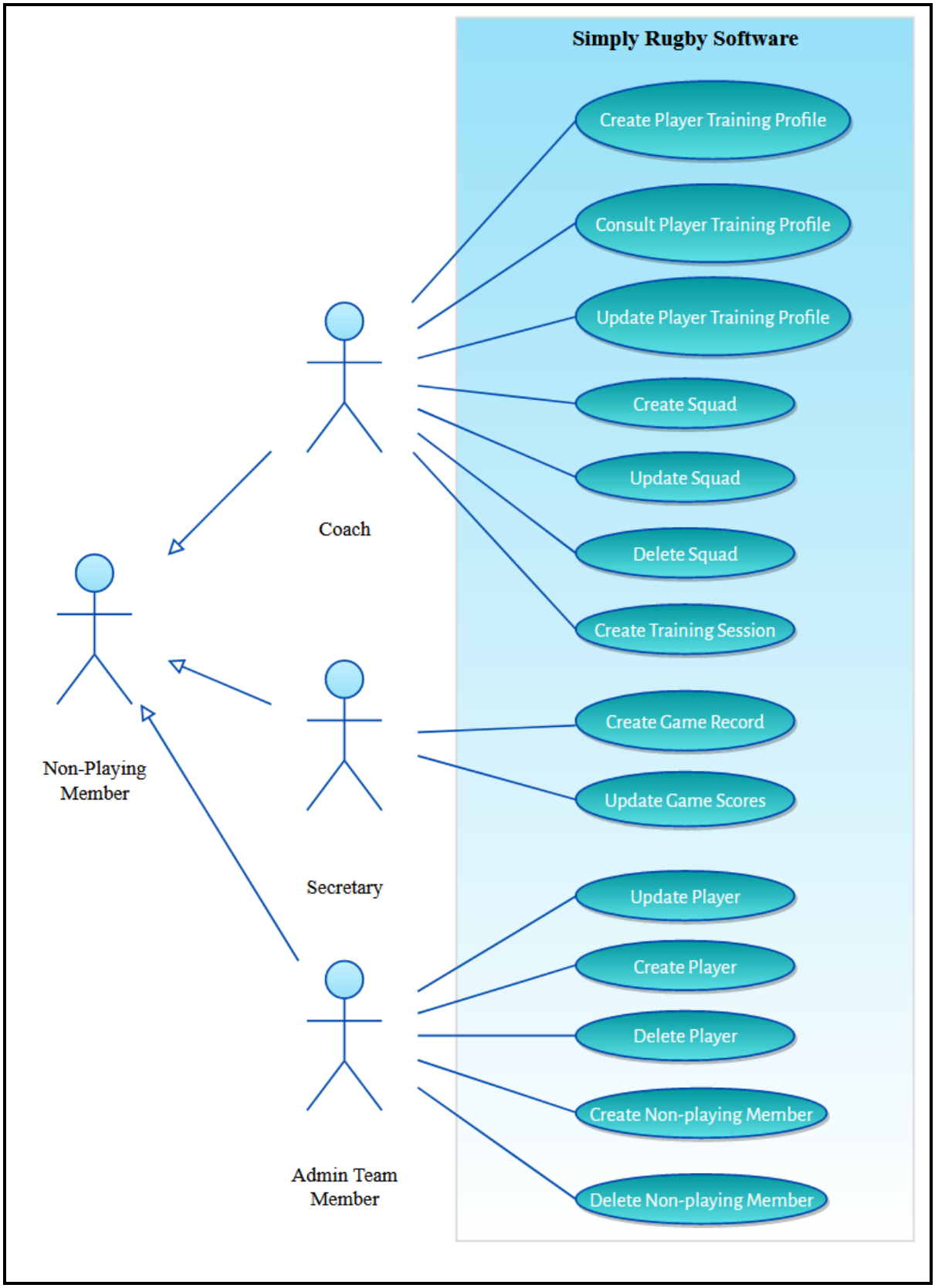
APPENDIX D – AGE GRADE LAW VARIATIONS



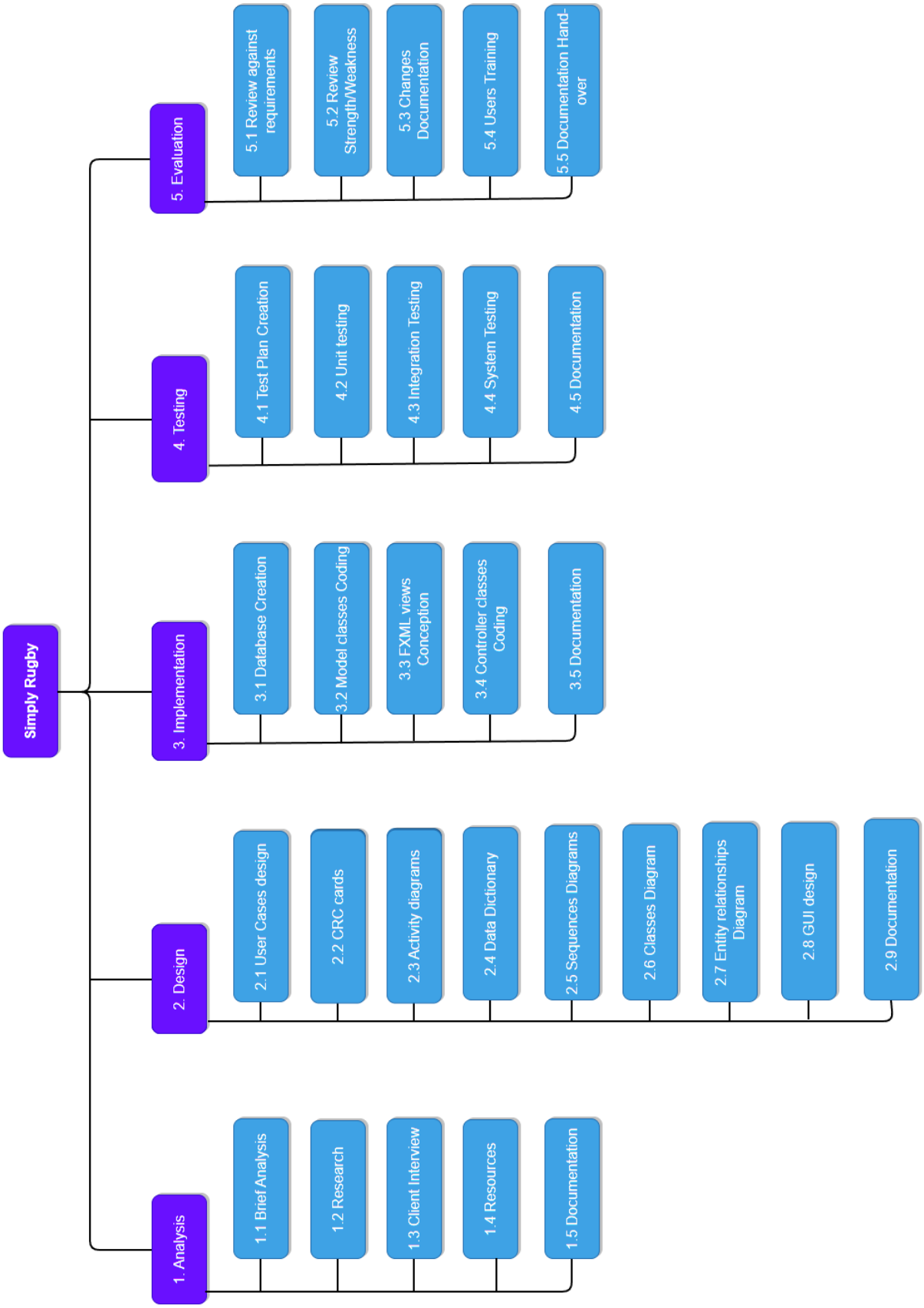


(Scottish Rugby, n.d)

APPENDIX E – GENERAL USE CASE DIAGRAM

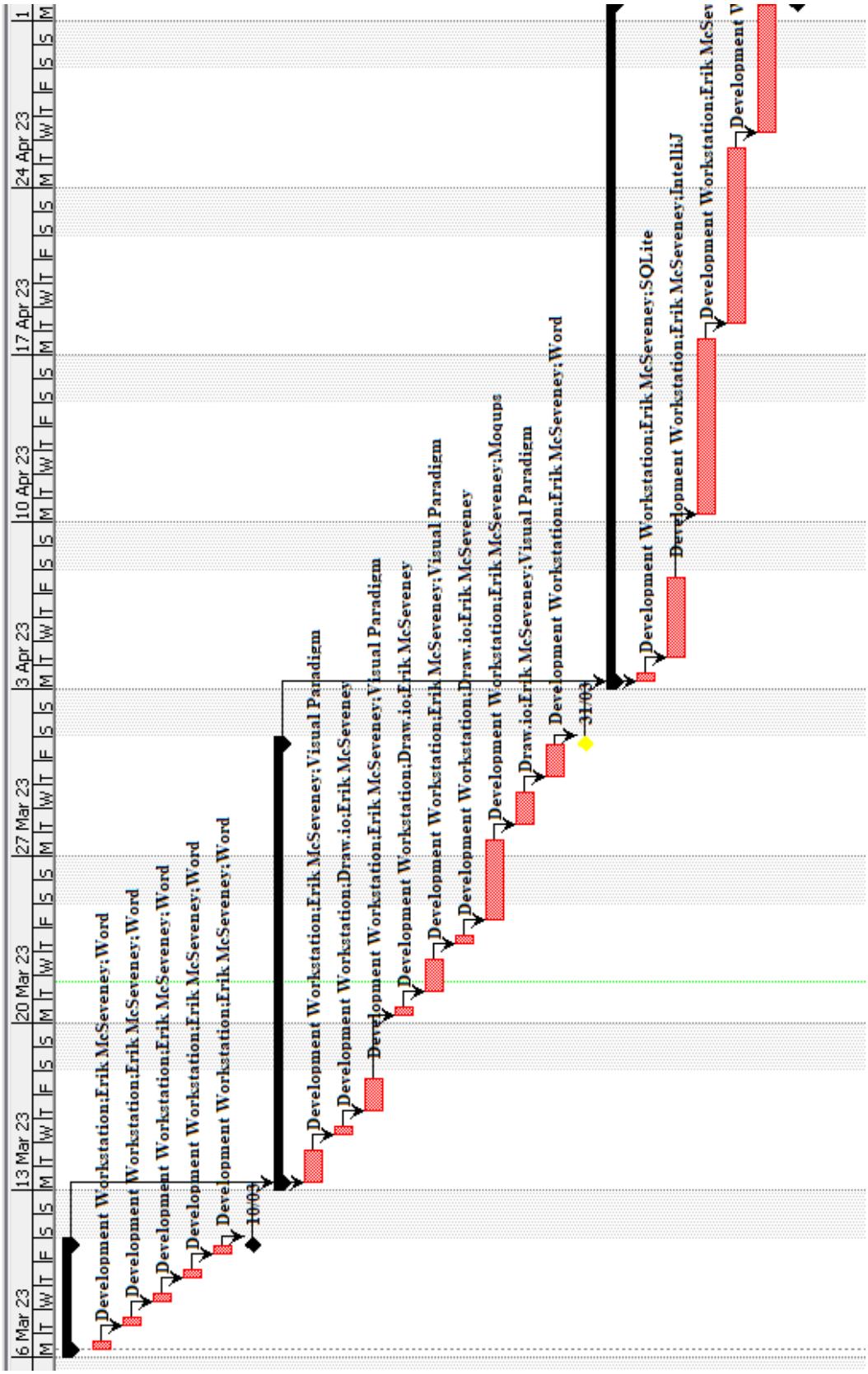


APPENDIX F - WBS

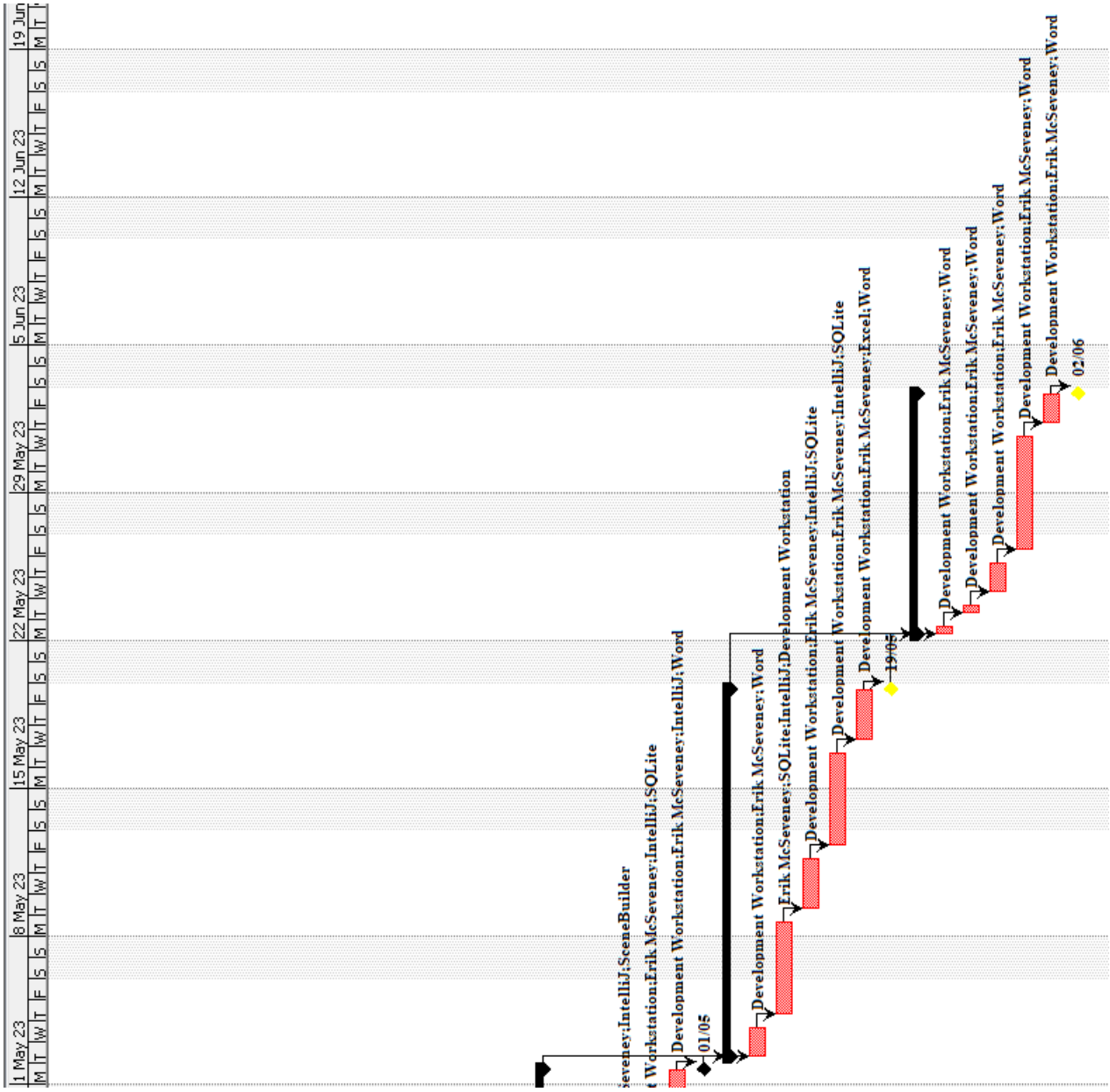


APPENDIX G – GANTT DIAGRAM

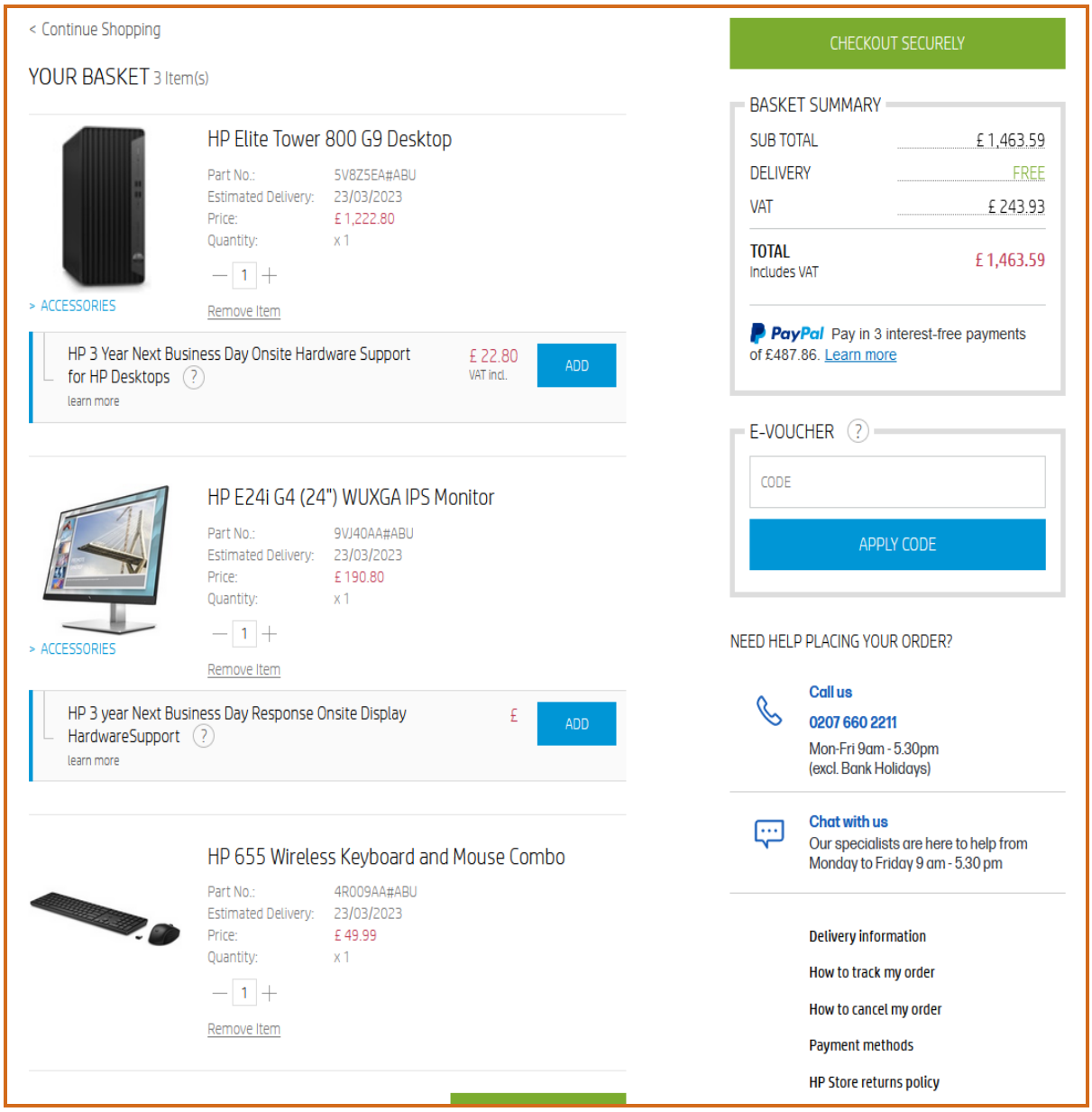
PART 1



PART 2


1 May 23
8 May 23
15 May 23
22 May 23
29 May 23
5 Jun 23
12 Jun 23
19 Jun
Development Workstation:Erik McSeveney:IntelliJ;Word
01/05
Development Workstation:Erik McSeveney:Word
Erik McSeveney:SQLite:IntelliJ;Development Workstation
Development Workstation:Erik McSeveney:IntelliJ;SQLite
Development Workstation:Erik McSeveney:IntelliJ;SQLite
Development Workstation:Erik McSeveney:Excel;Word
19/05
Development Workstation:Erik McSeveney;Word
Development Workstation:Erik McSeveney;Word
Development Workstation:Erik McSeveney;Word
Development Workstation:Erik McSeveney;Word
02/06

APPENDIX H – OPTIONAL PC DESKTOP



APPENDIX I – ENTITY RELATIONSHIP DIAGRAM

Due to the size, a full picture of the ERD is attached as a file on Canvas.

APPENDIX J – PERSONA CARDS

NUBIA SOUTHER

TECH

Internet

Social Networks

Games

Online Shopping

Coding

DEMOGRAPHICS

Age	32 years
Work	Software Engineer

GOALS

- Would like to win the club's cup this year

TAGLINE

Ok, no worries, we will beat them next time!

PERSONALITY

Outgoing | Gym lover | Geek | Gamer

BIO

Discovered the game early in life but developped a taste in coding and choose it over a career in sport.

Nubia decided to give time to coach young people in the sport he loves.

SHANNA SEGAL

DEMOGRAPHICS

Age	43 years
Work	Retail

TECH

Internet

Social Networks

Games

Online Shopping

GOALS

- See her oldest son playing in the Senios squad.
- Get that promotion at work.
- understanding what is all the fuss about Fortnite.

TAGLINE

No, I can't move you to the field before your coach agree to it.

PERSONALITY

Detail-oriented | Friendly | outgoing | dignified

BIO

Mother of two. the oldest soon tried rugby at school and decide it was for him. joined the club as U14 and been playing since.

Invested herself with the club to share more time with him and discovered she enjoyed participating in the club's fife.

STEVEN OLESON

DEMOGRAPHICS

Age	27 years
Work	Administrator

TECH

Internet

Social Networks

Games

Computers

GOALS

- To see the end of paper forms in the club.

TAGLINE

Sorry, too busy right now.

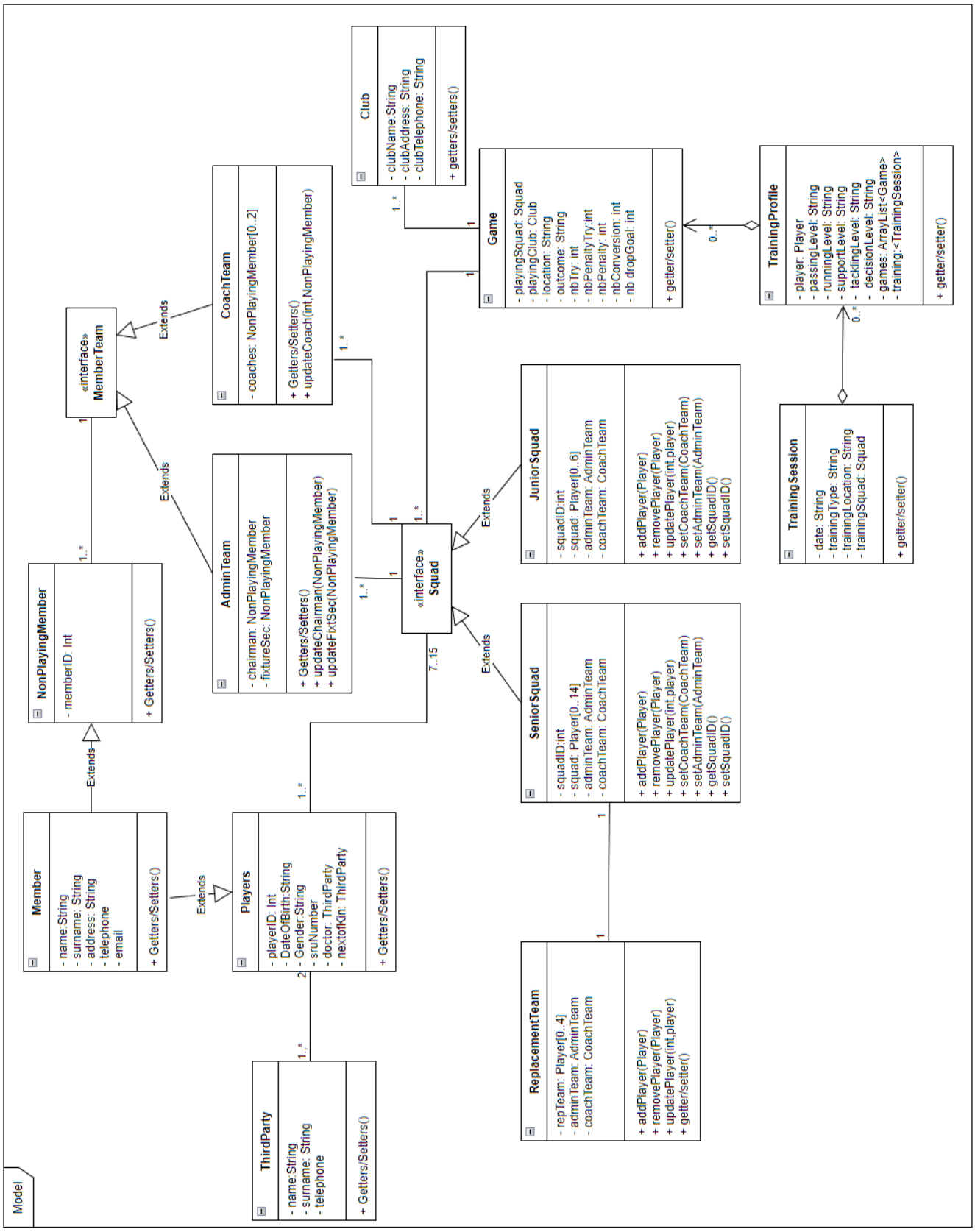
PERSONALITY

Technical-minded | Curious | Gamer

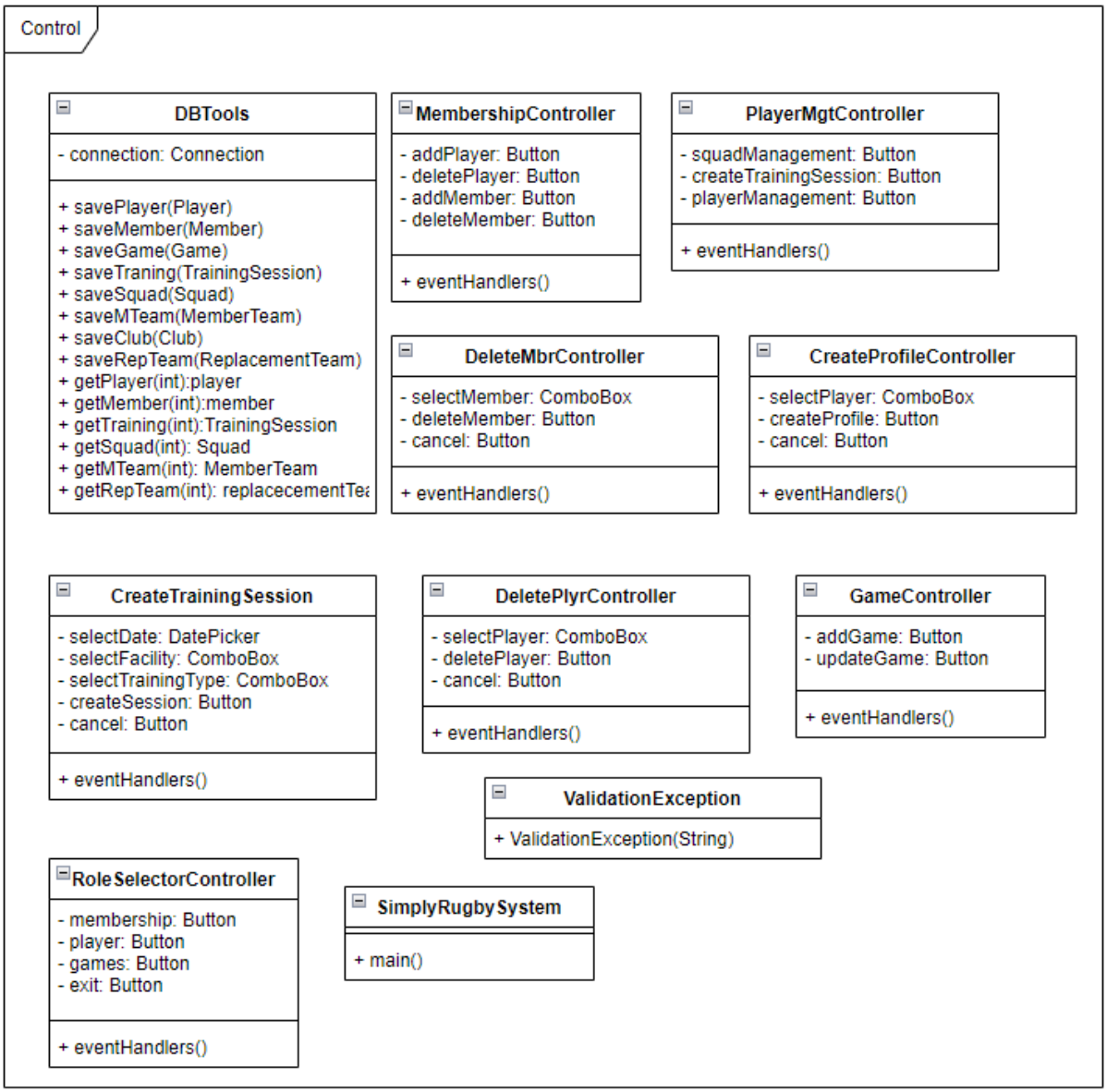
BIO

Studied computers and have a liking for rugby, joined the club admin team to join these two passions.

APPENDIX K-CLASS DIAGRAM



(The full diagram is attached on canvas)



The class diagram is also attached separately.