

Detailed Design Document

STUDENT NAME: KIAN GAULT–HND SOFTWARE DEVELOPMENT–GRADED UNIT



oop solutions ltd

Simply Rugby LTD

Contents

[Purpose 2](#_Toc128605063)

[Business view Designs 3](#_Toc128605064)

[Static Diagrams: 3](#_Toc128605065)

[CRC Cards: 3](#_Toc128605066)

[Class Diagram: 8](#_Toc128605067)

[Dynamic Diagrams: 9](#_Toc128605068)

[Activity Diagram: 9](#_Toc128605069)

[Sequence Diagrams: 15](#_Toc128605070)

[Use Case Diagram: 20](#_Toc128605071)

[User View Design 21](#_Toc128605072)

[Wireframes: 21](#_Toc128605073)

[#1: Login Window 21](#_Toc128605074)

[#2: Hub Area 22](#_Toc128605075)

[#3: Player Records Window 23](#_Toc128605076)

[#4: Player Skill Profiles Window 24](#_Toc128605077)

# Purpose

The purpose of this document is to give a detailed view of each diagram created for this project. These include both business models and user models.

The list of designs include, use case diagram, class diagram, activity diagrams, sequence diagrams, navigation map, CRC cards and User interface wireframes.

This document will be referenced within the main planning report that will conclude this phase of the project.

# Business view Designs

## Static Diagrams:

### CRC Cards:

CRC Cards are used to show the responsibilities and interactions of classes and other classes it collaborates with.

#1: Text

Description automatically generated

This CRC is for the class Club Member. This diagram shows that the responsibilities are to handle the logins for the two users by interacting with the login window for the application.

It’s main feature is the club member class is the parent class in our inheritance hierarchy system and is the key link to our interface called IMemberOperations. This interface provides abstraction methods which this class will then pass down to it’s children classes coach and admin.

This class is therefore the main reason we can have an object oriented system as it provides abstraction, inheritance and encapsulation.

#2: Text

Description automatically generated

This CRC card is for the class called Admin. This class is responsible for handling all behaviours related to the admin user.

This class is the child of the class club member so by default it inherits it’s abstract methods from the interface. This allows us to provide personal implementation for these methods that suits the admins needs.

This is extremely useful because our other user coach has different permission levels so it’s important we have different functionality for both users but without duplicating code. This is why abstraction is used to only take what the admin will need and discard the rest.

#3: Text

Description automatically generated

This CRC card is for the class called Coaches. This class is similar to the admin class because it’s also the child of the club member class meaning it also inherits Club Member’s interface.

This means coaches also has it’s own implementation of these methods unique to it’s own needs. This happens because the coach has certain features compared to the admin so it’s important the coach doesn’t have the same methods.

#4: Text

Description automatically generated with medium confidence

This CRC card is for the class that belongs to the window login. This class handles the events that happen when the user interacts with this window.

This main purpose of this class is to handle when the user presses the button to submit the login they just entered. Once this is pressed this class saves the login details then passes it over to the class club member.

Club member then does a check to see if the entered login matches a coach ID or an admin ID or neither. Whatever the answer club member will return an answer then login window class will decide if the user should be sent forward or not. It will also decide if the user is an admin or a coach.

#5: Graphical user interface, text, application

Description automatically generated

This class is one of our hub windows. This class handles the navigation for the current admin. The admin has two choices to either press the button to enter the player skill profiles window or the player records window. Whatever button is pressed will result in this class calling the correct window and sending the user there.

#6: Graphical user interface, text

Description automatically generated

This is our second hub class called Coach Hub. This is the area the user is taken if their ID matches one of the club’s coaches.

The admin and the coach have separate hubs because when they select one of the buttons to access either the player record window or the player skill profile window what is displayed to the user will be different to the other roles screen.

This is due to the admin having access to every team data and coaches only have access to their own teams data.

#7: Text

Description automatically generated

This CRC card is for the window class that will be used to control the player skill profiles. This achieves this by using a set of event handlers used as buttons. Once these buttons are pressed the class will call the appropriate user class to provide functionality to these buttons.

#8: Text

Description automatically generated

This final CRC card is for the window class that handles the player records. Similar to the skill profile window, this window has a set of event handlers that then triggers the correct call to provide functionality. It does this by communicating with the correct user class.

### Timeline Description automatically generatedClass Diagram:

T

This is our class diagram for the system. This shows each class of the system and what purpose they have and their relationships.

The main thing this diagram shows is our use of inheritance within the system and the use of an interface. This allows us to provide the same methods to many classes but with their own implementation. This is the use of abstraction which stops duplicating code because each class will take what they need from the interface then discard the rest.

This allows us to provide implementation depending on what user is using the behaviour. This is because our admin user and coach user have different needs while using the system. Abstraction allows us to provide separate implementation for the same method.

Abstract methods can be identified by being in italics style while the override methods are base style.

## Dynamic Diagrams:

### Activity Diagram:

#1: Diagram, timeline

Description automatically generated

This activity diagram is for the use case that handles the user login for the admin user. This diagram shows the flow of activities this user will make in this scenario. You can also see the decisions the admin makes for example what database they wish to access or what functionality they wish to use while on the database.

You can also see a branch into different diagrams called EditPlayerInfo and AddSearchAndDelete. This is done to show the full session in detail but dividing the activities into separate diagrams.

#2: Diagram

Description automatically generated

This second diagram continues from our first diagram with the path that happens if the admin wishes to edit the player records kept on the system.

Because we have multiple teams at senior and junior level the admin will have to choose what table of records they are going to edit. The reason we have two different paths is because the two levels don’t match in field types because the junior team players need parental consent to play for the club so this needs to be recorded.

#3:

Diagram

Description automatically generated

This third diagram shows the final path that the admin user could have taken. This shows what happens when the admin uses the search, add player or delete player buttons.

Search and delete follow the same path because the admin will have to search for a player first using the players given SRU number as this will be our key for our database tables. The admin will then choose if the player is found to either edit the player record or delete it.

The alternative path shows the process of adding a new record to the database.

#4: Diagram

Description automatically generated

This activity diagram shows the process for the coach user. Very similar to the admin user but in this case, there are many different coaches compared to one admin.

Like the admin activity diagrams this diagram branches off into two different diagrams showing the process for editing the player skill profiles and viewing their teams’ records.

#5:

Diagram, timeline

Description automatically generated

This diagram shows what happens when the coach user accesses the player record window. Unlike admin who can view every team, the coach only has access to their own teams’ records. This is due to GDPR being in affect so it’s important one coach doesn’t have access to every record.

The flow of activities shows the coach using the search feature to find a specific player in their squad using their SRU number.

#6: Diagram

Description automatically generated

This final activity diagram shows the process of the coach user editing the player skill profiles. Like records, the coach can only access their team’s skill profiles. It shows that that the editing process is done on a player-to-player basis so the coach will have to search for the player ID before making changes.

The coach also has the option to delete the skill profile.

The coach also has the option to add a new skill profile for their team.

### Sequence Diagrams:

#1: Graphical user interface

Description automatically generated with medium confidence

This sequence diagram shows the flow between objects that are used during the sequence of the admin logging into the system.

The sequences will always start from the login window object as this is the home screen of our app. This object sends a signal to the object clubMember when the button called btnSubmit is pressed. When pressed loginWindow sends the signal to the clubMmeber object window and also sends over two strings that contain the login.

clubMember then does a check to see if the entered details passed from loginWindow match a stored login for the admin. If this is true clubMember sends back a true boolean value which allows the loginWindow to send the admin to the Admin Hub.

#2: Graphical user interface

Description automatically generated

This sequence diagram follows the same sequence as the admin login sequence except it interacts with the coachHub object instead of the adminHub object.

Because there a many coaches the check happens with using a 1D array containing the login details for each coach of the club. This is also used to decide what should display for the current coach when accessing the player skill profile and player record windows as it will only display the coach’s team not every team.

#3: Graphical user interface, timeline

Description automatically generated

This sequence diagrams show the sequence of a coach accessing the player records for their team.

The start of the sequence follows the same route as the login taking the coach to the coachHub object. Once there the coach has two options, press the button that takes the coach to the player record database or to the player skill profiles.

This sequence shows the example of the coach pressing to access the records. Unlike the admin, the coach cannot edit the player records so their only option is to search for specific records.

This sequence starts by the coach pressing the button btnSearchRecord after inputting the SRU number for the player. Once this happens the object playerRecords sends over a string containing the inputted ID for the player to the object coachClass.

This then triggers this object to perform the method SearchPlayerRecord which performs an if statement check to see if the inputted ID matches a stored ID on the database. If true the objects send back a true statement to the record object and then the searched user is displayed.

#4: Graphical user interface, timeline

Description automatically generated

This sequence shows the final coach sequence of them updating the skill profiles for the club.

This is the alternative sequence to the coach pressing the skill profiles button on the coach hub.

Once here the coach has 4 buttons they can use. These are search, add, edit and delete.

btnSearchProfile is the button that sends the signal to the coachClass object to perform a search which sends over the search id as a string. It then performs an if statement check to see if the ID matches. If true it returns the answer to the profile window which then displays the searched record.

btnEditProfile allows the coach to edit the current searched record by interacting with the coachClass method EditSkillProfiles.

btnAddProfile does the same by interacting with the coach object.

Finally btnDeleteProfile interacts with the coach object and removes the current searched player profile completely.

#5: Graphical user interface, timeline

Description automatically generated

This final sequence diagram shows the flow of sequences for the use case when the admin edits the player records database.

Similar to editing the skill profiles the admin has 4 buttons to press, search, add, delete, edit.

These sequences are triggered by the records object and send a signal to the admin object to perform these tasks.

### Use Case Diagram:

Diagram

Description automatically generated

This is the use case model for our system. It depicts all our use cases into one single diagram showing the interaction with the actor and the activity.

It clearly shows the admin privileges and coach privileges. An example of this is the coach only wants to view their own teams personal records and skill profiles whereas an admin wants to have access to every record and profile.

The external actor called database will be our entity framework that stores our data.

# User View Design

## Wireframes:

### #1: Login Window

Diagram

Description automatically generated with medium confidence

### #2: Hub Area

Diagram

Description automatically generated

### #3: Player Records Window

Table

Description automatically generated with low confidence

### #4: Player Skill Profiles Window

Graphical user interface, diagram, application

Description automatically generated