Fantasy League: AU Edition

Final Report

Submitted for the BSc in
Computer Science with a Year in Industry

May 2016

by

Charles Stanley Alan Molyneux

Word Count: 15669

Table of Contents

1 Introduction

Project Context
The ‘Fantasy League: AU Edition’ (FAULs) application was initially requested by members of the Hull University Men’s Hockey Club (HUMHC).

The request was made by the acting Club President and Fundraising Secretary who sought the development of a mobile application which could be used to replace their existing solution for tracking hockey player statistics and managing their ‘Fantasy Hockey’ fundraising event. The solution currently in place by HUMHC utilises Excel for tracking individual club members playing statistics, as well as for storing club members individual ‘Fantasy Teams’.

Fantasy Sports
Fantasy ‘Hockey’ follows the same design as many other Fantasy Sporting applications. The ‘Fantasy League’ principle of this application is likened to popular and highly used game (or gambling) applications. The best example used in the United Kingdom would be the Barclay’s Premier League Fantasy Football application. This application allows users to select players across Premier League teams to form a ‘Fantasy’ team. The exact workings of a ‘Fantasy Leagues’ application will be explored in more depth within this report. It can however be defined at a very high level as the following, “owners either select their players or are randomly assigned players. During a sport’s season, points are generated for each of the participants’ “teams” based on real-world performances of the owners’” (*Farquhuar and Meeds, 2007*).

The Fantasy Sports industry has been valued as being worth billions, with some estimates valuing the American Fantasy Football market alone at $70bn (*Forbes, 2013*). In many cases, Fantasy Sports can be played for free with friendly leagues between friends and co-workers. However, many play Fantasy Sports more competitively by betting money on their own fantasy teams. This application hopes to deliver a facility allowing University Athletic Union teams the power they need to create their own Fantasy Sporting leagues and events and to be used as a fundraising platform for these University teams.

Projects Brief
The FAULs application aims to not only replace HUMHC’s existing solution but to provide the capability for any registered user to create their own Fantasy League. League administrators will control the content of a club page for displaying information relevant to the club, be able to create and manage fantasy players who can be a reflection of real-world players of the Users University team. As this project has been designed and developed with a generic template in mind in an attempt to extend the applications reach, the application should facilitate administrators to add in their personalised performance statistics for tracking and scoring purposes. The initial aims will be to support any teams consisting of 11 players and with the ability to track up to 11 playing statistics.

Report Structure
This report will outline the required aims and objectives which are considered necessary for the successful completion and delivery of this application. In addition to this, information and background areas considered pertinent to the topic area will be explored with in-depth analysis including Mobile Platforms, Cloud Services and existing solutions. Furthermore, the report will explore the technical development of this project including an exploration into the design, implementation and testing that were essential for successful delivery. The Designs section will give the reader an indication as to how the application intends to work at both user and system level.

The report will conclude with an evaluation of the final deliverable against the Aims and Objectives that had been put forward in Section 2. These objectives are what were considered as ‘required’ for a successful delivery. The evaluation section will consider and evaluate each Objective individually and explain any possible shortcomings. Following this section there will be an investigation into any further required functionality or possible fixes. A look into possible future development will also be examined and considered.

2 Aim and Objectives

To develop and deliver a mobile application which enables sporting clubs to create and manage a 'Fantasy League' comprised of players from their club. The application will provide a homepage to display useful club information.

The project aim stated above can only be achieved by meeting a set of deliverable objectives, these core objectives are listed below:

Objective 1 – Develop 'Fantasy Leagues' Front End

Users should be able to login, create, join and manage a fantasy league. The application should be responsive and provide a good experience.

Users should be able to view fantasy players and create teams from the pool of available players. Administrators will be able to add and manage players associated to their leagues (updating player statistics).

This objective forms the bulk of the project aim. Achieving this will be a priority in order to deliver the core application functionality.

Sub Objective 1.1 – Login and Register Users functionality

Users will be able to login and create an account through the applications interface.

Sub Objective 1.2 – Join and Creating a League functionality

Users will be able to create a 'fantasy league' for their club. Fantasy leagues will have an associated club page which includes information relevant to their club.

'Fantasy leagues' can have 'X' customisable league specific rules associated to them.

Each rule will have a name and point value associated to them. For example, a fantasy league may track a player's [Red Card] count. Each [Red Card] may be valued at [-10] points for each player occurrence.

Sub Objective 1.3 – Manage league and User League participation functionality

The league administrators will be able to create players within their league. Players will have 3 mandatory fields (i.e. player name, position/description, player value) associated to them. When a player is created within a league they will have any custom rules for that league automatically appended to their profile.

Users will be able to create a team from the pool of players available by spending their League Budget. Leagues administrators will determine the budget level for Users globally. The user may create a team with 'X' players within budget.

Objective 2 – Develop the 'Fantasy Leagues' App Service.

The aim of this objective is to create a back-end database that will allow for future development into other mobile platforms, and potentially a web application.

The database should be structured in a logical way and so that it facilitates the customisability of the application with a large number of generic data fields.

Objective 3 – Implement an enhanced 'Club Page'.

To differentiate this application from existing products, administrators will have the ability to manage and customise their club homepage content.

This objective contributes to the aim of delivering an application that improves the existing solution by providing users with a better interface and experience.

Sub Objective 3.1 – Manage Fixtures
Administrators will be able to update club fixtures. Additionally, users can view upcoming games, previous fixtures, results and game information.

Sub Objective 3.2 – Manage Club Information
There will be a number of customisable text fields to enable administrators to display information relevant to the club. For example; contact details, training times, training locations and sponsor information.

Sub Objective 3.3 – Club Page Colour Customisation
Administrators will be able to customise the club pages' background colour to match their club's colours.

Objective 4 – Implement functionality to merge user accounts with fantasy players.

Users will be able to merge their account with their 'fantasy' player counterpart. This increase the appeal of opening the application to compare their real-world performance against other team members. As HUMHC members enjoy comparing performance this feature will benefit the 'Fantasy Leagues: AU Edition' aim of providing a better solution.

Sub Objective 4.1 – Profile Merging
Users can merge their login account with their respective fantasy player counterpart.

Sub Objective 4.2 – View Profiles
Players can view other users within the same club and can compare their player statistics.

Bonus Objective 5 – Design and develop a basic web application to input league data.

Whilst not required to meet the overall aim, future development could include a web interface for administrators to manage league Meta data. This basic interface could allow administrators to manage their leagues players, rules and club information. This objective remains relevant to the project aim by enhancing the overall user experience but is not necessary to achieve the overall applications functionality

3 Background

3.1 Problem Context

This project proposal has been formed based on the clients (Hull University Men's Hockey Club – HUMHC) requirements. The club president has requested that their method of tracking player statistics and club member's fantasy teams be digitalised into a mobile application.

This section will clarify the objectives of the client, the purpose and meaning of 'Fantasy Leagues', existing products which may or may not already deliver on these objectives of this project and finally the various technologies that have been considered for the development and implementation of this proposal.

3.1.1 Client

As mentioned above the client for this project is the HUMHC team. They had provided a basic specification which simply requested for the existing Fantasy Hockey / Player Tracker method into a Native Mobile Application. This project will aim to deliver an initial launch for one of the major platforms, with development for the remaining platforms to be completed at a later date.

During the clubs bid to increase club funds, the fundraising secretary decided to use the playing statistics of the club's team members for the clubs very own 'Fantasy Sport' fundraiser. Participants had to contribute £5 into a prize pool, the member with the most points at the seasons end took 50% of the pot with the remaining 50% going to the club.

3.1.2 Fantasy Leagues

The idea behind 'Fantasy Team' stems from a widely used and popular gaming application, which is known as the Barclay's League Fantasy Football, "A competition in which participants select imaginary teams from among the players in a league and score points according to the actual performance of their players." (*Oxford Dictionaries, 2016*).
The concept as explained above it a simple one, people can register an account and select players from various football premier league clubs. Each user has a pre-set budget which allows them to purchase players. Player values vary depending on their performance throughout the season which forces to the user to select a team with a range of abilities. At the end of each week each player's performance is added up (goals scored, appearances) and their points are totaled.

What this application hopes to achieve is a facility which allows any sporting club to create their own Fantasy League. Users of the application will be able to create a 'Club Page' populated with customisable text fields and populate their 'Club Page' with 'Fantasy Players'. These Fantasy Players will be associated to real-life players and members of that particular club. The application will be written generically so any club / society can pick it up and introduce their own custom rule sets.

3.1.3 University Sport Market Share

With the application now being designed and implemented with the notion of being used as a 'generic' template for any university team, it seemed appropriate to investigate the potential market for any potential monetisation of the product.
The British Universities and College Sport (BUCS) organisation is the governing body for University Sport inside the United Kingdom. At present, there are 50 Sports included in BUCS with 170 member institutions (BUCS, 2016). Currently, these 170 member institutions account for 4800 teams and over 100 annual Championships being held.

The University of Hull itself has 50 Clubs and around 2500 members of the Athletic Union (*Hull Student, 2016*). From this research it can be concluded that the potential market for a University specific 'fantasy sports' application is quite promising.

3.2 Comparison of Technologies

Currently there are two major mobile platforms contenders - Android and iOS which together have a 96.7% worldwide market share for smartphones (IDC, 2015). Windows Phone, BlackBerry OS and 'Other' operating systems divide the remaining 3.3% between them. Figure 2.5 provides the breakdown of the operating systems market share, showing Android with the largest portion. This section considers the positives and negatives of Android, iOS, Windows, Cross Platform Development (CPD) and Web Applications.

Period	Android	iOS	Windows Phone	BlackBerry OS	Others
2015Q2	82.8%	13.9%	2.6%	0.3%	0.4%
2014Q2	84.8%	11.6%	2.5%	0.5%	0.7%
2013Q2	79.8%	12.9%	3.4%	2.8%	1.2%
2012Q2	69.3%	16.6%	3.1%	4.9%	6.1%

Figure 3.1 - Worldwide Smartphone OS Market Share (Share in Unit Shipments)
2012 Q2 – 2015 Q2
(International Data Corporation (IDC), Aug 2015)

3.2.1 Android

Android is the leading mobile operating system, see Figure 3.1. Android has maintained its lead over its competitors for the past 4 years despite a 2% decrease from the previous year. The main benefit of Android is the platforms popularity. By not choosing Android 82.8% of the potential market would not be reached. Furthermore, publishing applications to Googles Play store is easy with a less restrictive application policy. This also means Android is an ideal testing ground for beginner developers. There are some negatives in choosing Android. Android devices are fragmented, varying in size, specification and versions (see Figure 3.2). Developers having to decide between functionality and/or potential users and involves additional testing to ensure maximum device compatibility. This project would look to target version 4.1.x upwards, which would support 92.6% of Android devices.

Version	Codename	API	Distribution
2.2	Froyo	8	0.2%
2.3.3 - 2.3.7	Gingerbread	10	3.8%

4.0.3 - 4.0.4	Ice Cream Sandwich	15	3.4%
4.1.x	Jelly Bean	16	11.4%
4.2.x		17	14.5%
4.3		18	4.3%
4.4	KitKat	19	38.9%
5.0	Lollipop	21	15.6%
5.1		22	7.9%

Figure 3.2 - Android Platform Versions
(Data collected during a 7-day period ending October 5, 2015)

3.2.2 iOS
iOS is the second most widely adopted platform and saw a 2% rise in its market share from 2014 into 2015. Apple products carry a premium price tag, a possible reason for their smaller global market share. Noticeably, in wealthier countries the market share is more evenly distributed. Some sources identify iPhones accounting for 42.5% of British sales in October 2014 (Williams, 2015).

The main draw towards iOS development is the relative ease. iOS devices do not suffer from the same 'hardware' fragmentation as Android devices. Additionally, development could be extended to the popular iPad tablets. The main drawbacks to the iOS platform is the 'closed' eco-system. Development requires specific tools, such as a machine running OSX and xCode. Furthermore, Apple are stricter on publishing applications onto their store with each release requiring approval thus adding a layer of complexity to publishing.

3.2.3 Windows Phone
Microsoft have had a difficult time trying to increase their market share with their windows mobile devices. With still a relatively small share of the smartphone market not many developers are willing to invest time into porting their applications onto the windows platform.

The main draw towards the Windows platform is the developer's familiarity with the language and tools used (C# and Visual Studio). However, due to the small share of the market (2.6%) the platform will not be chosen or considered for initial launch.

3.2.4 Xamarim
Xamarin is a CPD tool that allows C# code to be used across the major mobile platforms (Android, iOS and Windows) and therefore appeals to a large number of developers. "Use the same language, APIs and data structures to share an average of 75% of app code across all mobile development platforms." (*Xamarin, 2016*). Full annual membership costs $1899. However, a free version is available for students.

The main disadvantage of using Xamarim is the developer's dependence on Xamarin providing support and with a smaller developer community there is less documentation available. Additionally, developers become dependent on Xamarin providing updates to support new mobile versions.

Xamarin Forms allows for the UI to be shared across platforms. This is best when “code sharing is more important than custom UI” (*Xamarin, 2016*). However, if requiring platform-specific APIs or custom UI objects then native app code still needs to be written. This increased the work required to deliver application with the same responsiveness as a native application.

3.2.5 Web Applications

“The Web technology stack has not achieved the level of performance we can attain with native code, but it’s getting close” (Charland & Leroux, 2011).

When designed correctly Web Applications work across all devices, significantly reducing cost and time required to deliver an application. There a number of reasons why ‘Fantasy Leagues’ will not be a Web Application. Firstly, the initial request from HUMHC was for a native mobile application. Secondly, native applications are considered more responsive, faster and overall a smoother User experience. As User experience is an important element of this project the benefits of native development supersede the ease of ‘cross platform’ web application design.
It would be expected for users to compare teams at the pitch side - using smartphones. Launching an application is a quicker process than navigating to a website, with a potentially intermittent connection.

3.2.6 Conclusion

This project will be developed for Android using Android Studio. This choice has been made based on Androids dominance in the smartphone market. Despite iOS sales having risen in the UK there’s no certainty whether this trend will continue. Additional consideration has been the tools required to undertake iOS development. There would be a significant overhead cost to purchase a Mac to run xCode as well as an iPhone for complete testing.

Android development is more accessible, using Windows or OSX and devices can cost significantly less. Xamarin will not be used due to the reliance on Xamarin providing updates, a smaller developer community and most importantly the limitations of sharing UI code. Tailoring the UI code would require extensive resources to ensure the best user experience is achieved. Finally, the decision against a web application was due to the reduced responsiveness and performance on mobile devices. The initial request had been for a native mobile application, which also played a role in the decision. Whilst initial release will focus on Android, future development wouldn’t discount a dedicated web application for ‘Fantasy Leagues’.

3.3 Mobile Services and Cloud Storage

A primary requirement of this project is a method of retrieving and storing data on the cloud. A number of solutions exist that would satisfy this requirement. The section below explores pros and cons of some of these solutions that have been considered as part of this project.

3.3.1 Microsoft Azure

Azure, launched in February 2010, offers an extensive number of features and services which would meet the aims and objectives of this project. *“Azure is a growing collection of integrated cloud services – analytics, computing, database, mobile, networking, storage and web – for moving faster, achieving more and saving money.”* (Microsoft Azure, 2016).

Azure offers a reasonably priced service which can host a SQL Database on a SQL Server and an App Service to allow the application to communicate with the SQL Server.

Azure is a PaaS (Platform as a Service), this allows the developer to essentially outsource the expensive and time consuming requirements of having to build and maintain an applications infrastructure.

The Azure platform has a range of features which includes User authentication through Facebook, Google, Twitter or Azures Active Directory. Additionally, Azure scales traffic automatically and supports the implementation of Push notifications. Azure supports all three major mobile platforms (Android, iOS and Windows) and the service integrates easily with web applications, allowing for the bonus objective of this project to be met (or for future development purposes).

3.3.2 Google Cloud

The Google Cloud platform was launched in May 2010 and intended to become a major competitor to Amazons S3 Service (Simple Storage Service) (Techcrunch, 2010). Since its release it has become one of the leading platforms for offering mobile and cloud services with a large number of leading companies using this service to deliver powerful applications. Companies include Ubisoft, Best Buy and Sony Music (*Google Cloud, n.d.*).

One of the products which the Google Cloud platforms provides is the Google App Engine.

"Google App Engine is a platform for building scalable web applications and mobile backends. App Engine provides you with built-in services and APIs such as NoSQL datastores, memcache, and a user authentication API" (Google App Engine, n.d.)

As mentioned above, the Google App Engine offers a range of services which are similar to those offered by the Azure platform. Services include Cloud Data Storage (NoSQL), User Authentication and automatic app scaling. The Google App Engine integrates easily with both Eclipse and Android Studio. In addition, the App Engine has native libraries for both Android and iOS (but not Windows). As a result of these features, this service would be able to deliver on this projects aims and objectives. Although, it would limit any possible future development into the Windows platform.

3.3.3 Parse

Parse, launched in 2012, and acquired by Facebook in April 2013, (Facebook Developers, 2013) is a back-end service which offers both cloud storage and a mobile service with a list of features similar to Azure and Googles App Engine.

The Parse SDK (a native SDK for each mobile platform) allows for the storage and retrieval of 'Parse Objects'. Features offered by the Parse service include Push Notifications, Cloud Storage, Automatic App Scaling and User Authentication.

Parse supports all major mobile platforms, even with the addition of Xamarim. Integration into a Web Application is documented as a relatively simple process. Parse is well documented for each of the Mobile Platforms and has a moderately sized community to offer ad-hoc support if required.

The main selling point of Parse is the ease of setup and free pricing tier. However, through experimentation with the service and exploring developer forums, many using the service have claimed it is only truly viable for small and/or application prototyping. More complex tasks can be quite difficult to implement.

During the initial project investigation, time was spent using Parse with a prototype application being built using the Parse SDK. The installation was quick and the basics were easily implemented. There were some difficulties joining tables together as the concept of a ‘Foreign Key’ had been replaced with ‘Relations’ and ‘Pointers’ to other tables. Overall the Parse service appears ideal for rapid prototyping but somewhat unwieldy at implementing more complex features.

3.3.4 Conclusion

For this project, the Microsoft Azure platform has been chosen for hosting the mobiles app service and the SQL Database. All three services mentioned above offer a similar set of features, a comparative amount of documentation and each one has a well-established community of developers to provide support should it be required. Due to these similarities the decision came down to existing expertise and the developer’s confidence in using the investigated platforms.

As a result of the developer’s familiarity with the Azure platform it will reduce the required time to become familiar with an entirely new, and foreign platform. There is also the confidence that Azure will be able to deliver on this projects requirements and therefore selecting another platform will unnecessarily increase the risk of meeting deadlines.

3.4 Alternative Solutions

The application can be seen to be doing two tasks. Firstly, providing a facility which allows users to create a ‘Fantasy League’. Populate that ‘Fantasy League’ with Players (These players are expected to reflect real people) and update points, club information and teams on a regular basis. Another, possible use of the application could be for team selection and player statistic tracking. Club captains could actually use this application to monitor individual player performance on historic data. Because of this, the solutions explored in this selection extend past the ‘Fantasy’ aspect of the application and look at player performance tracking and whether such an application exists.

This section explores a selection of these existing applications and evaluates the key features offered by each service and their strengths and weaknesses when compared to this proposal.

3.4.1 Fantastar

“Fantastar is a feature rich, mobile-optimised fantasy sports platform for creating branded fantasy sports games.” (Fantastar, 2005)

Fantastar offers a wide range of features which include team selection, leader boards, customisable rules and real time player statistics. Fantastar also offers a level of customisation which this project hopes to match, including customising league pages with organisational branding and the personalisation of league rules.

Fantastar’s primary market is aimed at higher levelled-tier sports and uses a 3rd party sports data specialist to automatically update player statistics. Even so, the Fantastar website says a service is available for setting up custom ‘Fantasy’ leagues for schools, similar to the aim of this project. Fantastar has managed to attract some high-profile companies, with clients such as McDonalds and O2, making it a worthwhile case study.

Evaluation
Fantastar’s primary drawback is the reliance on Fantastar creating, updating and maintaining the personalised league. Whereas the application proposed by this project aims to put full control into the hands of the league creators including updating players, customising fields

and editing league rules. In comparison to this proposal Fantastar builds a fixed application from a specification, and they manage the rest.

Fantastar does offer a comprehensive service with a wide range of exciting features which include social media integration, real time statistics tracking and a very responsive ‘team selection’ page. The pricing for this service is all speculative. To receive a quotation, they require a specification emailed to them of the league requirements.

3.4.2 Fantasizr

A simplistic web application which allows the creation of Fantasy Leagues for anything e.g. Sports, TV programs. Fantasizr allows league creators to manage scores, players and rules.

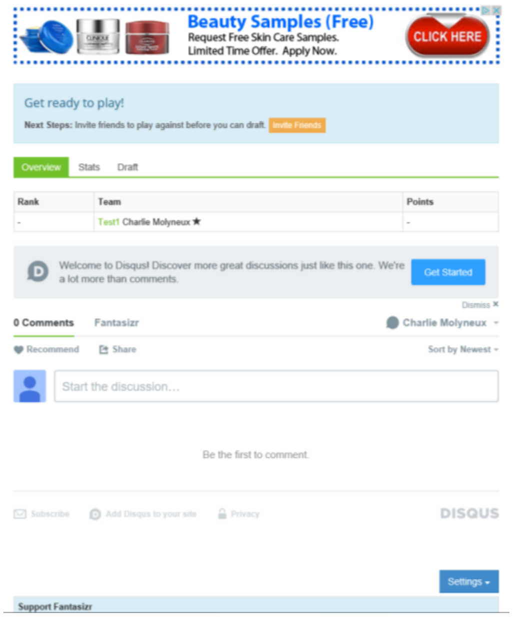
This solution puts the control into the hands of the users and allows them the ability to ‘Draft’ (add players), add and track stats and compare league member’s stats on a league table.

Evaluation

The level of customisation offered by this service is something this project hopes to achieve. However, there are a number of limitations which include poor user interface design and an expensive pricing structure.

Firstly, looking at the design and layout of the application as seen in Figure 3.3 the [Settings] button (used for league management, updating players, rules, etc.) is located on the bottom right of the screen. This button appears to be outside the applications page. It would make more sense to place it on the navigation pane. Additionally, the homepage is cluttered with irrelevant information and banner ads. Another limitation is users are restricted to tracking 3 Stats (illustrated in Figure 3.4), potentially putting off those who require more In-depth statistics.

The true limitations of the service were shown when investigating the pricing model for this service. Even at the most expensive tier ($49 upwards) the user is limited to tracking 5 Statistics. In addition, the season length can only be extended up to 12 weeks (a Hockey season lasts nearer to 22). The main drawback here is the limitation of statistics per league (rules). For a sport such as Hockey you would want as in-depth statistic tracking as possible, including Goals, Assists, Appearances, Clean Sheets, Bookings (3 Types) which is already exceeds their limit of 5.


Beauty Samples (Free)
Request Free Skin Care Samples.
Limited Time Offer. Apply Now.
CLICK HERE

Get ready to play!

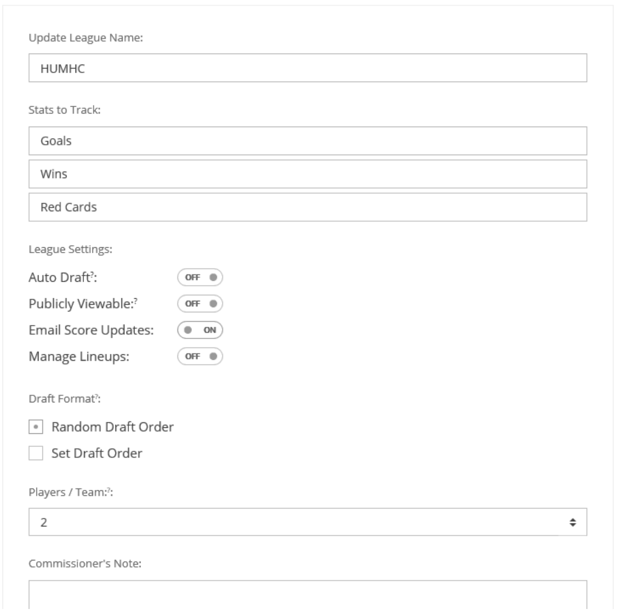
Next Steps: Invite friends to play against before you can draft. Invite Friends

Rank	Team	Points
-	Test1 Charlie Molyneux ★	-

Welcome to Disqus! Discover more great discussions just like this one. We're a lot more than comments.

Be the first to comment.

Figure 3.3 –
Fantasizr Custom League Homescreen,
(Fantasizr, 2015)


Update League Name: HUMHC

Stats to Track: Goals, Wins, Red Cards

League Settings:
Auto Draft?: OFF
Publicly Viewable:?: OFF
Email Score Updates: ON
Manage Lineups: OFF

Draft Format?:
[✓ Random Draft Order]
Set Draft Order

Players / Team:?: 2

Commissioner's Note:

Figure 3.4 -
Fantasizr Custom Statistics,
(Fantasizir, 2015)

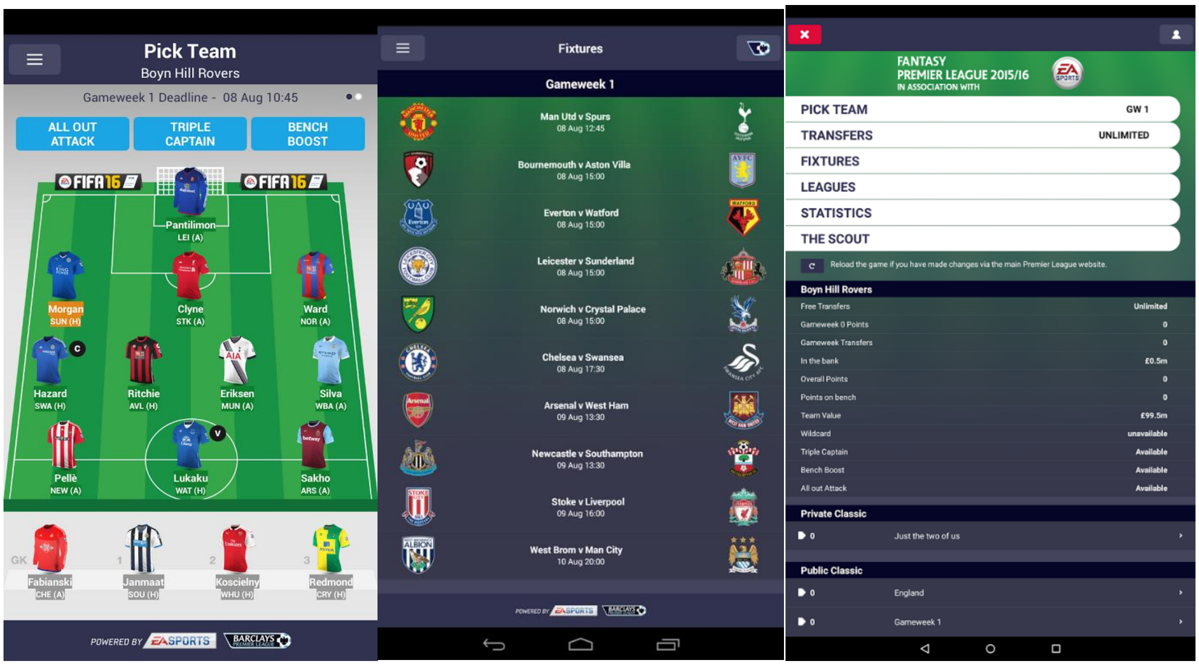
3.4.3 Fantasy Premier League Football

This application offers users a range of features including: team management, statistics, new feeds and upcoming fixtures. The application offers a service similar to what 'Fantasy Leagues' hopes to provide i.e. the management and tracking of a fantasy team. However, Fantasy Premier Leagues doesn't provide users the ability to build their own custom leagues and is focused solely on Premier league football.

Evaluation

As seen in Figure 3.5 (below) the application is well designed and through testing the application it is a fair assessment to say that it works well. However, it could be argued that for smaller screens the user may be overwhelmed with the amount of information displayed and this could be an issue for some users.

Overall the popularity of this application is easily understood. Whilst it plays on the popularity of the sport it does offer a comprehensive service and works as expected, delivering a seamless experience to the user.



Fixtures

Gameweek 1

Man Utd v Spurs
08 Aug 12:45

Bournemouth v Aston Villa
08 Aug 15:00

Everton v Watford
08 Aug 15:00

Leicester v Sunderland
08 Aug 15:00

Norwich v Crystal Palace
08 Aug 15:00

Chelsea v Swansea
08 Aug 17:30

Arsenal v West Ham
09 Aug 13:30

Newcastle v Southampton
09 Aug 13:30

Stoke v Liverpool
09 Aug 16:00

West Brom v Man City
10 Aug 20:00

FANTASY PREMIER LEAGUE 2015/16 IN ASSOCIATION WITH

PICK TEAM	GW 1
TRANSFERS	UNLIMITED
FIXTURES	
LEAGUES	
STATISTICS	
THE SCOUT	

Reload the game if you have made changes via the main Premier League website.

Boyn Hill Rovers	
Free Transfers	Unlimited
Gameweek 0 Points	0
Gameweek Transfers	0
In the bank	£0.5m
Overall Points	0
Points on bench	0
Team Value	£99.5m
Wildcard	unavailable
Triple Captain	Available
Bench Boost	Available
All out Attack	Available

Private Classic

0 Just the two of us

Public Classic

0 England

0 Gameweek 1

Figure 3.5 – Fantasy Premier League (Android Application: https://play.google.com/store/apps/details?id=com.allappedup.fpl1516 *, Oct 2015)*

4 Technical Development

4.1 System Design

The FAULs mobile application is formed from two key elements namely Azure Mobile Service (backend) and the mobile application (frontend). Both of these components are required to successfully deliver on any one of the 4 key project objectives.

The Azure platform hosts the FAULS relational database and the Mobile Service (Cloud Service) manages the communication between the mobile application and the database. In addition, the Mobile Service allows for a number of custom APIs to be created which have a number of uses such as fetching data and performance intensive calculations.

The mobile application element provides the User with an interface to communicate with the Mobile Service and FAULS database.

This section will cover how the database has been designed to facilitate the applications requirements and how the mobile application has been designed to provide a viable front end and pleasing User experience.

4.1.1 Database Design

A relational database has been designed and implemented for this project. The benefits of a relational database are that “data can be retrieved either from a particular table or from any number of related tables within the database. This enables a user to view information in an almost unlimited number of ways.” (*Hernandez,1997*). The relational model has been used as it reduces data duplication, enhances data integrity and enables complex joins to be performed across the various tables. The FAULS database is used to store all the necessary information for the application to function as expected and to deliver on the projects aims and objectives.

The overall design of the database and its corresponding tables required careful planning. Firstly, an overall idea as to what data might be required to fulfil the aims of the project had to be considered before any tables were created. Secondly,
consideration had to be given into avoiding unnecessary duplication of data through normalisation. Finally, maintaining data integrity and how best to structure the tables to facilitate the generalised nature of the application, influenced the overall database design.

Table names and columns all follow a similar naming convention to ensure consistency throughout the database. Where possible columns were assigned universal names to emphasise the generic nature to which the application seeks to achieve. An example of a generic column name can be found within the ‘fantasyPlayers’ table where a column named [RuleOcc1] is present. This column [RuleOcc1] simply states that this column relates to the first rule that occurs in the ‘leagueRules’ table. The ‘Rule’ to which this column refers could relate to anything e.g. it could be associated to either a [Goals Scored] or [Goals Saved] entry within the ‘leagueRules’ table.

The following diagrams are partial extracts from the FAULS database. They help demonstrate how the FAULS database has been designed to join the appropriate tables together, as well as to support the generic creation of a number of different sporting fantasy leagues.

Note: The full database structure can be found in Appendix D

4.1.1.1 Users and User Leagues

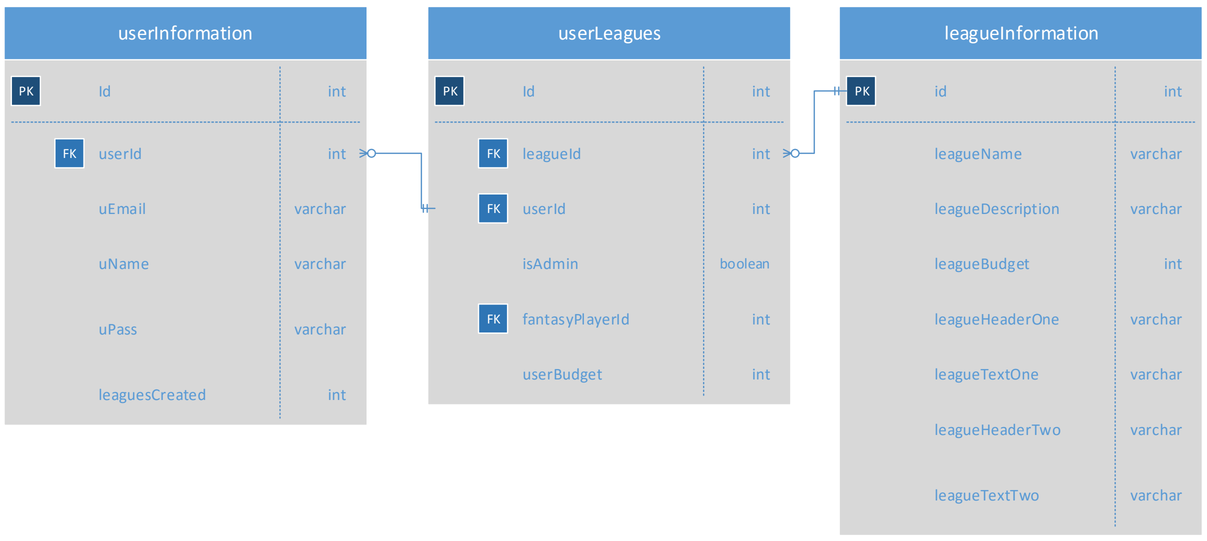


Figure 4.1.1 – User Table Design

The 3 tables above (Figure 4.1.1) form the foundation for subsequent tables. These tables have been designed to facilitate the joins between tables relating to both league information and the fantasy sport leagues functionality. When a league is created the 'leagueInformation' table is populated with the information entered by the user. Users can then join these leagues.

The association for joining a user to a league is done through the 'userLeagues' table. The relationship is a simple 'many users to many leagues'. The [isAdmin] column stored within the 'userLeagues' table helps determine who can access the applications league management side for that particular league.

4.1.1.2 League Fixtures

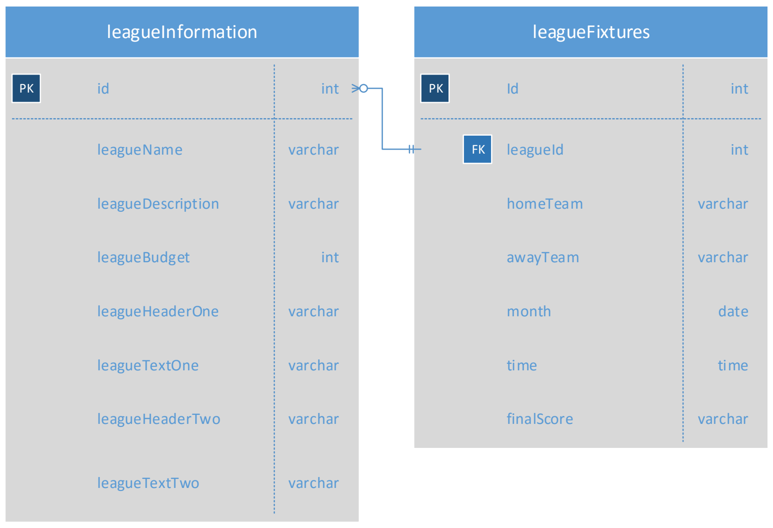


Figure 4.1.2 – League Fixtures tables

The application required the functionality to manage league fixtures as outlined in Objective 3. This was incorporated into the FAULS database by designing tables as seen above in Figure 4.1.2 which can join with one of the three main Primary Keys ([leagueId], [userId] or [fantasyPlayerId]). In this design a league can have '0 to many' 'leagueFixtures'. For initial release the design of the ‘leagueFixtures’ table has remained quite simple. However, the Database design will allow for further functionality to be implemented to include (for example) the fantasy player ‘goal scorers’.

4.1.1.3 Users and Teams

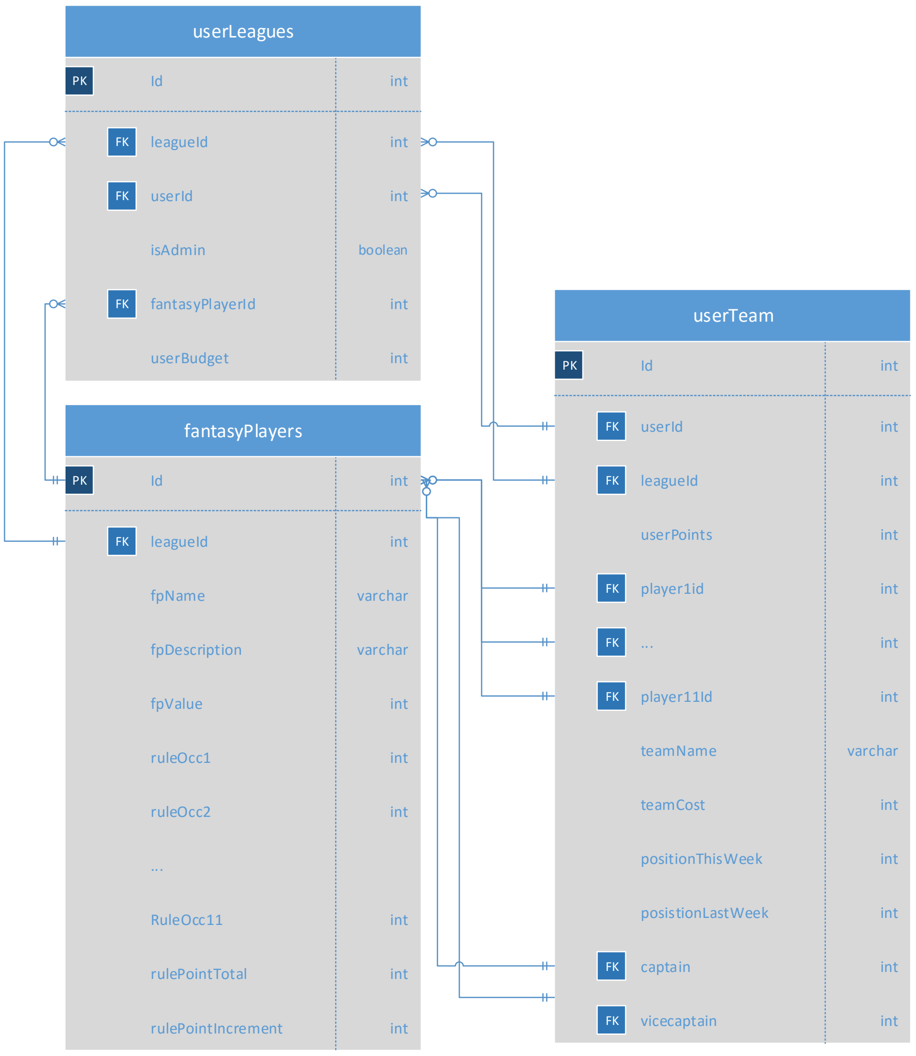


Figure 4.1.3 – User and User Team Table Design

The above diagram (Figure 4.1.3) is a simplified version of the joining between the three tables responsible for user team management. As some University students belong to numerous sporting teams the database has been designed to allow for ‘many users to have many leagues’. The interface and applications functionality has been designed to prevent the User from creating more than one team per league.

Each record within the ‘fantasyPlayer’ table is associated to a single league however a record may be present in more than one team.

The ‘userTeam’ table is comprised of a number of columns. Of these columns 11 are used to store the value of a ‘fantasyPlayer’ Id ([player1id], [player2Id] …[player11Id]). This design will enable the joining of the ‘userTeam’ and ‘fantasyPlayer’ together. In addition to this a [userTeam] record includes a [captain] and [vicecaptain] column. These columns store the Id of one of their 11 players and are used to provide point bonuses which shall be explored later within this report.

Finally, one of the projects objectives is the ability for a User to merge their account with a 'fantasyPlayer' profile. To meet this objective, the design of the database and above tables has allowed for a user within the 'userLeagues' table to store the Id of their associated 'fantasyPlayer'. The application will then be able to retrieve the appropriate player upon request.

4.1.1.4 Fantasy Player Scoring

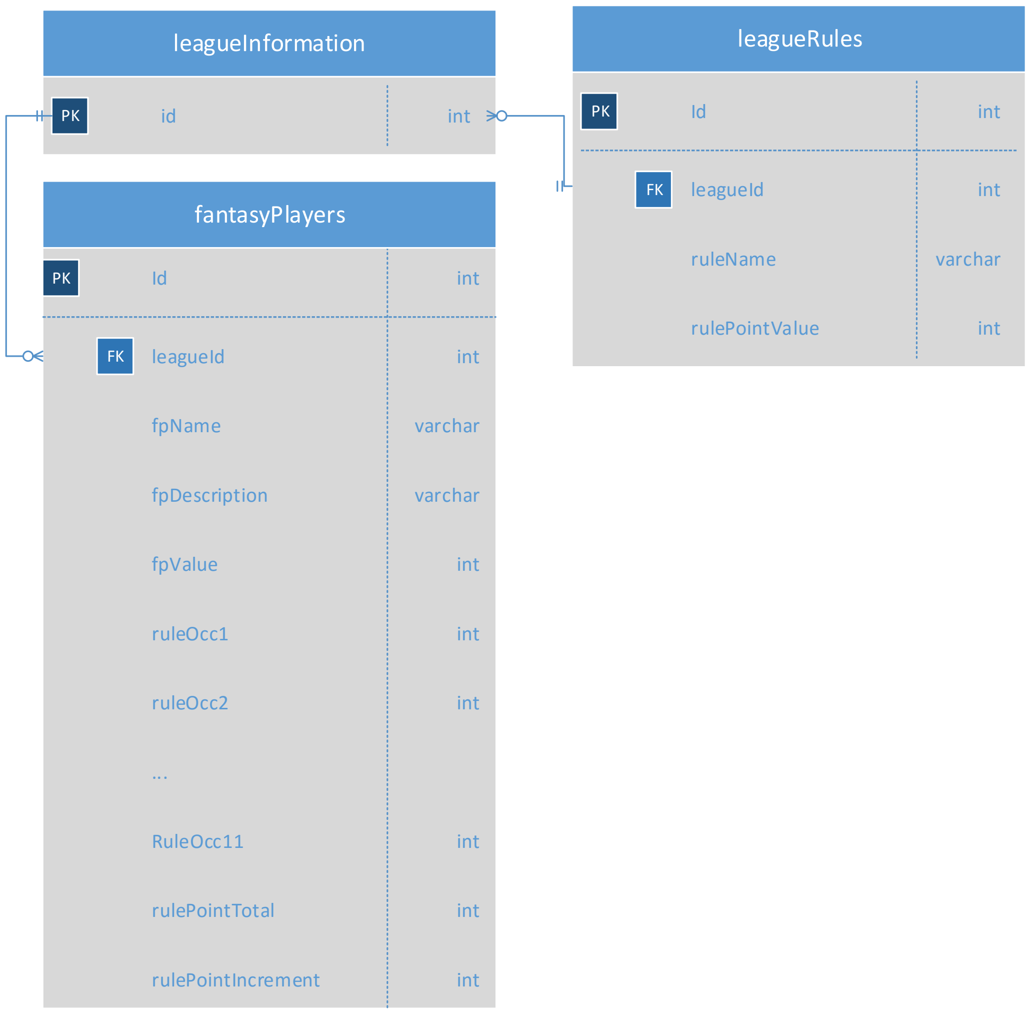


Figure 4.1.4 – Fantasy Player and Score tracking table design

An essential feature of any fantasy sporting application is the ability for players to accumulate points based off their real world performance. This application had to be designed to allow users to create and manage their own scoring system. The above tables (Figure 4.1.4) have been designed to store data relating to a player's real-world performance. However, due to the generalised nature of this application neither the database or mobile application knows the exact name, values or amount of player statistics to be tracked. The structure of these tables and the User Interface design has been influenced by the aim of proving a generic template for sporting teams to manage their fantasy leagues.

In this design the database and the tables illustrated above allow league administrators to add up to 11 league specific rules (or statistics). The rules being tracked by a particular league are stored within the 'leagueRules' table. This table is joined to both a league and 'fantasyPlayer' through the [leagueId] Foreign Key.

An example as to how the ‘rule occurrences’ of a fantasy player are matched to a particular rule is demonstrated below.

leagueRules		
leagueId	**ruleName**	**ruleValue**
1	Goals	10
1	Assists	5

fantasyPlayer						
Id	**leagueId**	**playerName**	**ruleOcc1**	**ruleOcc2**	**rulePointTotal**	**rulePointIncrement**
1	1	C.M	1	2	20	0

The mobiles user interace reads the leagueRules table in conjunction with the fantasyPlayer table with ruleOcc1 being associated to the first rule stored inside the leagueRules table. In the example above, [ruleName] ‘Goals’ would be associated to [ruleOcc1] of the fantasyPlayer and [Assists] would associate to [ruleOcc2].

When a league administrator updates the value stored inside either [ruleOcc1] (Goals) or [ruleOcc2] (Assists) for a particular player the application will multiply the [ruleValue] by the number of times the [ruleOcc(x)] has been incremented.

For example, incrementing [RuleOcc1] by an additional 2 so that [ruleOcc1] = 3 would cause the column [rulePointIncrement] to equal (10 * 2 = 20).

fantasyPlayer						
Id	**leagueId**	**playerName**	**ruleOcc1**	**ruleOcc2**	**rulePointTotal**	**rulePointIncrement**
1	1	C.M	3	2	20	20

The rulePointIncrement column will continue being added onto until a league administrator decides to calculate the points for that week. Once this happens a custom API adds the rulePointIncrement value onto the rulePointTotal value. Following this the rulePointIncrement columns value is reset to 0.

fantasyPlayer						
Id	**leagueId**	**playerName**	**ruleOcc1**	**ruleOcc2**	**rulePointTotal**	**rulePointIncrement**
1	1	C.M	3	2	40	0

The logic and implementation of how point scoring and player management functions will be explored within the System Implementation section of this report.

4.1.2 Mobile Application Design

The applications functionality and overall design had to meet the following requirements. The first requirement was to provide a frontend interface allowing Users to participate in a fantasy sporting league. The second requirement was the ability for Users to administer their own fantasy leagues if they should desire.

As a result of these two requirements (league management and user participation) the applications functionality has been designed very much into two sections with some areas of functionality overlapping.

4.1.2.1 Use Case Diagram

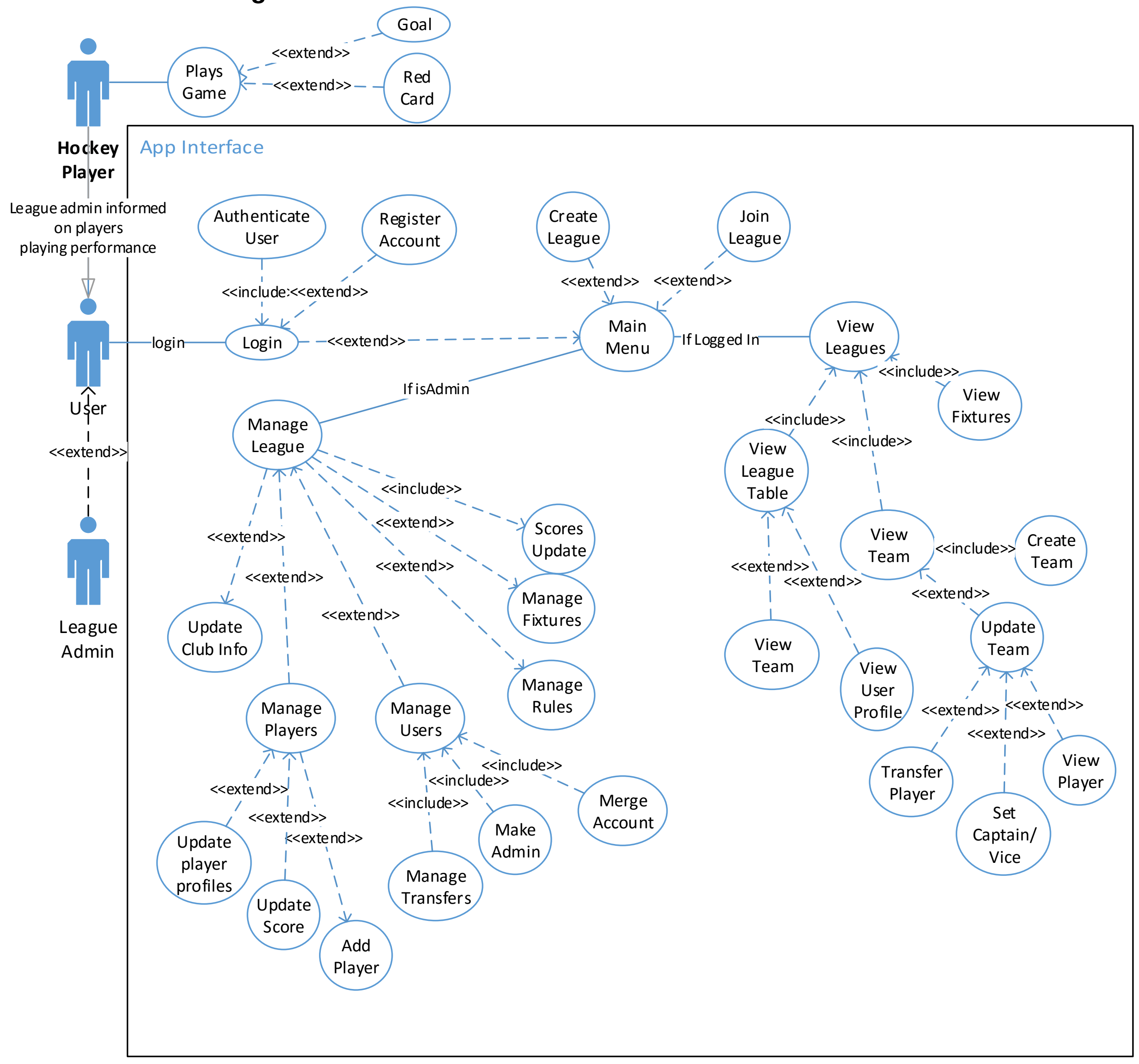


Figure 4.1.5 – Application Use Case Design

The Use Case Diagram above (Figure 4.1.5) illustrates how the application has been designed at a very high level.

The main purpose of this Use Case diagram is to illustrate that although the application is being launched as one product its functionality is very much divided in two sections. Firstly, the league and player management functionality and secondly the ‘fantasy leagues’ game itself made available through the [View Leagues] function.

In the above Use Case example, the fantasy leagues sport is Hockey. In this Use Case diagram, the above 'Hockey Player' (Actor) has played a Game. In this game the Player has had a chance to score a Goal and receive a Red card. These two attributes will represent two of the league specific 'League Rules'.

The application has been designed so that (real-world) and User participating in a fantasy league can be granted administrator privileges. Thereby enabling them to update each of their player's performance post-match (i.e. Goals scored) through the application.

The application will have a 'My Leagues' and 'Manage Leagues' sub-menu which can be accessed through the applications main menu. Users will gain access to the 'Manage Leagues' functionality through an [isAdmin] check.
League Management has three primary functions. Firstly, the Admin can manage the league (calculate scores, update descriptions, manage fixtures and manage rules). Secondly, the Admin can manage players by updating their playing statistics, name, value and position. Finally, the administrator can manage Users participating in their league. User management includes the ability to merge User profile to a fantasy player, updating player transfer limits and managing Admin status.

In comparison, a standard league User will have a more limited use of the applications functionality. A User will have the ability to create a fantasy team from the pool of players made available by the leagues administrators. The application has been designed so a standard User can ignore the Management functionality entirely and focus solely on participation. This design helps to fulfill the requirements outlined in Objective 1 by providing a functioning application to manage and participate in fantasy sports leagues.

4.1.2.2 Activity Diagrams – League Creation

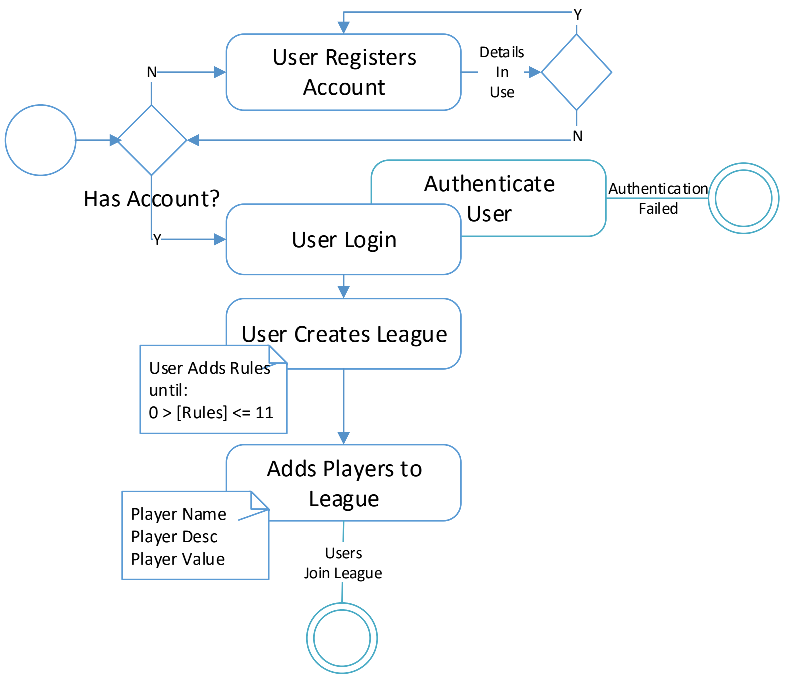


Figure 4.1.6 – Creating a League

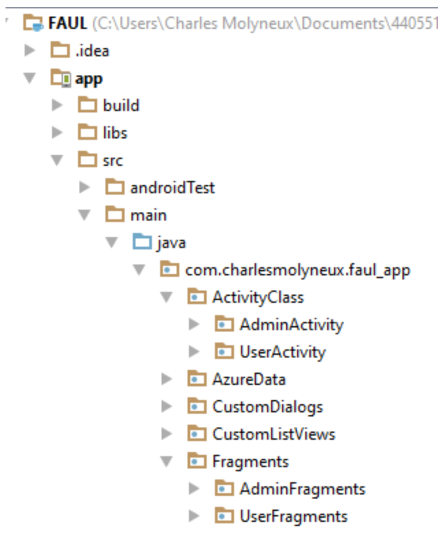
The UML Activity Diagram seen in Figure 4.1.6 helps to illustrate how the application has been designed to simplify the process of allowing any User to create a league and begin adding their leagues rules and fantasy players.

The proposed design aims to provide a system where Users can rapidly enter the required information to setup league as quickly as possible. This will make the application available to use in cases where a ‘Fantasy League’ application may be required promptly, possibly for an ad-hoc sports day event.

4.1.3 Class Design

4.1.3.1 Packages

The application has been developed with all Java classes belonging to one of five packages. Dividing classes into these packages has helped group relevant functionality together and made it easier to locate the required Java classes when required.



```
FAUL (C:\Users\Charles Molyneux\Documents\440551
  .idea
  app
    build
    libs
    src
      androidTest
      main
        java
          com.charlesmolyneux.faul_app
            ActivityClass
              AdminActivity
              UserActivity
            AzureData
            CustomDialogs
            CustomListViews
            Fragments
              AdminFragments
              UserFragments
```

Figure 4.1.7 – Application Packages

The Activity and Fragment classes each contain two further sub-directories to differentiate between the user and administrator pages and functionality.

In addition to the admin/user activities and fragment packages there are three additional packages. Firstly, the ‘AzureData’ package contains classes relevant to Azure service (data model). Secondly, custom dialog implementations with added functionality, these classes extend the ‘BaseDialog’ class. Finally, there is a package for the custom ‘ListView’ adapters. Custom ‘ListView’ adapters have been designed and implemented when the standard ‘BaseAdapter’ class fails to meet the required functionality.

4.1.3.2 Azure Classes

The Azure documentation states “The easiest way to query or modify data in the mobile service is by using the typed programming model” (*Azure, 2016*). This data model, that has been outlined in their documentation allows for “seamless serialization and deserialization to JSON using the gson library”. (*Azure, 2016*).

For every table that has been created inside the FAULS database there is a corresponding Java class inside the ‘AzureData’ package. The naming conventions used for these classes and the data members inside have been chosen to be a reflection of their Azure tables counterpart. This design choice has helped maintain consistency between the two components.

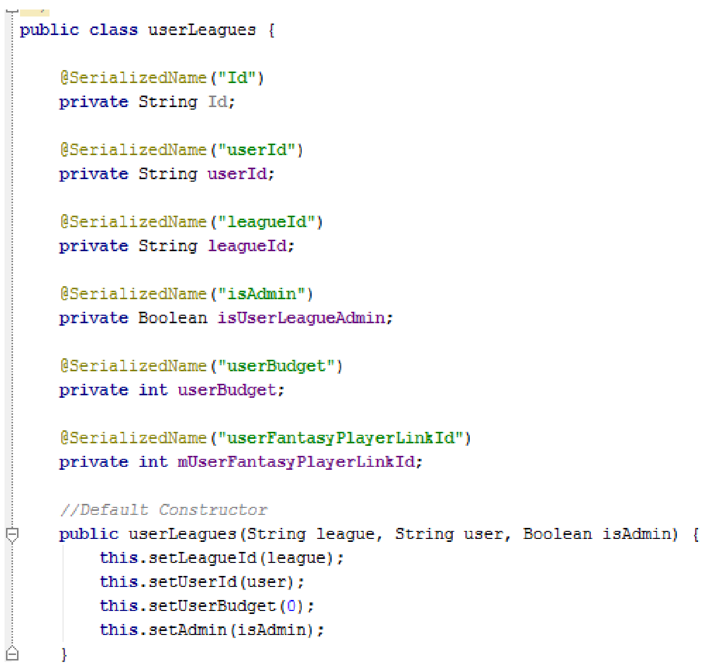
The ‘MobileServiceTable’ object is used to represent a table stored within the mobile service. The object has number of functions which support the insert, update, delete and querying of data inside a mobile service table.

As mentioned above, for the mobile application to read or insert data into a mobile service table it must first undergo serialisation (or deserialisation) into a JSON object (or vice-versa).

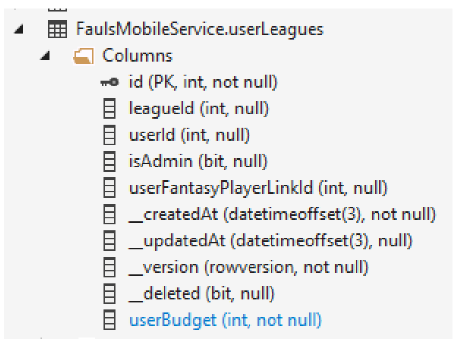
This is achieved by prefixing each data member that corresponds to a table column with the ‘*@SerializedName(“columnName”)’* annotation. This annotation indicates that the data member should be serialized into JSON with the specified field name (‘columnName’). The ‘columnName’ variable needs to be an exact match of the column name it is associated to inside the FAULS mobile service.

The illustration below is an example of the ‘AzureData’ class, specifically the ‘userLeagues’ class. The data members within this class have been prefixed with their serializable names which correspond to their matching column names inside the ‘userLeagues’ table.

4.1.3.3 AzureData:’ userLeagues’ Class and MobileService: ‘userLeagues’ Table

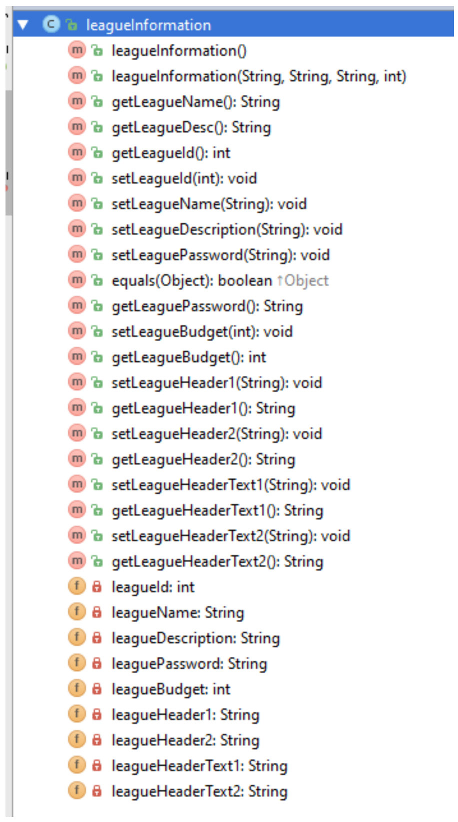


```
public class userLeagues {

    @SerializedName("Id")
    private String Id;

    @SerializedName("userId")
    private String userId;

    @SerializedName("leagueId")
    private String leagueId;

    @SerializedName("isAdmin")
    private Boolean isUserLeagueAdmin;

    @SerializedName("userBudget")
    private int userBudget;

    @SerializedName("userFantasyPlayerLinkId")
    private int mUserFantasyPlayerLinkId;

    //Default Constructor
    public userLeagues(String league, String user, Boolean isAdmin) {
        this.setLeagueId(league);
        this.setUserId(user);
        this.setUserBudget(0);
        this.setAdmin(isAdmin);
    }
```



```
FaulsMobileService.userLeagues
  Columns
    id (PK, int, not null)
    leagueId (int, null)
    userId (int, null)
    isAdmin (bit, null)
    userFantasyPlayerLinkId (int, null)
    __createdAt (datetimeoffset(3), not null)
    __updatedAt (datetimeoffset(3), null)
    __version (rowversion, not null)
    __deleted (bit, null)
    userBudget (int, not null)
```

Figure 4.1.8–AzureData Java Class and associated Mobile Service table.



```
leagueInformation
    leagueInformation()
    leagueInformation(String, String, String, int)
    getLeagueName(): String
    getLeagueDesc(): String
    getLeagueId(): int
    setLeagueId(int): void
    setLeagueName(String): void
    setLeagueDescription(String): void
    setLeaguePassword(String): void
    equals(Object): boolean ↑Object
    getLeaguePassword(): String
    setLeagueBudget(int): void
    getLeagueBudget(): int
    setLeagueHeader1(String): void
    getLeagueHeader1(): String
    setLeagueHeader2(String): void
    getLeagueHeader2(): String
    setLeagueHeaderText1(String): void
    getLeagueHeaderText1(): String
    setLeagueHeaderText2(String): void
    getLeagueHeaderText2(): String
    leagueId: int
    leagueName: String
    leagueDescription: String
    leaguePassword: String
    leagueBudget: int
    leagueHeader1: String
    leagueHeader2: String
    leagueHeaderText1: String
    leagueHeaderText2: String
```

Figure 4.1.9 – leagueInformation [AzureData] class structure

The picture above (see Figure 4.1.8 and 4.1.9) illustrate how the 'AzureData' classes are structured. In total there are 7 of these classes for each of the 7 tables being used by the application. Each class contains a data member for each column stored in the classes associated table, a default and custom constructor and properties (getter/setters) for each of the data members.

4.2 UI Design

A primary objective of this application is to provide users with an improved interface for managing their fantasy leagues. Whilst originally the application was to be devised solely for the use of HUMHC it later evolved to provide a generic interface facilitating any fantasy league application with teams of up to 11 players.

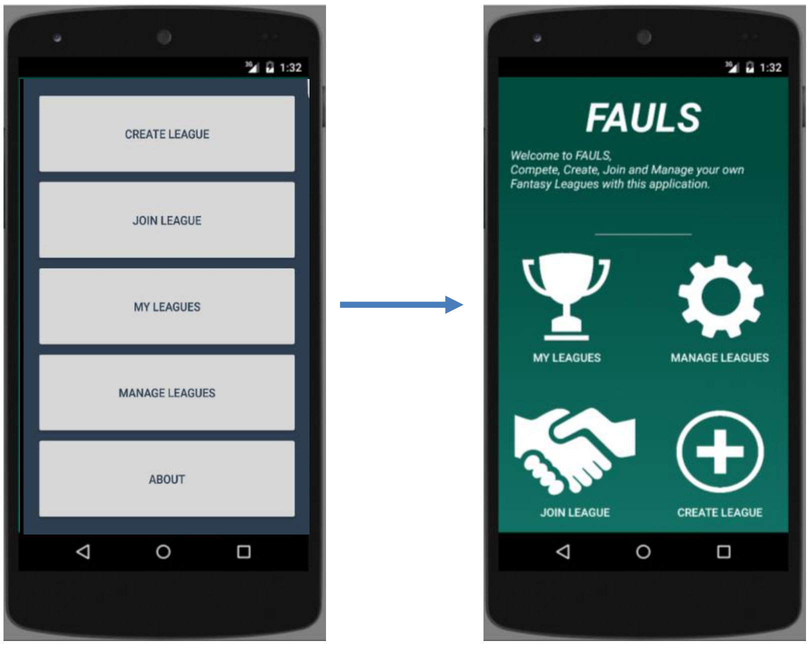
The User Interface design evolved overtime from initial design sketches to the use of graphical editing software to produce more graphically visual representations. These designs were shown to potential Users to obtain their opinions on the overall design.

Following the investigation of existing fantasy sports applications, a common design and set of features were identified. These existing applications helped form the basis of the FAULs application initial user interface design.

Additional reading was done into Androids new 'material' design ethos which, "synthesizes the classic principles of good design with the innovation and possibility of technology and science". (*Material Design, n.d.).* The material design philosophy and Androids documented processes for implementing in-app navigation and other design elements have been consistently referred back to when making design decisions. The key focus of material design is to be bold, intentional and provide meaning behind User motions.

4.2.1 Application Navigation

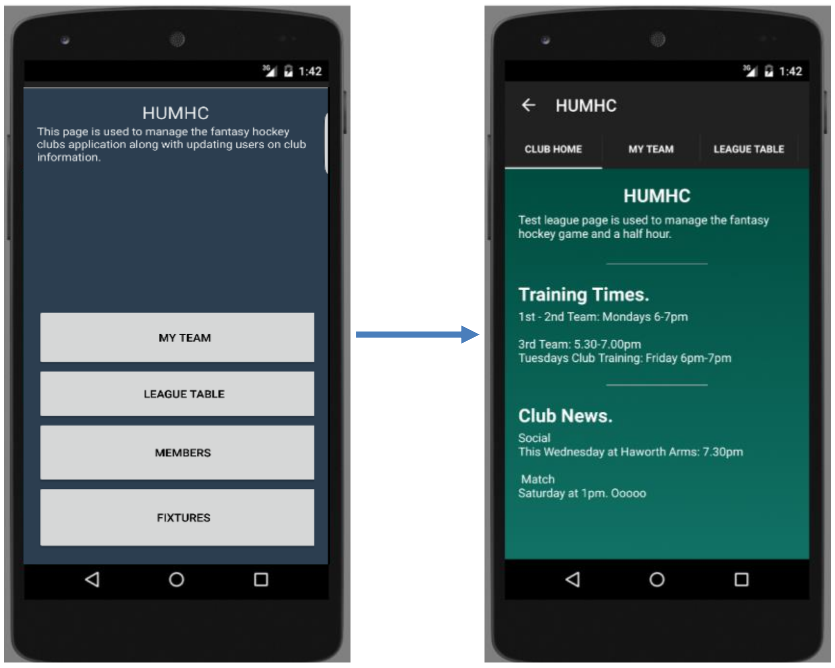
The application is divided into two sections, league management and User league participation. This division in functionality has resulted in the navigation including a mixture of descendant and lateral navigation.



Welcome to FAULS,
Compete, Create, Join and Manage your own Fantasy Leagues with this application.

Figure 4.2.1 – Application Main Menu dashboard design (initial → final)

As seen in Figure 4.2.1 the application has been designed with a dashboard interface as its main menu. Applying the feedback received from User testing the decision to use large, bold and appropriate graphics to provide navigation between key sections of the application had been incorporated. This design was used in place of a simple, yet functional list of menu items.



This page is used to manage the fantasy hockey clubs application along with updating users on club information.

Test league page is used to manage the fantasy hockey game and a half hour.

Training Times.
1st - 2nd Team: Mondays 6-7pm
3rd Team: 5.30-7.00pm
Tuesdays Club Training: Friday 6pm-7pm

Club News.
Social
This Wednesday at Haworth Arms: 7.30pm
Match
Saturday at 1pm. Ooooo

Figure 4.2.2 – Club Page design (initial → final)

The dashboard buttons on the applications main menu each lead to a level on a lower descent. Lateral navigation has been used for the League and Manage League club pages using a tab layout design.

The tab design enables users to swipe between tabs relevant to a league. This design provides a more natural movement and is more space efficient. As seen in figure 4.2.2 the Club Homepage is no longer cluttered with navigational controls as per the original design, thereby freeing up space for club information.

The direction and ordering of the tabs for both the League and League Management pages has been arranged so that it has a logical flow. The diagram shown in Figure 4.2.3 demonstrates navigation through the application from a User's perspective. The Manage league page has been designed to be a reflection of the users view only with additional functionality.

Note: More complete diagram of the applications flow can be seen in Appendix E.

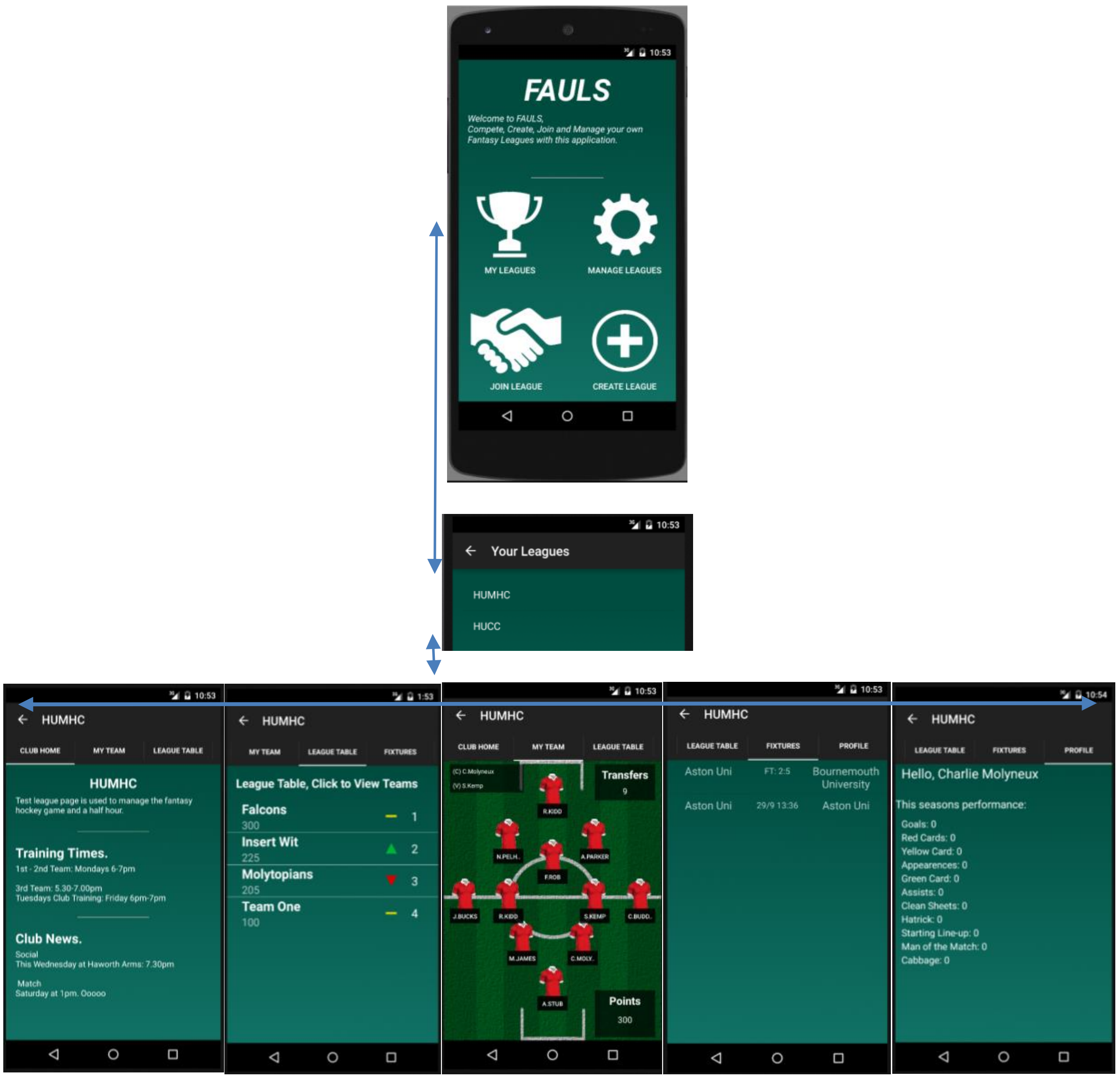


Figure 4.2.3 – Lateral user league navigation

4.2.2 Player Management Design

The interface had to be designed to make player management as simple and thoughtless a process as possible. League administrators could have between 11-16 players to update in anyone go (the most frequent number of players in a team based sport). If player management became too time consuming it may cause league administrators to lose interest in the service.

Additional design considerations were needed due to the generic and dynamic nature of the application. Due to the uncertainty of how many rules a league used for player tracking (between 1 and 11) the interface had to be designed to display the entire possible range.

The initial design illustrated below in Figure 4.2.4 was the mock design for the possible user Interface for managing players. The design of this page had to be intuitive and a rapid process to allow for the dynamic creation and management of the selected players personal statistics.

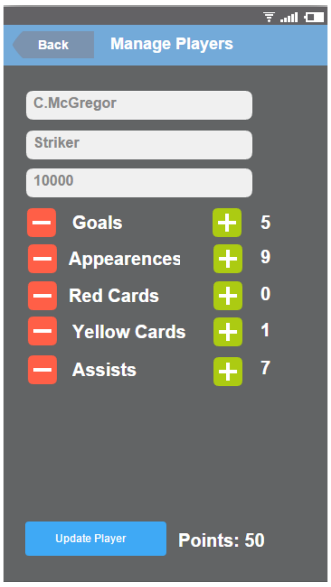
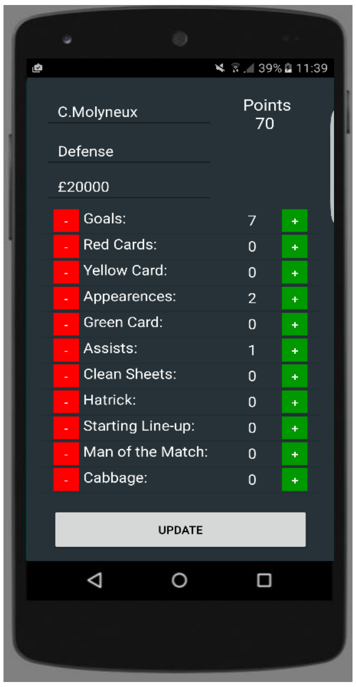


Figure 4.2.4 Mock-up Manage Players interface design

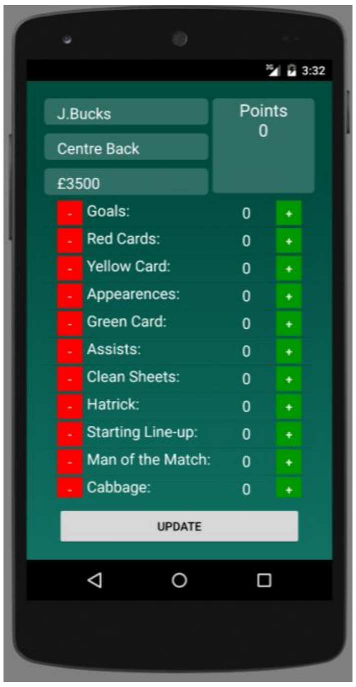
The initial design concept is one that made it all the way into the final product (with some minor adjustments). As pictured above in figure 4.2.4 the interface would be comprised of 4 static 'TextView' objects for player information alongside a Button object for updating the player. This design provides the administrator with all the information they need in a clear and structured manner.

The players' statistics will be drawn onto the screen using a 'ListView' container. The 'ListView' will then be assigned an adapter. This adapter will be a custom implementation to deliver the expected functionality in an aesthetically pleasing manner, this will be achieved by extending the 'BaseAdapter' class. The adapter will be populated during run-time with information stored inside the FAULs database. The final design of this screen can be seen below in Figure 4.2.5.



C.Molyneux
Points
70
Defense
£20000

Goals:	7
Red Cards:	0
Yellow Card:	0
Appearences:	2
Green Card:	0
Assists:	1
Clean Sheets:	0
Hatrick:	0
Starting Line-up:	0
Man of the Match:	0
Cabbage:	0



J.Bucks
Points
0
Centre Back
£3500

Goals:	0
Red Cards:	0
Yellow Card:	0
Appearences:	0
Green Card:	0
Assists:	0
Clean Sheets:	0
Hatrick:	0
Starting Line-up:	0
Man of the Match:	0
Cabbage:	0

Figure 4.2.5 – Prototype Design → Final Design Player Management Interface.

4.2.3 Data Entry (Players, Rules and Fixtures)

Another key requirement of this application was to allow league administrators to add Players and Rules. The interface needed to be designed in a similar manner so that data entry is as intuitive and rapid as possible through the application. A number of possible solutions had been considered and only one was feasible option post user testing.

The use of Dialogs would allow the league administrator to complete as few fields as required in a timely manner. The inclusion of auto complete text boxes will further enhance the data entry experience.

The illustrations below were used as mock representations as to how these dialogs could be designed.

Adding Rules

The diagram below (Figure 4.2.6) illustrates the initial functioning prototype version of the [New Rule] dialog. The design for this features interface followed very closely after the mock up design illustrated in Figure 4.2.7. However, the final design pictured below in Figure 4.2.8 opted for an additional ‘EditText’ object in place of the ‘NumberPicker’. This design choice was chosen following extensive user testing (explored in a later section) that found data entry for the Value field was far quicker with a simple digit keypad over a much slower incremental ‘NumberPicker’ object.

It is also worth noting the button placement changed from the original design. Having further investigated Googles material design and the dialog specification it became apparent dialog actions are to be placed on the right side of a dialog.

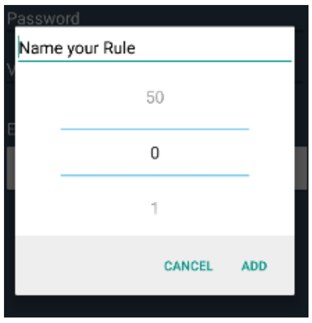


Figure 4.2.6 – Prototype Rule Dialog design

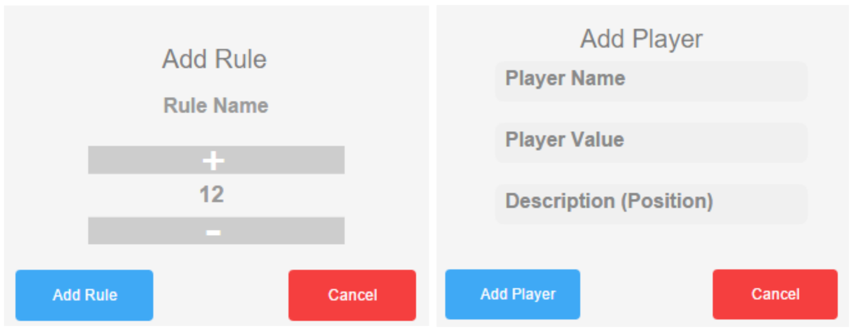


Figure 4.2.7 – Mock designs - data entry dialogs

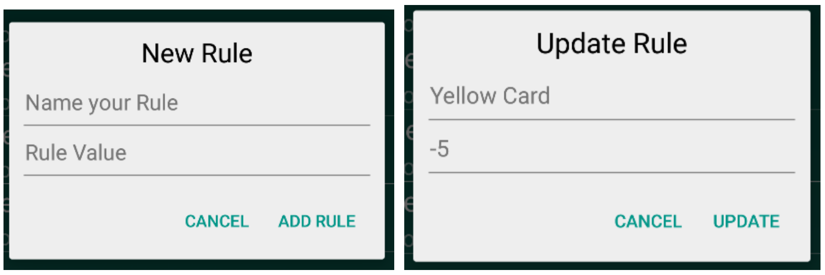


Figure 4.2.8 – Finalised Rule Dialog designs

Adding Players
Adding a player followed a similar design to adding and updating league rules. This decision was made as a result of the speed in which it allowed players to be entered as well maintaining a consistent design pattern throughout the application.

The finalised designs illustrated below (Figure 4.2.9) used the initial 'rule dialog' mock-ups as a base template. However, there was a slight change to the colour scheme and button placement due to the adoption of the material design dialog specification.

To enhance the user experience, the 'Description' field was prepopulated with an array of generic sporting positions. The success of these designs has been assessed within the User testing section of this report.

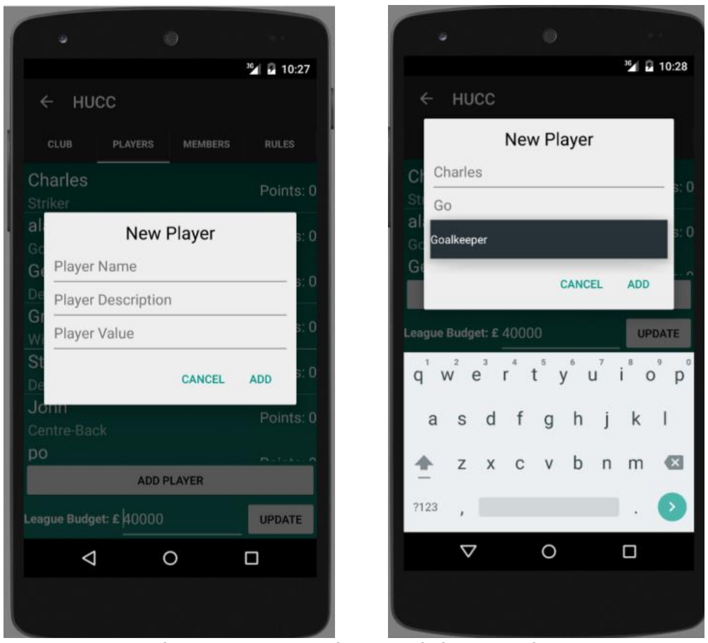
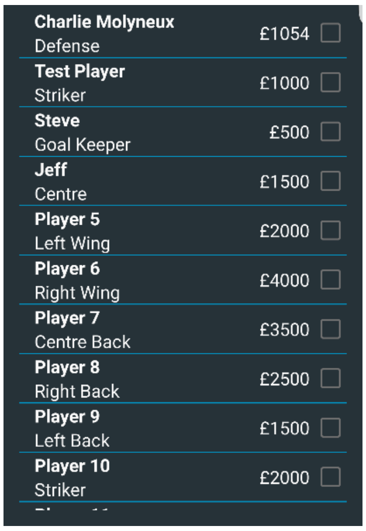


Figure 4.2.9 – Player Dialog Design

4.2.4 Team Management

Following the investigation of existing 'fantasy sport' applications, there was already a clear precedent as to what a user expected from this type of application. The initial prototype version was not a reflection of this standard and provided a less than friendly (or even intuitive) interface for creating a team. The prototype design used a simple 'ListView' object (illustrated below) to allow Users to select 11 players within the leagues budget.



Charlie Molyneux Defense	£1054	☐
Test Player Striker	£1000	☐
Steve Goal Keeper	£500	☐
Jeff Centre	£1500	☐
Player 5 Left Wing	£2000	☐
Player 6 Right Wing	£4000	☐
Player 7 Centre Back	£3500	☐
Player 8 Right Back	£2500	☐
Player 9 Left Back	£1500	☐
Player 10 Striker	£2000	☐

Figure 4.2.10 – Prototype team selection page

Whilst entirely functional it did not provide the User with an enjoyable or engaging experience. "The application should be responsive and provide a good experience" was a clear aim set by Objective 1 of this project. The above prototype failed to meet this objective.

The design and functionality of the team selection page is possibly one of the most important to this project. The interface required a graphically pleasing and interactive design that would allow all relevant information to be displayed including transfers, team score and the team's captains/vice-captains. Whilst the prototype design pictured in Figure 4.2.10 would prove fully functional it was quickly replaced by the finalised version seen in Figure 4.2.11.

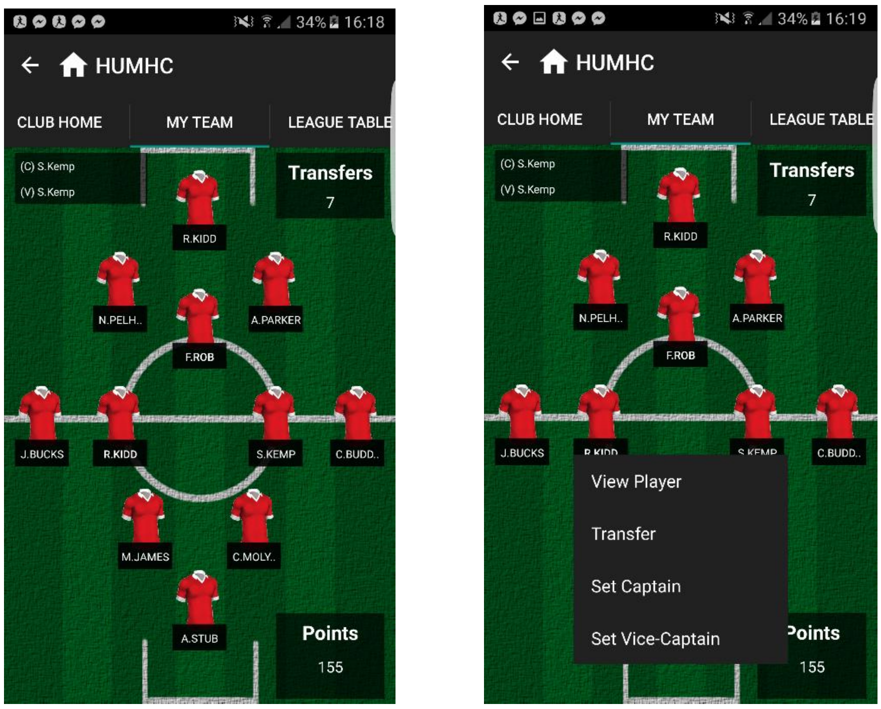


Figure 4.2.11 – My Team view (including player selected).

The design of the team page was inspired from existing fantasy sporting applications alongside sketches produced during the initial design process and user testing.

As illustrated the design provides a graphically rich page with all the essential information for a user to manage their team. Selecting a player provides the user with a number of options to explore and manage their players in a seamless and intuitive manner.

4.2.5 Fixtures

Objective 3 stated the requirement of an ‘Enhanced Club Page’ this included the ability for an administrator to manage club fixtures and a method for users to view these. The design of entering fixture data had to provide a quick and responsive solution. As fixture information cannot always be so easily obtained due to the low level leagues in which some teams play, data entry had to be manually entered.

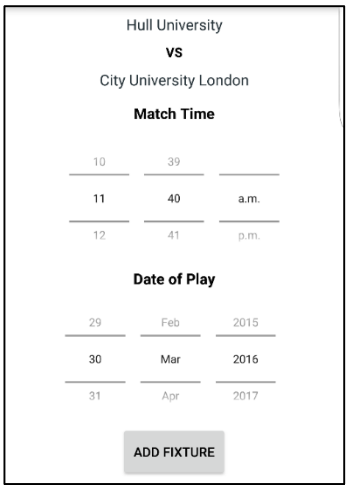


Figure 4.2.13 – Initial fixture management designs

The design concepts illustrated above in Figure 4.1.13 demonstrate how creating a new fixture will be managed through the application. The interface has been designed so the required data entry fields have been arranged in a logical manner. The screen will provide two prepopulated dropdown boxes for the Home/Away teams. The Time and Date Picker objects have been used to hasten entry and to limit erroneous entries (through mismatching date formats).

Once a game has been played a league administrator will be able to click on the fixture. This will show a dialog box which helps maintain the applications consistency of using them for data entry. The dialog has been designed so it provides a clear display of what exact information is required for that section.

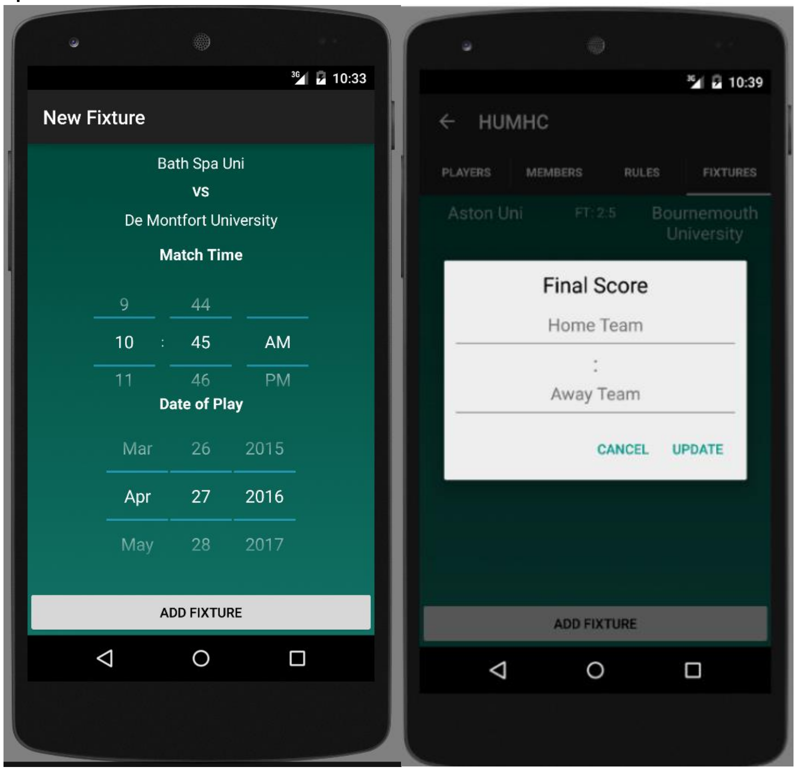


Figure 4.2.14 - Finalised fixture screen design

4.3 Experimental Design

Note: No personal information was recorded or retained during this testing phase.

User Interface Design

The design and implementation of the applications interface evolved during the project lifecycle. Initial designs were used to build up an idea as to how the final product would appear. However, some of the smaller details required user testing and feedback.

The projects objectives included providing users a “responsive” and “good experience”. The fulfilment of these objectives could only be met by engaging with the target audience. Members of HUMHC were used to test certain aspects of the interface and functionality. Feedback was obtained and used help influence further design decisions.

A primary concern facing this project was the ability to input data in a timely and intuitive manner. The experiments outlined below focus on the applications data entry functionality. Each of the 3 test users were given sample test data to input into the application. The purpose of these experiments was to help determine the best design for data entry. Test data can be found in Appendix M.

Test 1: Adding 11 League Rules

Number Picker			
Test Id	**Time Taken Seconds** (+/- 1s)		
-	**User 1**	**User 2**	**User 3**
1	134	114	152
2	125	100	140
3	136	105	142

EditText			
Test Id	**Time Taken Seconds** (+/- 1s)		
-	**User 1**	**User 2**	**User 3**
1	101	97	104
2	70	84	98
3	105	95	106

The above test compared the time taken to add 11 rules to a league. The test aimed to assess the usefulness of the ‘NumberPicker’ object for assigning rule values. The above results and feedback from testing yielded an overall consensus that was in favour of the ‘EditText’ field. Whilst the ‘NumberPicker’ was graphically more pleasing and allowed for less margin of error the ‘EditText’ field was far quicker at entering larger values.

The same conclusion was found for adding fantasy players to a league. With league budgets being set in the thousands a ‘NumberPicker’ object was not suitable for the task of assigning a player value. Additionally, having to hardcode the values assigned to a ‘NumberPicker’ took away the freedom and flexibility the application wants to deliver.

Test 2: Updating 3 Players

Buttons			
Test Id	**Time Taken Seconds** (+/- 1s)		
-	**User 1**	**User 2**	**User 3**
1	15	14	15
2	17	11	16
3	15	14	13

EditText			
Test Id	**Time Taken Seconds** (+/- 1s)		
-	**User 1**	**User 2**	**User 3**
1	32	30	41
2	36	34	32
3	34	35	37

The second test assessed how best to manage fantasy player scores. Again, two designs were considered. Firstly, incremental buttons to increment/decrement the score by 1 with each click. Secondly, an ‘EditText’ field for inputting the rules occurrence manually.

As discussed a player may have 11 league rules associated to their profile. At any one point (using Hockey as an example) an administrator could have up-to 16 players to update. The application had to provide an interface that would allow for a players’ rule occurrence to be updated rapidly to prevent the administrator losing interest.

The conclusion drawn from this test and user feedback it was agreed that the incremental button design was far quicker, more responsive and more aesthetically pleasing. As a result, this was the design that made it into the final version of the mobile application.

4.4 Test Design and System Testing

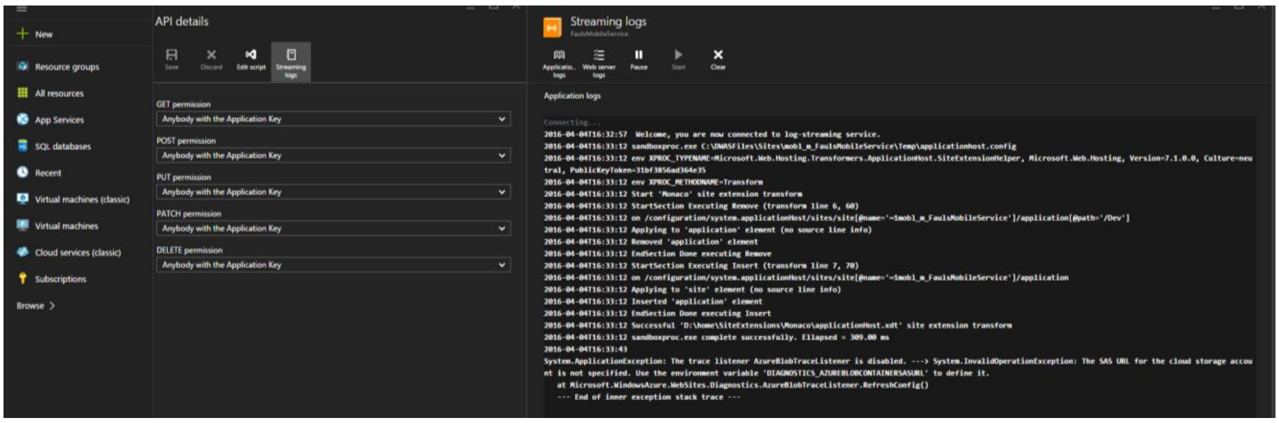
The application needed to be thoroughly tested to ensure it was robust enough for delivery. Testing for this project fell under two key phases. Firstly, Unit Testing was required whenever a new testable method had been implemented. These tests were performed manually, or where possible automated using the JUnit testing library. The second phase was acceptance testing. These tests ensured the applications functionality operated as expected and that UI components were displaying and behaving as expected on a range of different devices.

4.4.1 Debugging

With a fair portion of the applications logic and functionality occurring server side (i.e. updating tables) debugging was not always a simple process.

During initial development when attempting a table insert, delete, update or API invocation from the client it was not always successful. Debugging the issue from Android Studio often yielded little useful explanation. Azure would return a generic HTTP status code which would only give some idea as to what the problem cause might be. As a result, debugging required the use of both Android Studios debugging environment and more importantly, Azures diagnostic logs.

The Azure diagnostic logs appear as shown below in Figure 4.4.1. Debugging required reading through the logs and trying to determine the cause of the issue. Issues were usually caused by typographical errors in table names, incorrect syntax for SQL queries and/or incorrect value types being inserted into columns (i.e. String into an Integer field).



GET permission: Anybody with the Application Key
POST permission: Anybody with the Application Key
PUT permission: Anybody with the Application Key
PATCH permission: Anybody with the Application Key
DELETE permission: Anybody with the Application Key

```
Connecting...
2016-04-04T16:32:57  Welcome, you are now connected to log-streaming service.
2016-04-04T16:33:12 sandboxproc.exe C:\DWASFiles\Sites\mobl_m_FaulsMobileService\Temp\applicationhost.config
2016-04-04T16:33:12 env XPROC_TYPENAME=Microsoft.Web.Hosting.Transformers.ApplicationHost.SiteExtensionHelper, Microsoft.Web.Hosting, Version=7.1.0.0, Culture=neu
tral, PublicKeyToken=31bf3856ad364e35
2016-04-04T16:33:12 env XPROC_METHODNAME=Transform
2016-04-04T16:33:12 Start 'Monaco' site extension transform
2016-04-04T16:33:12 StartSection Executing Remove (transform line 6, 60)
2016-04-04T16:33:12 on /configuration/system.applicationHost/sites/site[@name='~1mobl_m_FaulsMobileService']/application[@path='/Dev']
2016-04-04T16:33:12 Applying to 'application' element (no source line info)
2016-04-04T16:33:12 Removed 'application' element
2016-04-04T16:33:12 EndSection Done executing Remove
2016-04-04T16:33:12 StartSection Executing Insert (transform line 7, 70)
2016-04-04T16:33:12 on /configuration/system.applicationHost/sites/site[@name='~1mobl_m_FaulsMobileService']/application
2016-04-04T16:33:12 Applying to 'site' element (no source line info)
2016-04-04T16:33:12 Inserted 'application' element
2016-04-04T16:33:12 EndSection Done executing Insert
2016-04-04T16:33:12 Successful 'D:\home\SiteExtensions\Monaco\applicationHost.xdt' site extension transform
2016-04-04T16:33:12 sandboxproc.exe complete successfully. Ellapsed = 309.00 ms
2016-04-04T16:33:43
System.ApplicationException: The trace listener AzureBlobTraceListener is disabled. ---> System.InvalidOperationException: The SAS URL for the cloud storage accou
nt is not specified. Use the environment variable 'DIAGNOSTICS_AZUREBLOBCONTAINERSASURL' to define it.
   at Microsoft.WindowsAzure.WebSites.Diagnostics.AzureBlobTraceListener.RefreshConfig()
   --- End of inner exception stack trace ---
```

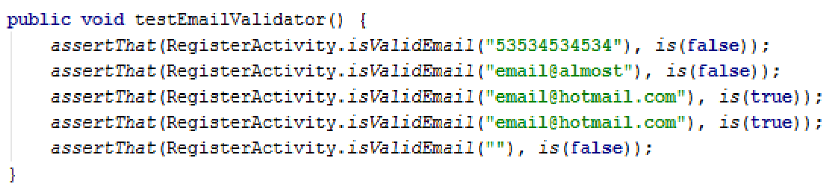
Figure 4.4.1 – Azure Streaming Diagnostic logs

4.4.2 Unit Testing

Where possible Unit tests were used to test individual methods. Unit tests helped ensure methods could handle any data whilst adhering to the standards specified by the data model. With data entry being a pivotal feature of the applications objectives it was imperative to ensure no erroneous data was being entered into the mobile service tables, and that any ‘dirty’ data is handled gracefully.

The application has been designed to try and mitigate the possibility of erroneous data being entered (e.g. submit buttons cannot be pressed whilst text fields are empty, using autocomplete text fields, number and date pickers and incremental buttons to adjust values opposed to manual entry). However, in some cases ‘EditText’ fields give the user free reign as to what can be entered. Android allows ‘EditText’ fields to be assigned an ‘input type’ (digits, password, text, etc.) which helps direct the user. Additionally, it has been necessary to ensure no unwanted data makes it through to the methods responsible for communicating with the mobile service.

An example unit test can be seen below in Figure 4.4.2. This test was written to ensure the email field inside the register activity correctly validated email addresses prior to registering the user.



```
public void testEmailValidator() {
    assertThat(RegisterActivity.isValidEmail("53534534534"), is(false));
    assertThat(RegisterActivity.isValidEmail("email@almost"), is(false));
    assertThat(RegisterActivity.isValidEmail("email@hotmail.com"), is(true));
    assertThat(RegisterActivity.isValidEmail("email@hotmail.com"), is(true));
    assertThat(RegisterActivity.isValidEmail(""), is(false));
}
```

Figure 4.4.2 – Sample Unit Test, validating email

4.4.3 Acceptance Testing

A range of devices were used to test the mobile application, both physical and emulated. For physical testing the Samsung Galaxy S6 and LG4 devices were used. The device was also tested on a Nexus 5 emulator. These devices have screen sizes ranging from 4.95 to 5.5". The application had been designed to function correctly on these devices.

Testing the application on a number of devices helped ensure the user interface was being displayed as expected. Initial release will only support a select few devices due to the projects time constraints. Supporting any additional devices would greatly increase the required work and testing. Using a smaller range of devices has allowed for a more exhaustive test plan to be followed.

Acceptance testing was an integral part in verifying that implemented functionality behaved as expected. Tests were being continuously updated as new features were being implemented. The tests helped ensure that new functionality worked and did not have a negative effect on existing and signed-off functionality.

These acceptance tests tested a range of different scenarios for each of the applications activity or fragment classes. A sub-set of the tests followed can be seen below with a more complete set of tests being found in Appendix F.

Page	Test	Expectation	Pass	Comment
Manage Leagues	Updating Club Information	Club informtation is updated correctly	P	
Manage Leagues	Updating Budget	League budget is updated correctly.	P	
Manage Leagues	Calculate Weekly Point	Points are correctly calculated for each team present in the league.	P	
Manage Leagues	Calculate Player Points.	Players have their points correctly calculated when their statistics change.	P	
Manage Leagues	Merge a User account with a players profile	User correctly has their profile merged. Profile can be viewed in the User Leagues activity.	P	
Manage Leagues	Updating a players transfer count	Players have their transfer count incremented in the Mobile Service	P	

4.5 System Implementation

4.5.1 Implementing the Interface

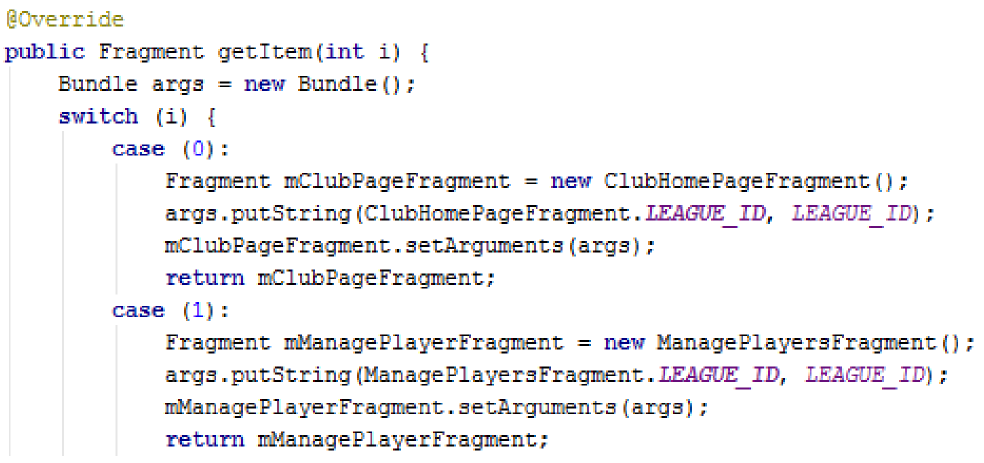
The implementation of the applications interface was essential in meeting the aims and objectives of this project. This section covers some of the more interesting aspects of the applications interface design and implementation of the programs functionality.

Due to the generic nature of this application, it was not always apparent as to what values or number of objects in a list (as an example) would be required to be displayed on a page. This required a large portion of the objects to be drawn or interacted with programmatically (i.e. setting Text values).

Android user interfaces are primarily designed and constructed using XML layout resource files. Android Studio has both a text and GUI editor for populating the layout files with desired objects. Android User interface objects can include (but are not limited to) a number of useful widgets, containers and text fields for displaying information. In some cases, it has been required to extend these interface objects to enhance their base functionality and design to meet the aims and objectives of this project.

The delivered application used both Activity and Fragment classes to provide the desired functionality and User experience. The User and Manage league sections are contained inside their own base Activities. Each of these Activities contains a singular 'ViewPager' object inside their respective Layout files. The 'ViewPager' class is a "Layout manager that allows users to flip left and right through pages of data". (Android Developer, 2016).

For this application the 'ViewPager' has been assigned a 'FragmentStatePagerAdapter'. This adapter is responsible for handling the saving and restoration of a fragment's state, as well as for inflating the correct view onto the 'ViewPager'. When a User swipes between the tabs or selects one from the action bar the 'PagerAdapter' class calls the 'getItem(int)' method (see Figure 4.5.1). This method returns the appropriate Fragment to the 'ViewPager'.



```
@Override
public Fragment getItem(int i) {
    Bundle args = new Bundle();
    switch (i) {
        case (0):
            Fragment mClubPageFragment = new ClubHomePageFragment();
            args.putString(ClubHomePageFragment.LEAGUE_ID, LEAGUE_ID);
            mClubPageFragment.setArguments(args);
            return mClubPageFragment;
        case (1):
            Fragment mManagePlayerFragment = new ManagePlayersFragment();
            args.putString(ManagePlayersFragment.LEAGUE_ID, LEAGUE_ID);
            mManagePlayerFragment.setArguments(args);
            return mManagePlayerFragment;
```

Figure 4.5.1 – Page adapter class sample code

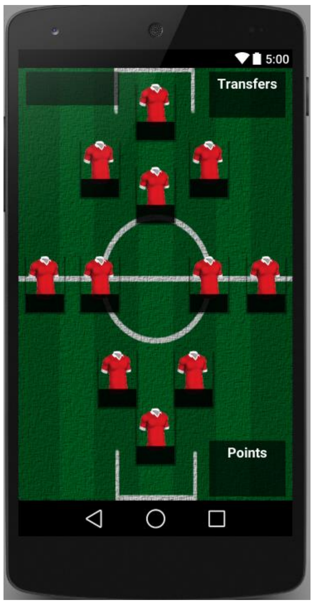
Android Activities and Fragments inflate their designated layout files once the onCreateView() method has been called. Layout files are inflated onto the mobile screen using the 'SetContentView(R.layout.[layout_file_name)'; and 'inflate(R.layout.example_fragment_layout_file, *null*)'; methods respectively. Interface objects can then be retrieved and manipulated using the 'findViewById()' method.

4.5.2 Team Management

A key example of user interface objects being programmatically dynamically populated include the ‘My Team’ and ‘League Table’ screens.

When the application is initially compiled it is unaware of which league the User is a member of or will decide to view. During application run time the mobile client queries all information relating to the league selected by a User. The application must quickly determine who is in their team and correctly populate the interface objects with the correct information.

The team page as illustrated in Figure 4.5.2 has 11 ‘Button’ and 4 ‘TextView’ objects. Initially these objects contain no values as the appropriate Azure tables need to be queried in order to populate the UI with the information a User needs to
successfully manage their team.



1. Invoke ‘getplayers’ API
2. Deseralise results
3. runOnUiThread
    a. update UI objects

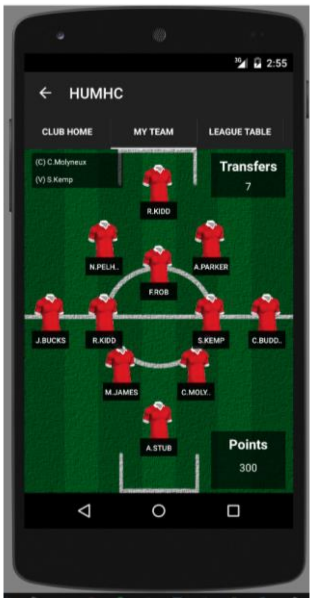


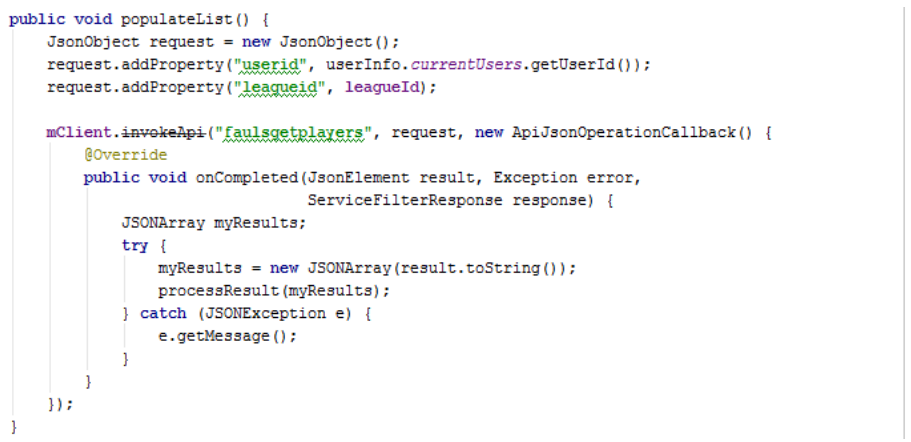
Figure 4.5.2 – populating the Team View

Implementation
The full implementation of this page and view required data stored from both the ‘fantasyPlayer’ and ‘userTeam’ tables. The view would use information stored from both these tables to populate the layouts widgets and UI components with the appropriate information. Initially the functionality for retrieving this information had been implemented by executing a number of queries to the mobile service table.

However, this proved to be quite a slow process as no method existed to easily join multiple tables together. Instead the process required to loop through each of ‘userTeam’ [playerId] columns and then execute multiple queries to the ‘fantasyPlayer’ table to find the designated player.

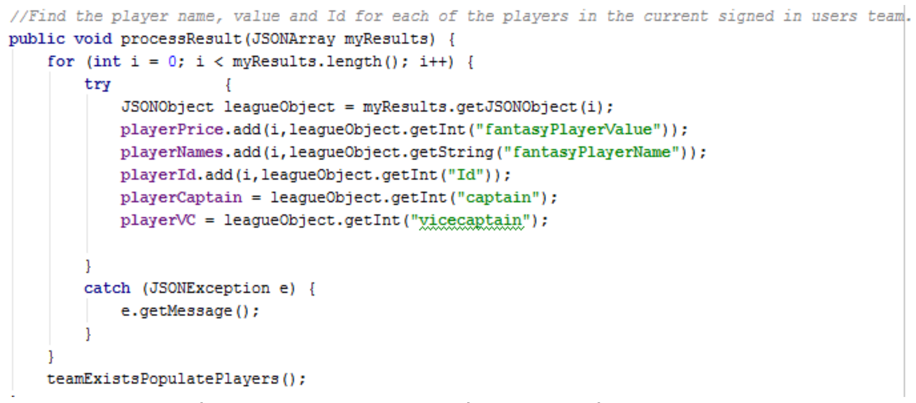
As a result, the page did not deliver on the required ‘responsiveness’ the application sought to deliver. This led to a ‘Custom API’ being written (see Appendix H) using a SQL query which was able to join the tables together server side and return the required information almost instantaneously.

This returned data was then processed and deserialised by the methods seen in Figure 4.5.3. The application now had the required information to populate the team page. This was achieved by calling the ‘teamExistsPopulatePlayers()’ method which executes an Asynchronous ‘runOnUiThread’ thread which allows for UI objects to be updated with the fantasy player names, points, remaining transfers and captaincy information.



```java
public void populateList() {
    JsonObject request = new JsonObject();
    request.addProperty("userid", userInfo.currentUsers.getUserId());
    request.addProperty("leagueid", leagueId);

    mClient.~~invokeApi~~("faulsgetplayers", request, new ApiJsonOperationCallback() {
        @Override
        public void onCompleted(JsonElement result, Exception error,
                                ServiceFilterResponse response) {
            JSONArray myResults;
            try {
                myResults = new JSONArray(result.toString());
                processResult(myResults);
            } catch (JSONException e) {
                e.getMessage();
            }
        }
    });
}
```

Figure 4.5.3 – Method to Invoke the ‘customAPI’



```java
//Find the player name, value and Id for each of the players in the current signed in users team.
public void processResult(JSONArray myResults) {
    for (int i = 0; i < myResults.length(); i++) {
        try            {
            JSONObject leagueObject = myResults.getJSONObject(i);
            playerPrice.add(i,leagueObject.getInt("fantasyPlayerValue"));
            playerNames.add(i,leagueObject.getString("fantasyPlayerName"));
            playerId.add(i,leagueObject.getInt("Id"));
            playerCaptain = leagueObject.getInt("captain");
            playerVC = leagueObject.getInt("vicecaptain");

        }
        catch (JSONException e) {
            e.getMessage();
        }
    }
    teamExistsPopulatePlayers();
```

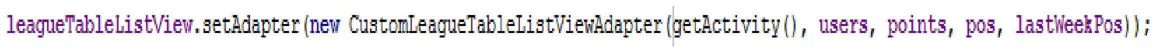
Figure 4.5.4 – Process the returned JSON Array

League Table

The ‘League Table’ tab is another key example of using Azure table data to dynamically populate a UI object.

The League Table uses a ‘ListView’ UI object to present the required data in a logical and structured manner (ordered by user team ranking). However, the base ‘ListView’ adapter is limited in its design and contains only 1 ‘TextView’ object. As a result of this limitation a custom ‘ListView’ adapter has been implemented. This is achievable by extending the Android ‘BaseAdapter’ class.

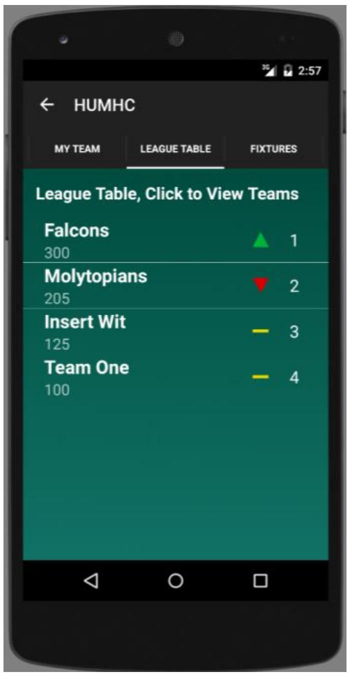
In this example the League Table ‘ListView’ object initialises the custom ‘ListView’ adapter with the activities context, 3 String ‘ArrayLists’ (Team Name, Team Points and Rank) and an Integer ‘ArrayList’ containing the Users previous rank position.



```java
leagueTableListView.setAdapter(new CustomLeagueTableListViewAdapter(getActivity(), users, points, pos, lastWeekPos));
```

This custom ‘ListView’ adapter inflates the specified layout resource file. The resource file is used as the template for each ‘ListView’ item. In this example each ‘ListView’ item is comprised of 3 ‘TextView’ and 1 ‘ImageView’ UI object. The list is programmatically populated with data once it has been initialised as seen above.

The adapters logic selects the appropriate image to display for the rank change and populates the ‘TextViews’ using the data stored within the Arrays. The result of this custom implementation can be seen in Figure 4.5.5. The implementation and design of a custom ‘ListView’ layout has provided a much cleaner and professional looking ‘ListView’ layout design. (pictured).



League Table, Click to View Teams

Falcons 300	▲	1
Molytopians 205	▼	2
Insert Wit 125	—	3
Team One 100	—	4

Figure 4.5.5 – Dynamically populated Custom ListView adapter (League Table)

4.5.3 Fantasy Players Management

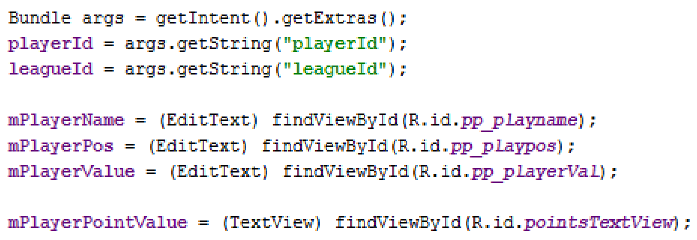
The core principle of a ‘fantasy sports’ application is that real-world players play real-world games and then in turn their fantasy counterpart has their profile updated to reflect their real-world performance.

As previously discussed a fantasy league can have ‘0 to Many’ fantasy players. These fantasy players can then have between 1 and 11 league specific rules associated to their fantasy profile.

The application has been designed to be used as a generic template for many different sports to use. As such the exact number of league rules and player information is unknown until application run-time. This has influenced the implementation of the fantasy player management page

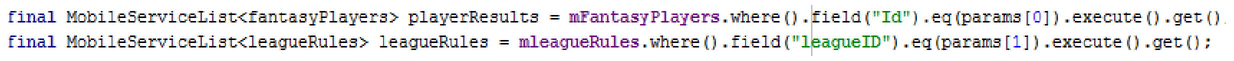
The Manage Players Activity receives player information stored inside an Intent object. An Intent “can be thought of as the glue between activities” (*Developer Android, 2016*) and can store extras inside a ‘Bundle’ class instance which is defined as “a mapping from String values to various parcelable types” (*Developer Android*, 2016)

The Fragment contains a ‘ListView’ container which displays all the players associated to the current league. Once a player is selected the players ‘playerId’ and ‘leagueId’ is stored inside the Intent to be passed to the ‘ManagePlayers’ Activity (Figure 4.5.6). The ‘ManagePlayers’ Activity uses this additional information to query the Azure mobile service tables to retrieve the player information as well as the league rules and the players rule occurrence values. In addition, the ‘leagueId’ is also passed into the Activity to query and retrieve the appropriate rows from the ‘leagueRules’ table.



```
Bundle args = getIntent().getExtras();
playerId = args.getString("playerId");
leagueId = args.getString("leagueId");

mPlayerName = (EditText) findViewById(R.id.pp_playname);
mPlayerPos = (EditText) findViewById(R.id.pp_playpos);
mPlayerValue = (EditText) findViewById(R.id.pp_playerVal);

mPlayerPointValue = (TextView) findViewById(R.id.pointsTextView);
```

Figure 4.5.6 – Using intents to get player information



```
final MobileServiceList<fantasyPlayers> playerResults = mFantasyPlayers.where().field("Id").eq(params[0]).execute().get()
final MobileServiceList<leagueRules> leagueRules = mleagueRules.where().field("leagueID").eq(params[1]).execute().get();
```

Figure 4.5.7 – retrieving the specified fantasy player from the mobile service tables

The interface for managing a player includes 3 ‘EditText’ fields, 1 ‘TextView’ field and a ‘ListView’ container. The fields and containers are populated with the required information at run time using the information stored inside the ‘fantasyPlayer’ and ‘leagueRules’ table (See Figure 4.5.7).

The back-end database has facilitated this through the implementation of the designs proposed in section 4.1 which included 2 key tables, ‘fantasyPlayers’ and ‘leagueRules’. Similar to the designs the ‘fantasyPlayer’ table has 11 columns ([ruleOcc1], [ruleOcc2], …, [ruleOcc11]). The value inside these columns relates to the number of times a particular rule has occurred for that player.

Once the required table data has been retrieved, a method is executed. The executed method iterates through the ‘leagueRules’ table results in conjunction with the ‘fantasyPlayers’ result. The rule name is stored inside a String ‘ArrayList’ and the fantasy players rule occurrence is stored inside a separate ‘TextView’ ‘ArrayList’.

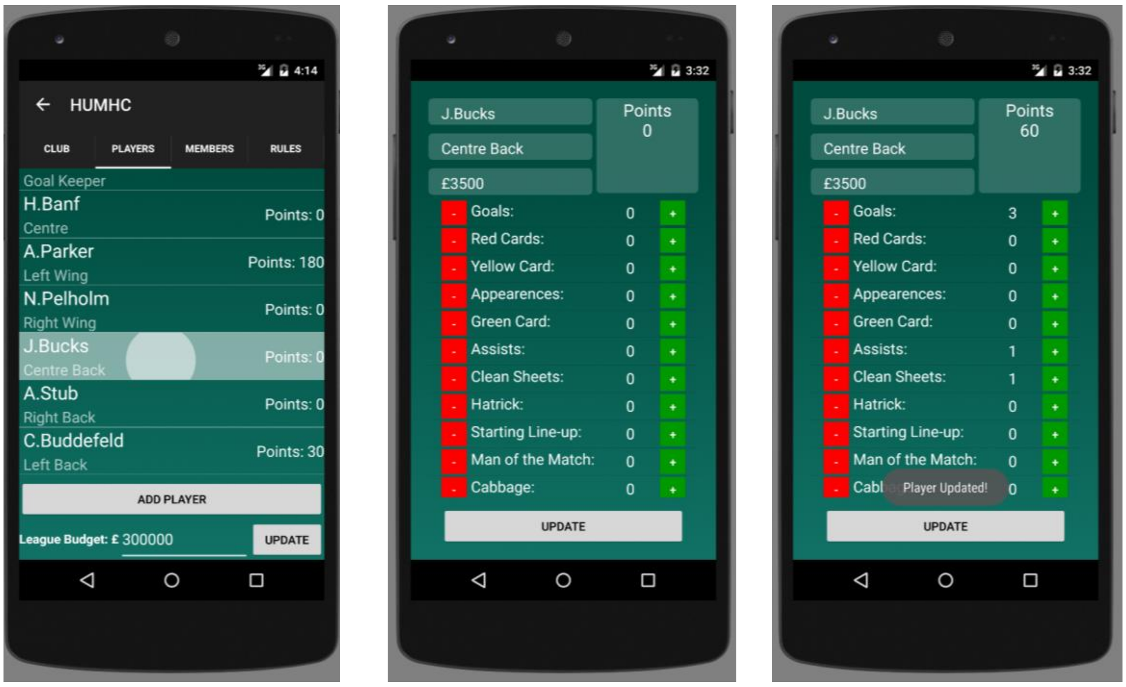
The required functionality for this page was to deliver an intuitive manner for league administrators to manage and update their players. Following the experimental testing phase the easiest way to update a player’s statistics was through buttons.

The implementation of the player ‘statistic’ section was achieved by implementing another custom ‘ListView’ adapter. The layout file inflated by this adapter included two Buttons ([-], [+]) and two ‘TextViews’ to contain the rule name and player rule occurrence. The adapter implemented an ‘onClickListener’ event for each of the buttons which adjusts the rule occurrence value appropriately and notifies the ‘ListView’ a change has been made.

Once the league administrator is ready to commit the changes made they simply need to click the [Update Player] button (*see* Figure *4.5.8*). This invokes the ‘updatePlayer(*playerId*)’ method which executes an AsyncTask to update the player in the background and then update the UI accordingly. The AsyncTask object has been used throughout the application to “perform background operations and publish results on the UI thread without having to manipulate threads” (*Android Developer, 2016*). The Android Development documentation states the AsyncTask object is ideal for short operations (lasting a couple of seconds).

This method will update the appropriate fantasy player with the new values stored inside the ‘EditText’ fields and the rule occurrence point array. Following the table update the user interface updates to reflect the changes made to the fantasy players point value.

A player’s point value is calculated through the multiplication of the [ruleValue] column stored inside the ‘leagueRules’ table and the value stored inside the corresponding [ruleOcc*(x)*] column.


HUMHC

CLUB PLAYERS MEMBERS RULES

Goal Keeper
H.Banf Centre Points: 0
A.Parker Left Wing Points: 180
N.Pelholm Right Wing Points: 0
J.Bucks Centre Back Points: 0
A.Stub Right Back Points: 0
C.Buddefeld Left Back Points: 30

ADD PLAYER

League Budget: £ 300000 UPDATE

J.Bucks
Centre Back
£3500
Points 0

	Before	After
Goals:	0	3
Red Cards:	0	0
Yellow Card:	0	0
Appearences:	0	0
Green Card:	0	0
Assists:	0	1
Clean Sheets:	0	1
Hatrick:	0	0
Starting Line-up:	0	0
Man of the Match:	0	0
Cabbage:	0	0

Points 60

Player Updated!

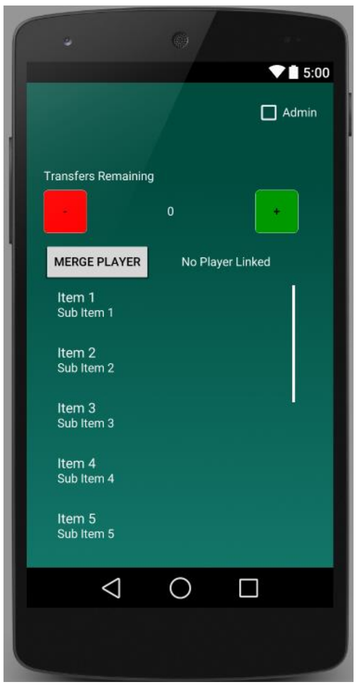
UPDATE

Figure 4.5.8 – Selecting a player, incrementing values and committing the changes

4.5.4 User Management

As per the objectives the application required the functionality to merge user accounts with their fantasy player counterpart. In addition to this, the ability to manage user administrator status and player transfer limits became part of the requirements.

The process involves the administrator selecting a user from the ‘users in league’ tab. When a user is selected an intent is assigned a number of ‘extras’ including the user id, league id, name and team name. The blank template (Figure 4.5.9) is then populated using these values during the ‘onCreate()’ method (Figure 4.5.10).


Admin

Transfers Remaining

0

MERGE PLAYER No Player Linked

Item 1
Sub Item 1
Item 2
Sub Item 2
Item 3
Sub Item 3
Item 4
Sub Item 4
Item 5
Sub Item 5

Figure 4.5.9 – Unpopulated Manage User Template Screen

Administrator status is managed using a checkbox. (Un)checking this box will invoke an ‘updateAdminStatus(Boolean isChecked)’ method. This method starts a new asynchronous task to update the ‘userLeagues’ table in the background.

Player transfer limits is managed using two incremental buttons. These buttons update the value displayed on the interface and then invoke a ‘updateTransfer(transRemainingCount)’ method. In a similar implementation to the admin status an AsyncTask is used to update the ‘userTeam’ table containing the transfer count for a given user.

Objective 4 stated the requirement of associating a user profile to a fantasy player profile. This feature had been implemented using a ‘custom API’ (*Appendix I*). When a player has been selected from the ‘ListView’ the application creates a JSON Object to be passed to the API. The API deserialises these variables and uses them to construct the query responsible for updating the ‘userLeagues’ table to include the selected fantasy players Id.

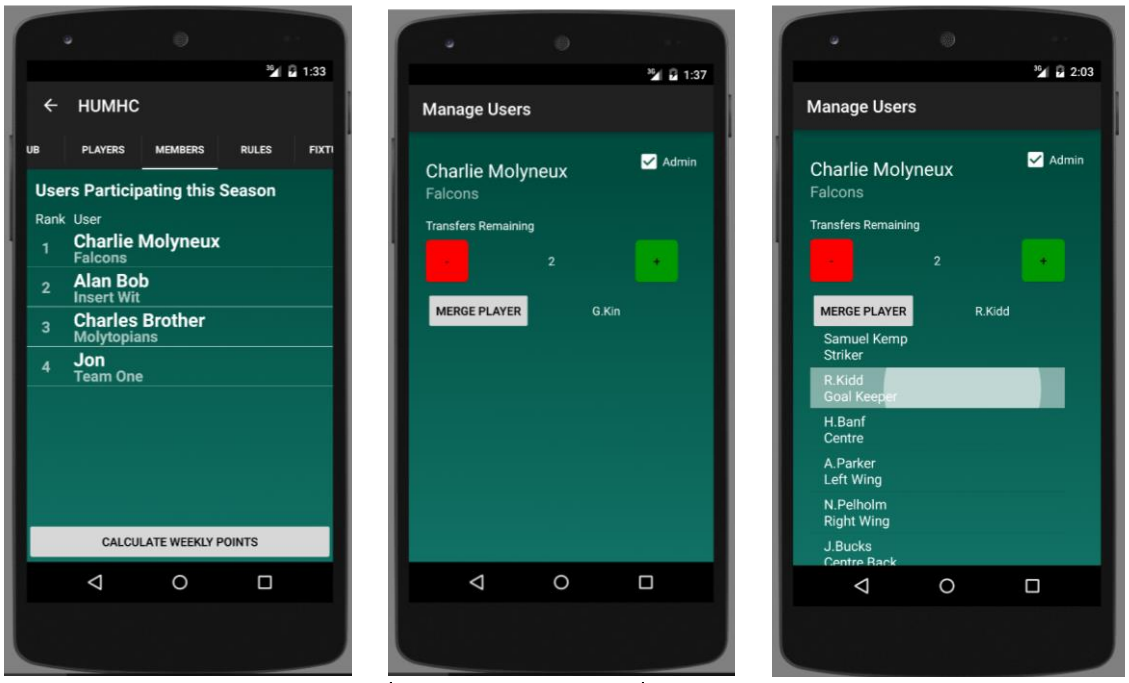
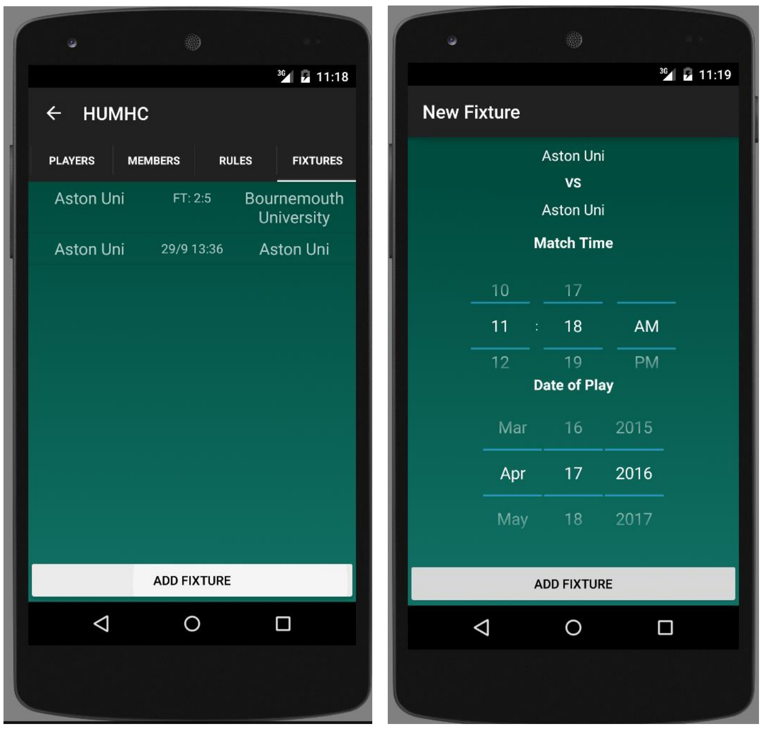


Figure 4.5.10 – Managing a User

4.5.5 Fixture Management

As part of Objective 3 the application had to provide the functionality for league administrators to add and manage their upcoming match fixtures. This functionally had been implemented through the provision of a [Fixtures] tab in the manage league page. The user interface objects used had been chosen with rapid data entry in mind.

Administrators are able to add new fixtures which redirects them to the screen seen in Figure 4.5.11.



Aston Uni	FT: 2:5	Bournemouth University
Aston Uni	29/9 13:36	Aston Uni

Aston Uni
vs
Aston Uni

10	17	
11	18	AM
12	19	PM

Mar	16	2015
Apr	17	2016
May	18	2017

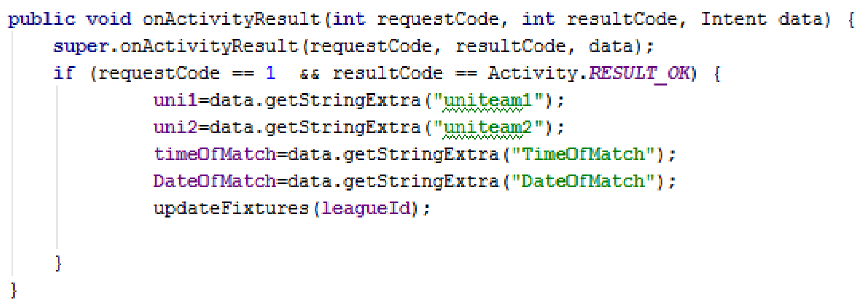
Figure 4.5.11 – Fixture Addition

The ‘Add Fixture’ Activity is put on top of the Activity stack using the ‘startActivityForResult’ method. The calling fragment which in this case is the Manage Fixtures page calls this method expecting to receive a result.

Once the administrator is happy that the match Fixture is correct they click the [Add Fixture] button. This executes an ‘onClick’ event method where the fixture information is stored inside an Intent and the result code is set inside a ‘setResult()’ method. The ‘setResult()’ method stores the result that the activity can return to its caller. ‘finish()’ is then called from within the Activity to alert the calling fragment to expect a result code.

The Add Fixtures Activity is comprised of two Spinner objects which have been pre-populated with University names, as well as a time and date picker. The pickers provide a responsive, clean and elegant solution for entering in a fixtures time and date, limiting the possibility of erroneous data being entered.

The ‘onActivityResult()’ method is called once the previously launched Activity has finished or has been exited. This method can be seen below (see Figure 4.5.12), firstly it ensures that the returned ‘resultCode’ is equal to ‘*RESULT_OK*’ if not then the method exits. However, if the ‘resultCode’ returned is equal to ‘*RESULT_OK*’ then the fixture data stored inside the intents bundle is obtained and the method for updating the Azure ‘fixtureTable’ is called.

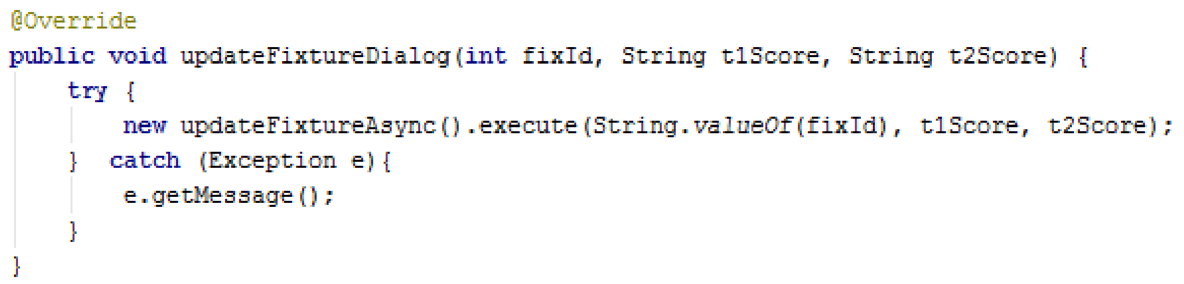


```
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (requestCode == 1  && resultCode == Activity.RESULT_OK) {
            uni1=data.getStringExtra("uniteam1");
            uni2=data.getStringExtra("uniteam2");
            timeOfMatch=data.getStringExtra("TimeOfMatch");
            DateOfMatch=data.getStringExtra("DateOfMatch");
            updateFixtures(leagueId);

    }
}
```

Figure 4.5.12 – handling the returned data (adding a fixture)

In addition, a fixture can be either deleted or updated to include the final match score. This has been achieved through the implementation of a popup menu. If the administrator decides to delete the fixture an ‘AsyncTask’ is executed and deletes the appropriate table column. If the fixture is ‘updated’ then a dialog is shown (see below), asking the administrator for the required information.

The dialog used to update a fixture with a match result is a custom dialog implementation. The implementation of this dialog is done through a Java class named ‘ManageFixtureDialog’. This class extends the base ‘DialogFragment’ class and has an interface declared within. In order to use the update fixture dialog inside the Fixtures tab the host Activity (‘ManageClubViewPager’) must implement the interface specified inside the ‘ManageFixtureDialog’ class. (The implementation can be seen below in Figure 4.5.13)



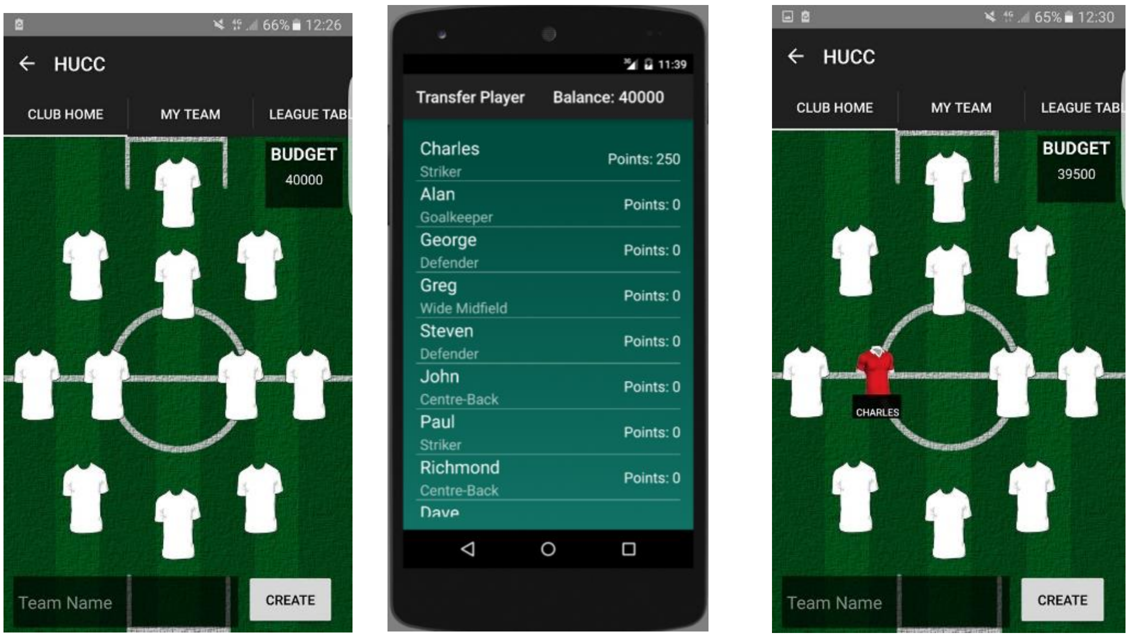
```
@Override
public void updateFixtureDialog(int fixId, String t1Score, String t2Score) {
    try {
        new updateFixtureAsync().execute(String.valueOf(fixId), t1Score, t2Score);
    }  catch (Exception e){
        e.getMessage();
    }
}
```

4.5.13 – implementing the ‘ManageFixtureDialog’ interface

The interface requires a method which is passed in the values of the fixtures id as well as the two scores (for both the home and away teams). The interface has been implemented so that it executes an AsyncTask to update the correct entry inside the fixtures table.

4.5.6 Team Creation Interface

When a User joins a new league they must first create a fantasy team in order to gain access to the league pages. The team creation layout is similar in design to the Manage Team layout. Presenting the User with 11 unpopulated playing shirts and an empty text field for a team name (Figure 4.5.14).

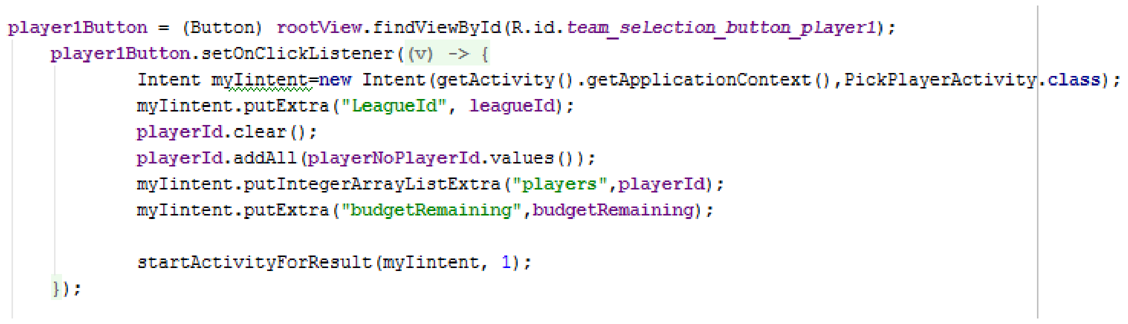


Charles Striker	Points: 250
Alan Goalkeeper	Points: 0
George Defender	Points: 0
Greg Wide Midfield	Points: 0
Steven Defender	Points: 0
John Centre-Back	Points: 0
Paul Striker	Points: 0
Richmond Centre-Back	Points: 0
Dave	

Figure 4.5.14 – Creating a team

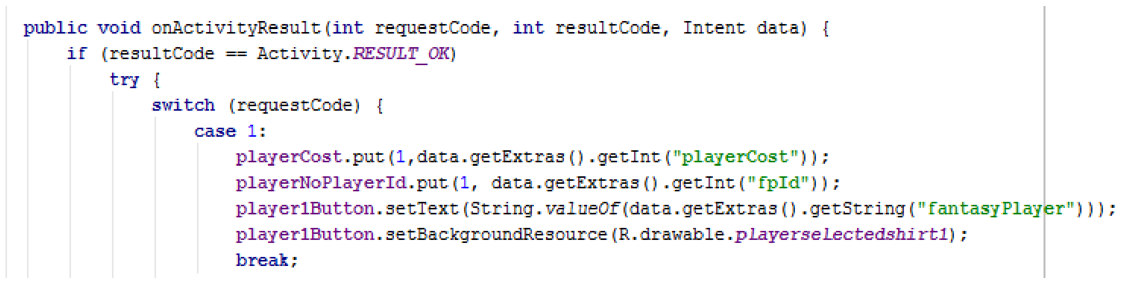
The User can select an unpopulated shirt to view the available players to choose from. If a player is selected and is within the users budget they can either be viewed (for additional player detail) or 'purchased' for the team. If a player is over the users budget they do not receive an option to 'purchase' and are automatically directed to the players' information screen. 'Purchased' players can be sold back to reclaim the costs and possibly fund a more expensive player. During this stage of the league joining process the user has infinite transfers. However, once the team has been committed and inserted into the 'userTeam' table the only way for a User to change their team is by being given 'transfer' tokens by a league administrator.

Identical to the Team Management class the Team Creation class implements the player shirts as Buttons. As seen in Figure 4.5.14 when a players' shirt is clicked an Intent is created and the leagues Id, purchased player Ids and remaining user budget are stored inside the intent and passed into the 'PickPlayerActivity' class. This calling activity is expecting a result, if a player has been 'purchased' from the 'PickPlayerActivity' class then a '*RESULT_OK'* will be returned and handled by the calling activities 'onActivityResult()' method (Figure 4.5.15).



```
player1Button = (Button) rootView.findViewById(R.id.team_selection_button_player1);
    player1Button.setOnClickListener((v) -> {
            Intent myIintent=new Intent(getActivity().getApplicationContext(),PickPlayerActivity.class);
            myIintent.putExtra("LeagueId", leagueId);
            playerId.clear();
            playerId.addAll(playerNoPlayerId.values());
            myIintent.putIntegerArrayListExtra("players",playerId);
            myIintent.putExtra("budgetRemaining",budgetRemaining);

            startActivityForResult(myIintent, 1);
    });
```

Figure 4.5.15 – selecting a player event listener



```
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (resultCode == Activity.RESULT_OK)
        try {
            switch (requestCode) {
                case 1:
                    playerCost.put(1,data.getExtras().getInt("playerCost"));
                    playerNoPlayerId.put(1, data.getExtras().getInt("fpId"));
                    player1Button.setText(String.valueOf(data.getExtras().getString("fantasyPlayer")));
                    player1Button.setBackgroundResource(R.drawable.playerselectedshirt1);
                    break;
```

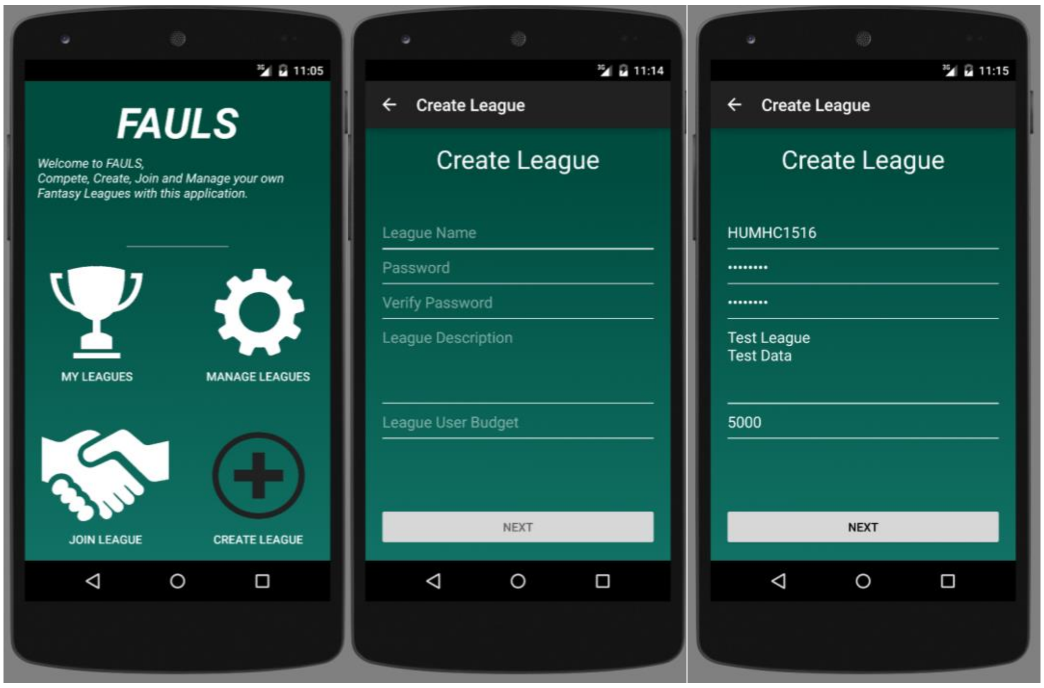
Figure 4.5.16 – Handling of a purchased player

If a player has been 'purchased' then the players cost will be subtracted from the users' budget and the [playerId] stored in an Integer 'HashMap'. The 'playerNoPlayerId' 'HashMap' is populated with keys from 1 to 11. The keys represent the player number and their associated values represent the players Id. It is the players Id that will be inserted into the 'userTeam' table.

4.5.7 League Creation

The application has been designed and implemented in a way such that any FAULs user can create their own league. League creation is managed through the [Create League] menu item.

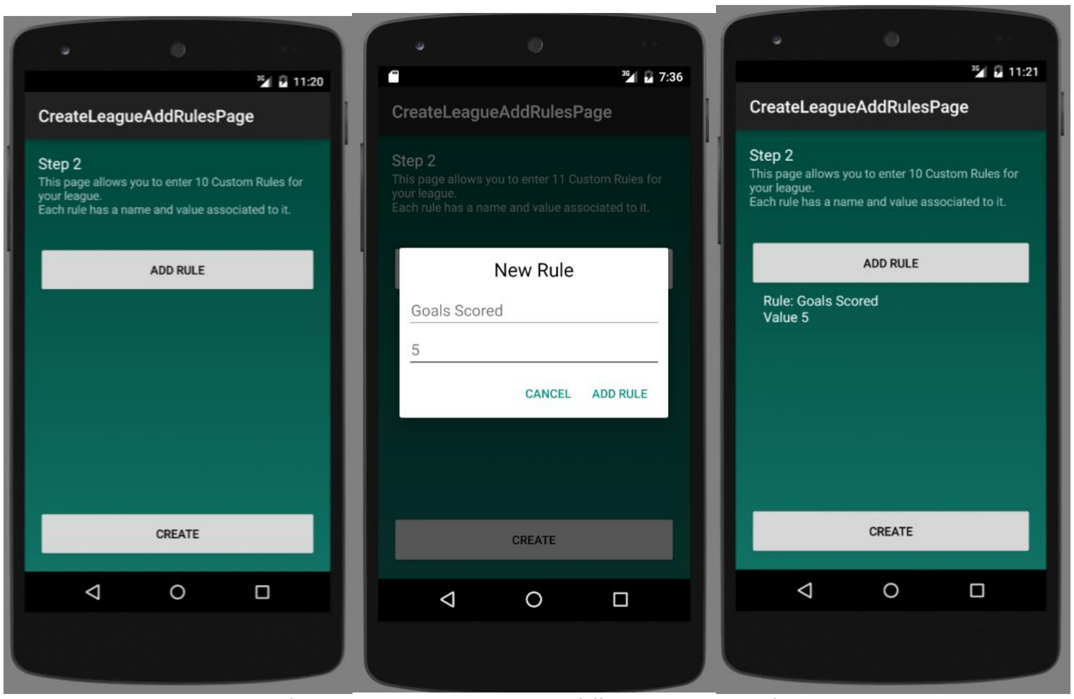
There are 2 essential steps to create a league. Firstly, information about the league is required. This is obtained through a simple 'league registration' styled menu. The required fields include a League Name, League Password (information required for Users to join the league), League Description (brief introduction to the leagues purpose) and a Budget. The League User Budget is the budget each User that joins the league will have to spend on forming their fantasy team. The steps involved can be seen in Figure 4.5.17 and Figure 4.5.18



HUMHC1516
••••••••
••••••••
Test League
Test Data
5000

Figure 4.5.17 – Step 1 Creating a league

The second step in creating a league is the addition of 1-11 league rules. As discussed league rules are used to determine a players point total post-game. Entering league rules has to be quick to ensure the User does not get bored mid league creation. If a User exits 'backs' out by this stage, then their league will be deleted from the 'leagueInformation' table.


CreateLeagueAddRulesPage
Step 2
This page allows you to enter 10 Custom Rules for your league.
Each rule has a name and value associated to it.
ADD RULE
CREATE
New Rule
Goals Scored
5
CANCEL
ADD RULE
Rule: Goals Scored
Value 5
Figure 4.5.18 – Step 2: Adding League Rules

Once the User has adding a number of league rules they can [Create] the league. This button invokes a method which inserts the league rules into the 'leagueRules' table as well as setting the [isAdmin] field to true inside the [userLeagues] table.

Once the league is created they will have access to the management portal. Here they can add new players, update rules, club information and fixtures.

4.5.8 Calculating Team Points

Calculating individual player point values is an important step in delivering a functioning fantasy sport application. However, equally important and a more complex requirement is calculating all participating users team scores when required.

Each user has a team of 11 players including a captain and vice-captain. The current implementation provides point bonuses for each teams' captain and vice-captain. This decision was made to help drive the competition allowing users to tactically select their captain and vice-captain at any given point.

As previously discussed, fantasy player point totals are updated when an administrator makes adjustments to their rule occurrence values. Point changes are added onto the fantasy player's [rulePointIncement] column. However, the point change is not committed to a fantasy profile until the administrator clicks [Calculate Weekly Points] on the Members tab.

Team captains generate double the points and Vice-Captains 1.5x the points for the value stored inside their [rulePointIncrement] column. This point bonus is applied to a user's team once the administrator updates the team scores for that time period.

Implementation

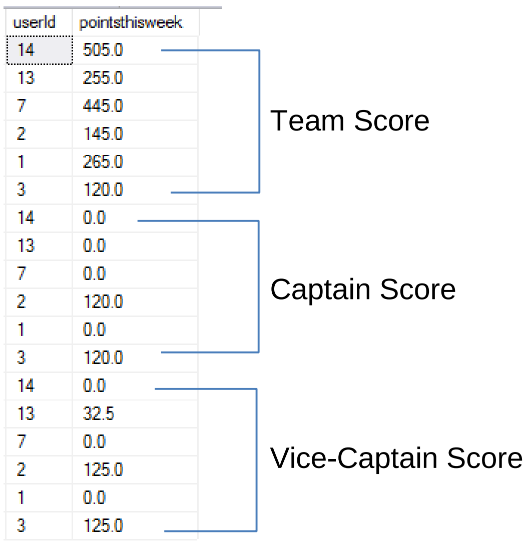
The fundamental logic for calculating the score changes for each fantasy team in the league for a given time period is achieved by joining the 'fantasyPlayers' and 'userTeam' tables together. For this, the Ids stored within the [playerId] and [captain]/[vicecaptain] columns inside the 'userTeam' table are used to obtain the appropriate fantasy players [rulePointIncrement] value. The 11 [rulePointIncrement] values (and bonus point calculations) are added together and used to update the ‘userTeams’ [userPoints] column.

Initially the applications implementation of this functionality involved executing a ‘get’ statement against the ‘fantasyPlayer’ table 13 times. The executed statement returned an instance of a ‘fantasyPlayer’. The [rulePointIncrement] value would then be obtained through the returned ‘fantasyPlayer’ instance and added onto a locally declared ‘teamPointsThisWeek’ integer value. Once all of the ‘userTeam’ [playerId] columns have been iterated through the ‘teamPointsThisWeek’ value is added onto the ‘userTeams’ [userPoints] column.

The above process was repeated for each team present in the current league. This resulted in the execution of many of the same queries making this particular implementation slow, unreliable and inefficient. As a result, an alternative method was required and a custom API was created to handle the table joins and point calculations Server side.

‘CalculateUserPoints’ API
The API invoked by the mobile client instantaneously adds up all the appropriate ‘fantasyPlayer’ [rulePointIncrement] columns for each team. The API script uses the SQL SUM function to add the [rulePointIncrement] columns where the ‘fantasyPlayer’ Id is ‘IN’ one of the 11 [playerId] ‘userTeam’ columns. The SUM total is then grouped together on the [userId] column.

The result of the calculation can be seen in the example table below (Figure 4.5.19).


Team Score
Captain Score
Vice-Captain Score

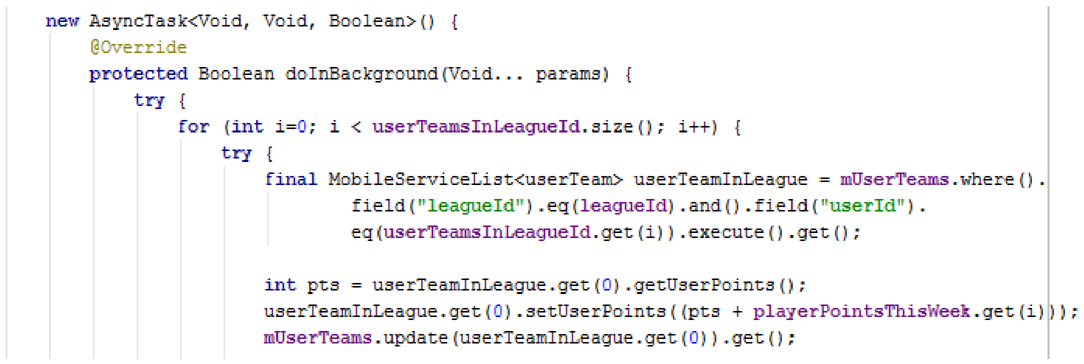
userId	pointsthisweek
14	505.0
13	255.0
7	445.0
2	145.0
1	265.0
3	120.0
14	0.0
13	0.0
7	0.0
2	120.0
1	0.0
3	120.0
14	0.0
13	32.5
7	0.0
2	125.0
1	0.0
3	125.0

Figure 4.5.19 – Team point calculation result (SQL Query)

The captain and vice-captain ‘bonus’ points are returned as separate rows and are not ‘summed’ alongside the ‘team score’ point total. The API returns the above table results back to the client where it can be deserialised into a JSON Object.

The JSON Object is converted into a JSON Array. The JSON Array is then iterated through. With each iteration the returned [pointsthisweekvalue] and [userId] value is read and stored inside an ‘ArrayList’ object. Once the JSON Array has been completely iterated through a new background thread is executed using the ‘AsyncTask’ method.

Inside this ‘AsyncTask’ method each user has their ‘userTeam’ [userPoints] property updated to include the new points for the given week and then updated inside the database. The code for how this is achieved can be seen below in Figure 4.5.20.



```
new AsyncTask<Void, Void, Boolean>() {
    @Override
    protected Boolean doInBackground(Void... params) {
        try {
            for (int i=0; i < userTeamsInLeagueId.size(); i++) {
                try {
                    final MobileServiceList<userTeam> userTeamInLeague = mUserTeams.where().
                            field("leagueId").eq(leagueId).and().field("userId").
                            eq(userTeamsInLeagueId.get(i)).execute().get();

                    int pts = userTeamInLeague.get(0).getUserPoints();
                    userTeamInLeague.get(0).setUserPoints((pts + playerPointsThisWeek.get(i)));
                    mUserTeams.update(userTeamInLeague.get(0)).get();
```

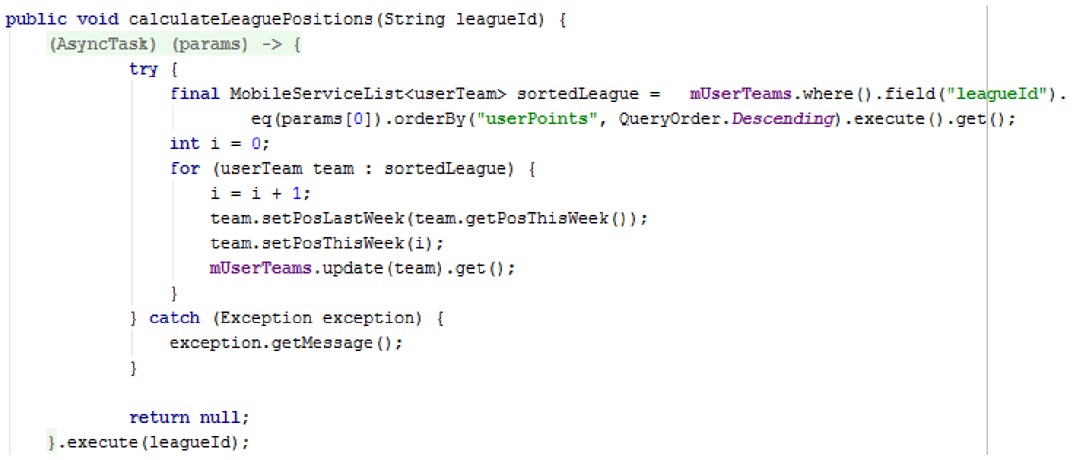
Figure 4.5.20 – Inserting the team scores

4.8.9 Calculating Team Positions

Once the scores for each team have been updated an ‘onPostExecute()’ method is called. This method invokes the ‘calculateLeaguePositions’ method which is responsible for updating the team ranking positions.

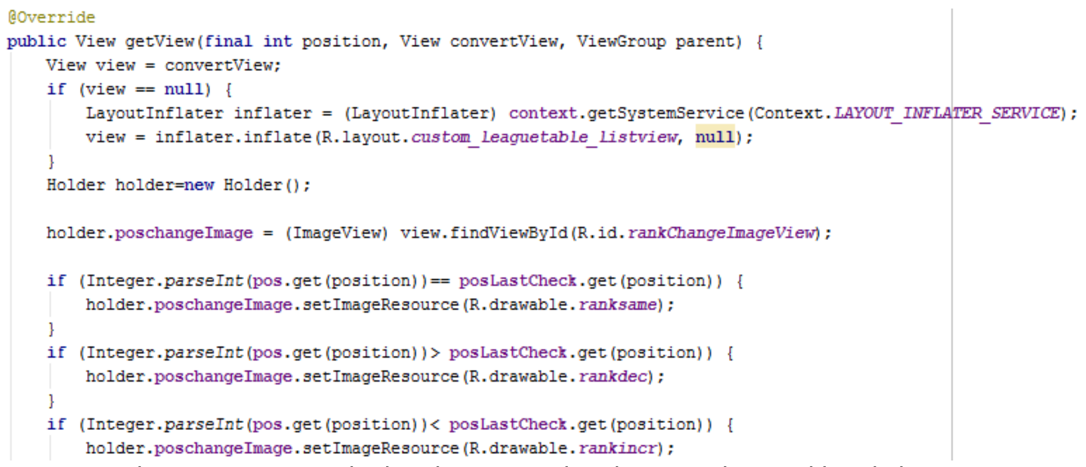
This update is necessary as the information on ranking change is required by the league tables custom ‘ListView’ adapter implementation. The logic for calculating a teams’ rank change has been made possible due to the databases design which stores the values for [positionThisWeek] and [positionLastWeek] inside the ‘userTeam’ table.

The code below (Figure 4.5.21 and 4.5.22) illustrates how the ranks (and positional change) are calculated through querying the mobile service ‘userTeam’ table and implementing some basic logical checks.



```
public void calculateLeaguePositions(String leagueId) {
    (AsyncTask) (params) -> {
            try {
                final MobileServiceList<userTeam> sortedLeague =   mUserTeams.where().field("leagueId").
                        eq(params[0]).orderBy("userPoints", QueryOrder.Descending).execute().get();
                int i = 0;
                for (userTeam team : sortedLeague) {
                    i = i + 1;
                    team.setPosLastWeek(team.getPosThisWeek());
                    team.setPosThisWeek(i);
                    mUserTeams.update(team).get();
                }
            } catch (Exception exception) {
                exception.getMessage();
            }

            return null;
    }.execute(leagueId);
```

Figure 4.5.21. – calculating new league positions



```
@Override
public View getView(final int position, View convertView, ViewGroup parent) {
    View view = convertView;
    if (view == null) {
        LayoutInflater inflater = (LayoutInflater) context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
        view = inflater.inflate(R.layout.custom_leaguetable_listview, null);
    }
    Holder holder=new Holder();

    holder.poschangeImage = (ImageView) view.findViewById(R.id.rankChangeImageView);

    if (Integer.parseInt(pos.get(position))== posLastCheck.get(position)) {
        holder.poschangeImage.setImageResource(R.drawable.ranksame);
    }
    if (Integer.parseInt(pos.get(position))> posLastCheck.get(position)) {
        holder.poschangeImage.setImageResource(R.drawable.rankdec);
    }
    if (Integer.parseInt(pos.get(position))< posLastCheck.get(position)) {
        holder.poschangeImage.setImageResource(R.drawable.rankincr);
```

Figure 4.5.22 – assigning the appropriate image using positional change

5 Evaluation

5.1 Project Achievements

The success of this project can be predominantly assessed by the evaluation and analysis of the aims and objectives considered relevant to the projects successful delivery.

Since the projects original conception the project aims and objectives have evolved and have been refined. This is a result of not having an exact specification from HUMHC prior to the project beginning and following an iterative design methodology. Having members of the Hull University Athletic Union (predominately HUMHC) test the application at key stages has allowed for the projects development to grow from its 'loosely' defined initial specification into a desired product for the client.

The following section shall investigate any differences between the original stated objectives and the delivered functionality / product.

5.1.1 Objective 1 – Develop the Fantasy Leagues Front End

The aim of this objective was to provide users an intuitive interface on their Android devices that would allow them to perform a number of tasks. The application had to provide an interface for managing and participating in a Fantasy League and the experience had to supersede that being offered by the current system being used by the Hull University Men's Hockey Club. To achieve this the functionality had to allow users to create, manage and participate in fantasy leagues.

As a result of extensive background research into existing products the applications user interface design was able to draw inspiration as to what provides a good and bad user experience for fantasy sports.

The greatest achievement for this objective is the Team Page (Figure 5.1) which displays the users team and all relevant information in a bright, visually pleasing manner. The page offers all the required functionality of setting captain/vice-captains, transferring players and viewing individual player performances.

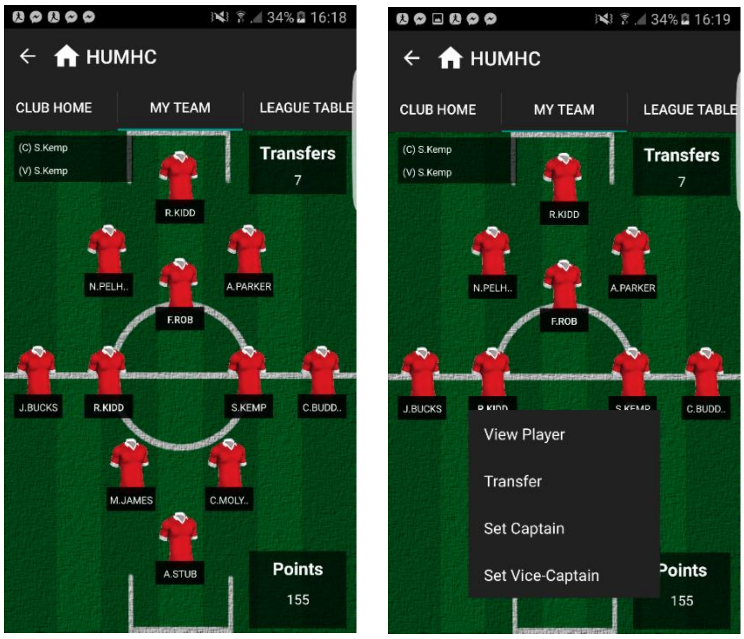


Figure 5.1 – Team View Page

The user interface was designed so there is a level of coherence and similarity between the Manage and the User League interfaces. Careful consideration had been placed into the flow of the application and through extensive user testing this objective can be considered met.

Sub-Objective	**Details**	**Completed**
User Login and Register Functionality	Allow users to create an account and login to the application.	Yes
Create League and Join League functionality	Allow users to create their own leagues for their AU club. Users can then join these leagues.	Yes
Add Player Update Player Create Team Manage Team	League administrators can add 'fantasy' players to their leagues as well as update their player statistics. Users can create a team from the pool of available players. They can transfer players as well as set a Captain / Vice-Captain for bonus points.	Yes

5.1.2 Objective 2 – Develop the 'Fantasy Leagues' Web Service

The project required a Cloud Service that would facilitate future development onto a number of different platforms (iOS, Windows and a Web Application). For this project the Microsoft Azure service was used as it fulfilled the requirement stated above.

This objective had been divided into two sub-objectives.

Firstly, the application required a database that would store all the information considered to be required for the application to deliver its desired functionality. Data to be stored included information relating to user details, fantasy players, user teams, league information, league rules and match fixtures. This information was stored in logically structured tables with appropriate Primary and Foreign keys being used. The developers experience with databases and the time spent designing the databases structure had ensured that this sub-objective was achieved.

The second sub-objective was to create and implement a number of 'Custom APIs'. This second sub-objective was added later in the project lifecycle because a need was identified for faster table join operations as well as calculations. These APIs can be reused on any platform supported by the Azure service. In addition, the APIs facilitated the requirement of calculating team point totals. Point calculation had been attempted through the application but had proved to be very slow and performance intensive, this is what led to the need of a better, faster solution.

Both of these sub-objectives have been fulfilled. The database is being hosted on a Microsoft server and is affiliated to the FAULs App Service. This will allow for future development onto additional platforms through the use of a single App Service. The database has been designed so that all required information is stored and again, can be reused by a number of different platforms.

Sub-Objective	Details	Completed
Create FAULs Database	Logically structured Database for storing all data pertinent to the FAULs application.	Yes
Creation of Custom APIs	Create a number of Custom APIs that can be invoked by the application. Processing done Server side rather than client to improve performance.	Yes

5.1.3 Objective 3 – Implement an enhanced 'Club Page'

This objectives primary aim was to help differentiate the application from the fantasy sporting applications already available on the market. This was to be achieved by giving league administrators the ability to manage content on their club home page as well as displaying club fixtures.

This objective had been divided into 3 sub-objectives. 2 of the 3 sub-objectives have been successfully implemented to some extent.

Firstly, customising the club page had been fully achieved. With a number of text-fields being present to display whatever information a club administrator desires.

Secondly, a fixtures page has been implemented with all the required functionality. As it stands the fixtures page allows administrators to add and delete new fixtures. This is done through an intuitive interface. Fixtures include team names a match time and match date. Fixtures can also be updated with the final time score, fulfilling the requirement of providing 'results and game information'. However, future development would like to extend 'game information' to include which players scored. This could be implemented through an 'ExpandableListView' object.

Finally, an initial sub-objective was to allow club administrators to change the applications background colour to match their clubs' colours. This feature has not been implemented, as when testing the application with a range of different colours a number of those used lowered the quality of the applications design. A decision was made that the application will simply use one uniform colour as its background. Future work will explore different proposals for providing users a more personal experience, for now Club Colours is not the solution.

Sub-Objective	Details	Completed
Fixtures Page	Display and manage match fixtures	Yes
Customise Club Page	Allow Administrators to add important club information to their homepage.	Yes
Club Colours.	Change league page colours to those associated with their club.	No

5.1.4 Objective 4 – Implement Functionality to Merge User Accounts with fantasy players.

The ability for league administrators to merge participating league user accounts with an associated fantasy player had been successfully implemented. Users can now view their own playing statistics and other users can view which teams belong to which players in their teams.

The user is able to access their personal player profile easily through the users leagues [User] tab. Whilst other users can explore their teammates performance through the league table tab. Selecting a team will give the option to view 'player profile'

This was a comparatively small objective in comparison to those above but does enhance the application by allowing its users to compare their playing performances in a competitive manner with fellow club members.

5.1.5 Bonus Objective 5 – Design a Web Application for Data Entry.

Due to the time limitations of this project a decision was made that a 'successful' delivery of this bonus objective may compromise that of the mobile applications.

As a result of the ambitious nature of both this bonus objective and the mobile applications the web application for data entry has been put on hold for future development. This decision has allowed for a successful implementation of the application and has met the overall project aim and objectives.

5.2 Further Development

5.2.1 Club Page Enhancements

There is further potential for additional features to be incorporated on the club page screen that would provide an even greater level of functionality.

Implementing more customisability to the 'Club Page' will be the main focus of future development. The possibility of including a Map object to pinpoint training and social locations would be quite beneficial to new members joining the club. Additionally, providing Clubs 'placeholder' objects where they can store sponsor graphics or their own club insignias would make for a more 'personalised' experience.

5.2.2 Fantasy Player Profiles

A future enhancement of the fantasy player pages would be to include images of the player. These images could be uploaded by an administrator and stored in Azure using Blob storage.

5.2.3 Fixtures Page

The current fixture page delivers on all of its requirements outlined in the objectives section. However, a recommended addition would be to replace the existing 'ListView' with an 'ExpandableListView'. This would display the match scorers and/or man of the match details.

This functionality had only been considered in the later stages of the projects lifecycle. It was not implemented due to timescales and the risk of breaking existing functionality. It is expected that this enhancement would require an additional table to be created to store match fixture ids, player ids and goal count. (i.e. Game | Player Scored | Number of Goals).

5.2.4 Additional Device Support

With a large number of Android devices available on the market and the time constraints of this project it had not been possible to test and ensure the full support of the entire range of available devices. The initial project decision was to target devices with screen sizes varying between 4.95 to 5.5" and pixel densities between xhdpi and xxhdpi.

Future development would seek to provide layouts and drawables to support the entire range of mainstream android devices with the varying screen sizes and pixel densities.

5.2.5 iOS Application

The application has been successfully developed for Android devices. However, future development will seek to deliver a native iOS version. The Mobile Service is capable of supporting this additional platform and will increase the applications potential user base. iOS has been selected as the primary focus for future development due to the market share of as identified during background research.

6 Conclusion

The projects original aim was to deliver a mobile application to replace the existing 'fantasy hockey' system for the Hull University Men's Hockey Team (HUMHC).

This aim was the driving factor behind many of the design decisions and helped prioritise the implementation of the applications functionality. However, the delivered product provides users a much more powerful service than first envisioned. The application has been designed and implemented so it supports a much wider array of sports increasing its real-world potential value.

The application has delivered the functionality for HUMHC to manage and participate in their own fantasy league games. Additionally, any University team to sign-up, create a league and manage it in a manner that suits their needs. Whilst the applications current design has focused on 11 player team sports it does not prevent sports with fewer or greater players using it. Future development will look into implementing league creation with variable team sizes.

The FAULS mobile service and database tables have been created to facilitate the requirements outlined by the objectives. Special consideration has been put into the design of the database to ensure it will support future development and maintain data integrity through continuous use.

Considering all that has been discussed and following from the evaluation of the final product the project can be considered a success. With all core objectives being achieved the application has delivered on the functionality it had promised at the commencement of the year.

Appendix A: Task List

Id	Task Name	Description	Duration (days)
1	Design UI	Design the theme that will be applied consistently throughout the application across all screens.	5
2	Design database schema	Design database structure, considering requirements (including: joins, normalisation, etc.). The overall structure may evolve as additional data may be required.	5
2	Develop and implement user login authentication	Implement the functionality for registering users, storing user login details and to authenticate (login) returning users.	5
3	Develop and implement 'Create League'	Implement the functionality for an Administrator to create a league page, with a randomized unique league ID.	5
4	Develop and implement 'Join' League	Implement the functionality for Users to join an existing league.	5
5	Implement functionality to add Players.	Implement functionality for Administrators to add 'Fantasy' players to their league. With relevant player information.	6
6	Implement functionality to manage Players.	Implement the ability for Administrators to manage player scores. (Increment Goals Scored, Goals Conceded, Games Played, etc.).	6
7	Implement functionality for Administrators to set scoring rules.	Implement functionality for Administrators to determine league rules. i.e. User spending limit, Player values, point distribution, number of players in a team, etc.	6
8	Interim report (Deliverable)	DELIVERABLE: Write the interim report deliverable	14
9	Implement functionality for Users to create a team	Implement ability for Users to spend virtual funds on adding players to their fantasy team of 'x' number of players.	12
10	Implement League table screen.	Implement a table for users to view their teams ranking against other users in their league.	5
11	Implement 'Club Page'	Implement the clubs home screen. Administrators should be able to edit the meta data here. I.e. include club information (training times, upcoming events, upcoming fixtures).	12
12	Implement 'Merge profiles' functionality	Implement the ability for Users to request to merge their account profile with their respective 'Fantasy' player profile.	6
13	*Bonus Objective: Create basic web interface for data entry.*	*Bonus Objective: Create a basic web interface for league administrators to update player statistics. Not required to meet the aim of this project.*	*21*
14	Final Report (Deliverable)	DELIVERABLE: Write the final report deliverable	20

Appendix B: Original Time Plan

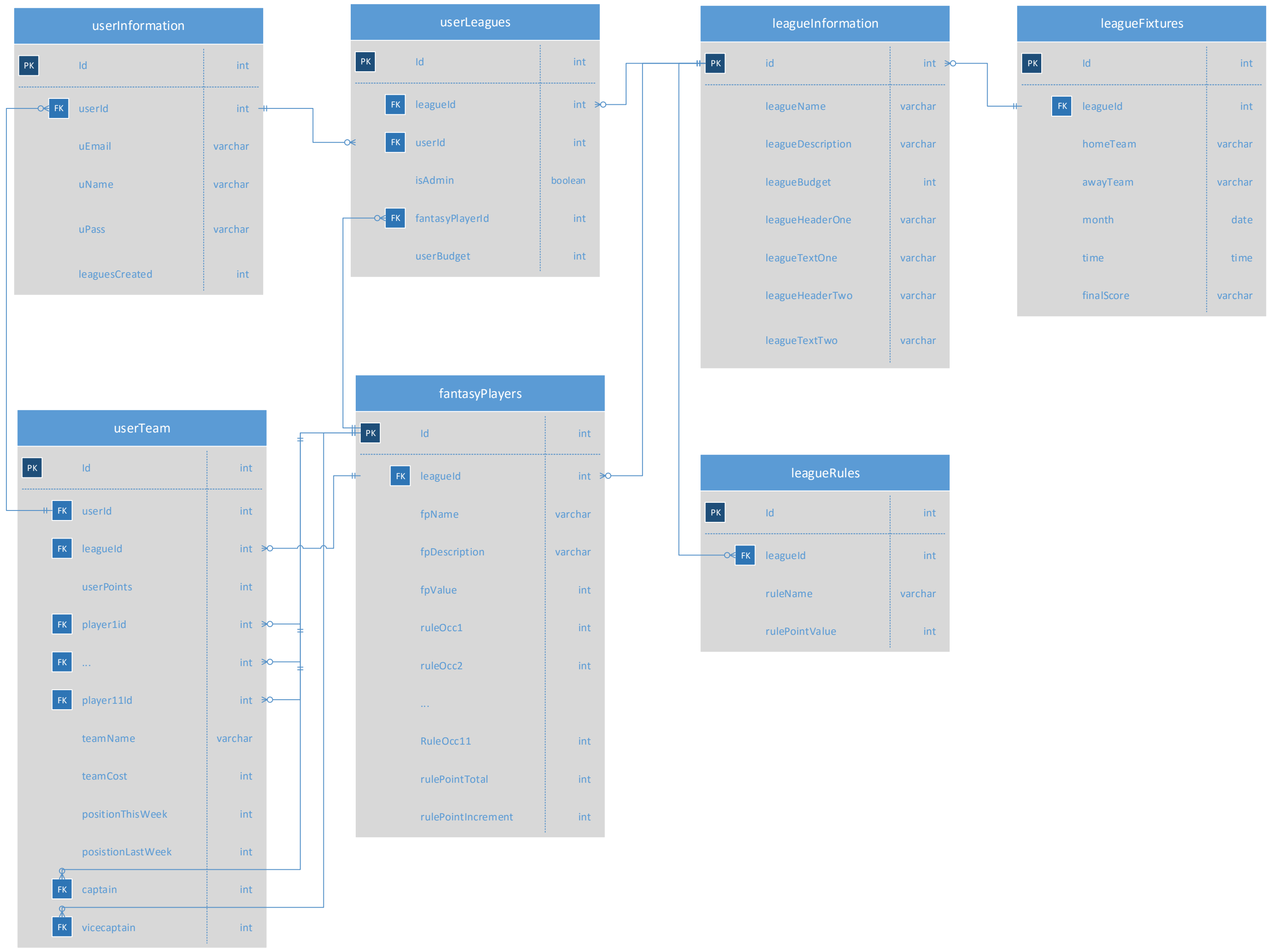
			University Calendar Weeks (Semester 1)															
#	Duration (D)	Task Name	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1		**Initial Specification**	🟩															
2	5	Market Research		🟩	🟩													
3	5	Platform Research			🟩	🟩												
4	7	Design Application Work Flow					🟩											
5	7	Design Database Structure					🟩	🟩										
6	7	Prototype UI Designs						🟩	🟩									
7	0	Reseach / Design Phase Completed.								**Milestone**						**TIME BUFFER**		
8	4	Implement 'Login / Register' Functionality								🟩								
9	4	Implement 'Create League' Functionality									🟩							
10	4	Implement 'Join League' Functionality									🟩	🟩						
11	7	Implement 'Add Fantasy Players' Functionality										🟩	🟩					
12	7	Implement 'Manage Players' Functionality											🟩	🟩				
13	0	Multiple Project Deadlines (reduced resources)												🟩	🟩			
14	5	Implement 'Set Rules' Functionality																
15	14	Interim Report															🟥	🟥 **Deliverable**

			University Calendar Weeks (Semester 2)																
#	Duration (D)	Task Name	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
16	12	Implement 'Create Team'	🟩																
17	4	Implement 'League Table'		🟩	🟩														
18	12	Implement 'Club Page' Functionality				🟩													
19	0	Core Project Aim satified				**Milestone**												**TIME BUFFER**	
20	5	User Testing					🟩												
21	10	Consider / Implement Post Testing Feedback						🟩											
22	12	Implement 'Merge User / Players' Functionality							🟩	🟩									
23	14	Develop a Web Application to edit League Meta data									🟩	🟩							
24	7	Code Testing											🟩	🟩					
25	0	Deliver 'Fantasy Leagues: AU Edition' Application													**Milestone**				
16	7	Document Functionality													🟩				
27	14	Final Report														🟥	🟥 **Deliverable**		

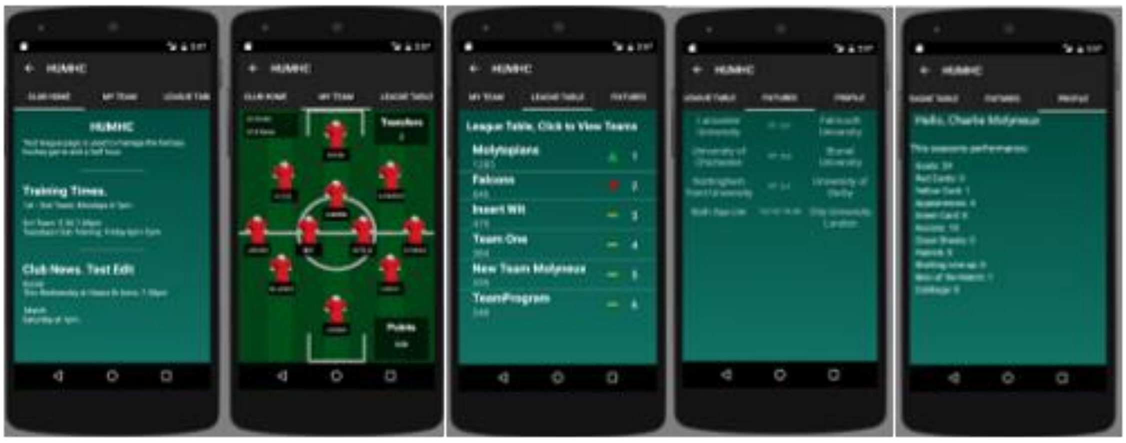
Appendix C: Amended Time Plan (For 2nd Semester)

			University Calendar Weeks (Semester 2)																
#	Duration (D)	Task Name	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
1	14	Interim Report	🟩	🟥 Deliverable															
2	5	Complete Login Functionality		🟩	🟩	🟩													
3	5	Complete 'Create League' Functionality		🟩	🟩	🟩													
4	3	Complete 'Join League' Functionality				Milestone												T	
5	5	Implement 'Add Fantasy Players' Functionality					🟩	🟩										I	
6	5	Implement 'Manage Players' Functionality					🟩	🟩	🟩									M	
7	7	Implement 'Create Team' Functionality						🟩	🟩	🟩								E	
8	5	Implement 'League Table' Functionality							🟩	🟩	🟩								
9	5	Implement 'Club Page' Functionality								🟩	🟩	🟩						B	
10	0	Core Project Aim satified											Milestone					U	
11	5	Enhance UI										🟩	🟩	🟩				F	
12	5	Implement 'Merge User / Players' Functionality											🟩	🟩				F	
13	21	Develop a Web Application to edit League Meta data										🟩	🟩	🟩				E	
14	0	Deliver 'Fantasy Leagues: AU Edition' Application												🟥 Deliverable				R	
15	20	Final Report												🟥	🟥	🟥	🟥 Deliverab		
16	15	Code Testing			🟩		🟩			🟩			🟩		🟩				
17	10	User Testing					🟩			🟩		🟩		🟩					

Appendix D: Database Structure

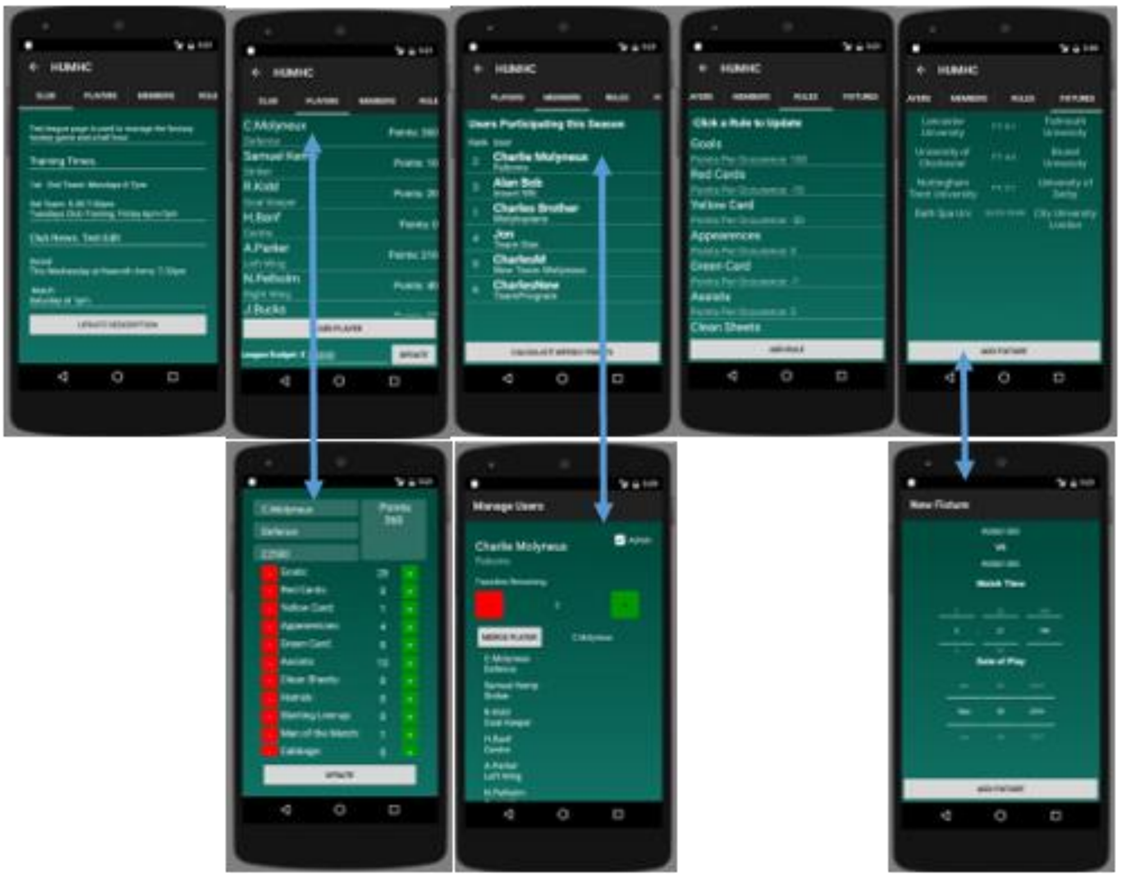


Appendix E: User and Management League Interface designs (Finalised)



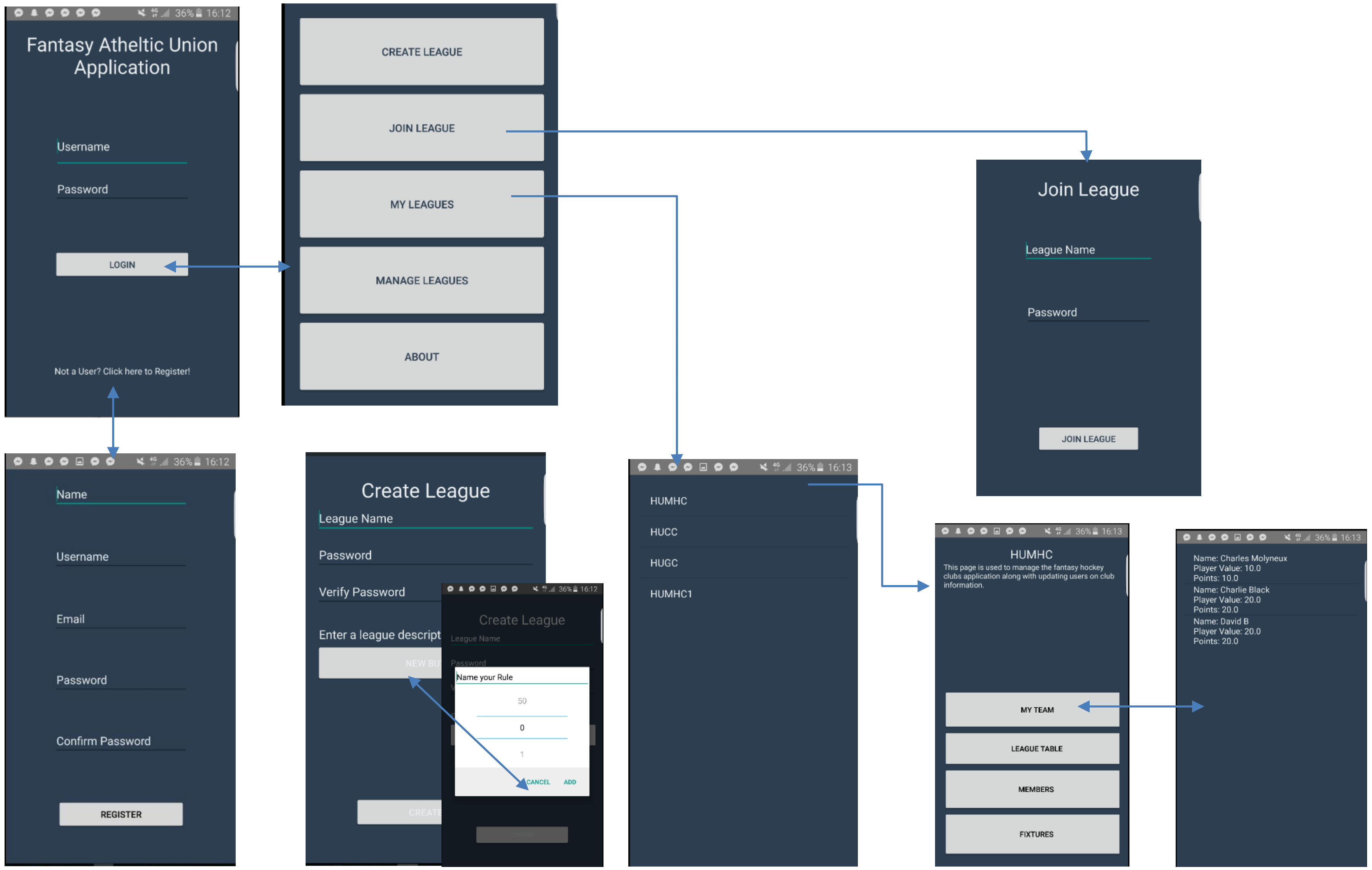
(Above: User League View)

(Below: Management League View)



64

Appendix F: Prototype Designs (Including Basic Application flow)



This page is used to manage the fantasy hockey clubs application along with updating users on club information.

Name: Charles Molyneux
Player Value: 10.0
Points: 10.0
Name: Charlie Black
Player Value: 20.0
Points: 20.0
Name: David B
Player Value: 20.0
Points: 20.0

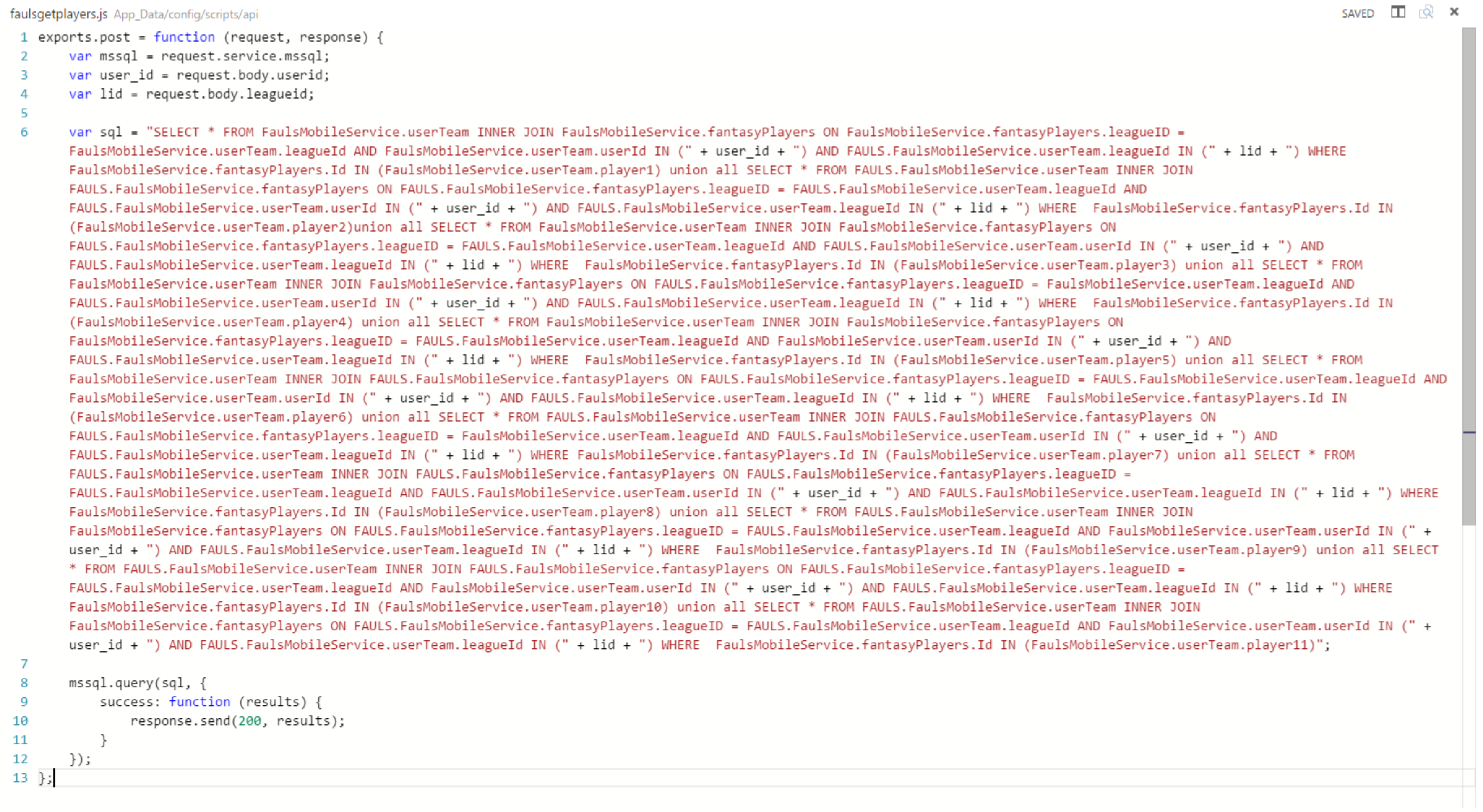
Appendix G: Sample Acceptance Tests

Page	Test	Expectation	Pass
Register Activity	Registering a new account	All fields are required before an account can be registered. Email and Password verifcation works.	P
Login Activity	Logging into the application with the registered account.	Connected to the FAULS application and redirected to the main menu.	P
Login Activity	Saving and Unsaving User Credentials	Checking / Unchecking the 'Saved Credentials' functions as expected.	P
Login Activity	Prevcent 'Login' functionality if required fields are blank	Button is locked until user enters details into the username and password fields	P
Create League	Creating a new league (1)	Once the required information has been provided the user is directed to the [Add Rules Page]	P
Create League	Creating a new league (2)	User can add upto 11 rules for their league. Cap is in place.	P
Create League	Creating a new league (3)	When a user is ready the league can be created. This adds a new league into the mobile service table and sets the user to an admin.	P
Manage Leagues	Updating Club Information	Club informtation is updated correctly	P
Manage Leagues	Updating Budget	League budget is updated correctly.	P
Manage Leagues	Calculate Weekly Point	Points are correctly calculated for each team present in the league.	P
Manage Leagues	Calculate Player Points.	Players have their points correctly calculated when their statistics change.	P
Manage Leagues	Merge a User account with a players profile	User correctly has their profile merged. Profile can be viewed in the User Leagues activity.	P
Manage Leagues	Updating a players transfer count	Players have their transfer count incremented in the Mobile Service	P
Manage Leagues	Creating a new fixture	Fixtures are added into the Mobile Service and displayed correctly.	P
Manage Leagues	Updating a Fixture	Fixutures score is added and the ListView updates	P
Manage Leagues	Adding a player		P
Player Management	Updating a player	The correct league rules are assigned to a player and the correct rule occurrence for the player are matched accordingly.	P
User Leagues	Creating a team	User is redirected to the team creation page if first time viewing the league. Team budget updates correctly and inserted into the mobile service once the user has finished.	P
User Leagues	Manage team	Players are transferrable if within budget and user has transfers	P
User Leagues	Manage team	Captain / Vcaptains can be set and changed	P
User Leagues	Manage team	If a user has 0 transfers the option is removed.	P
User Leagues	League table	Positional changes are calculated and handled by the listview	P
User Leagues	League table	Users can view user profiles and user teams	P

Appendix H: Initial Sample Acceptance Tests (Draft)

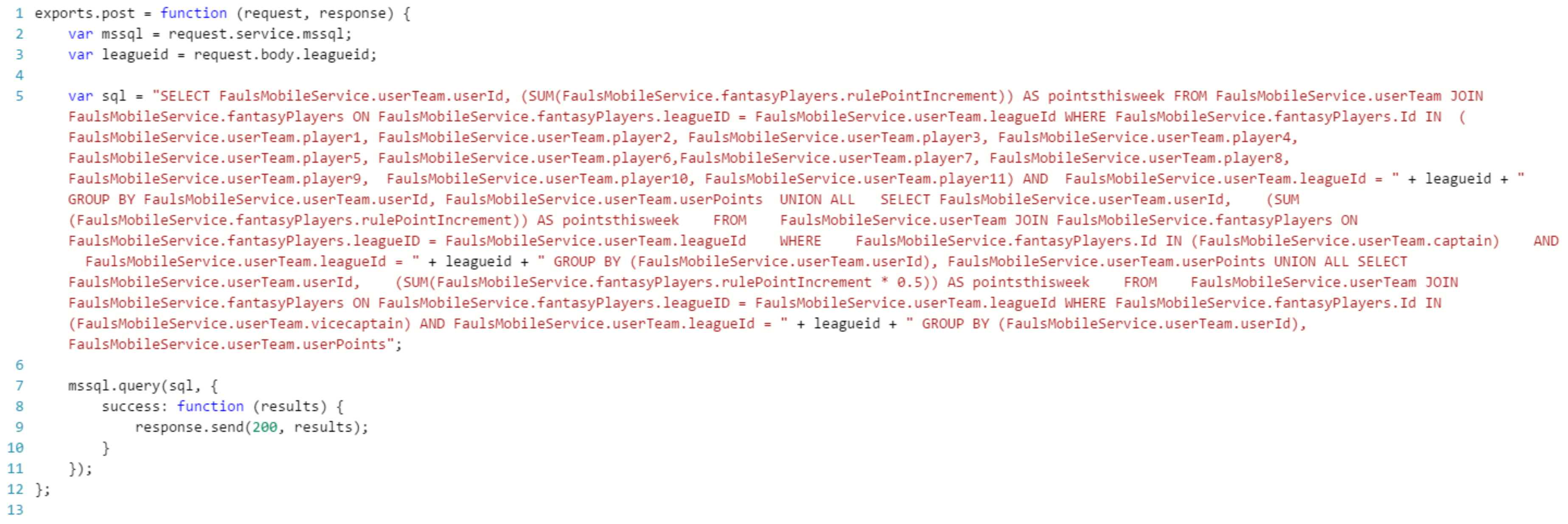
	Activity Page									
Tests	Login Screen	Register Screen	Create League	Join League	Add Player	Update Player	Create Team	Manage Team	Club Page Management	**Comments**
String Tests										
Handles Erronous Characters being entered into text fields										
Email Validation/Verification										
Username Validatation/Verification										
Password Validation/Verification										
Numbers in a Textbox										
Handles Null Values										
Rule point value correctly converted to numeric										
Logic Tests										
Points calculated correctly										
Value ranges / Point ranges set correctly										
User budget updated correctly										
Users cannot exceed their budget.										
Changing rule point value will automatically update all players associated.										
General Tests										
Application flow works as expected										
Rules are added to the correct leagues										
Rules are associated to the correct Players										
Player Addition Works										
Player Deletion Works										
Player Update Works										
correct information is retrieved from database										
Users can join leagues										
Users can leave leagues										
Users link with the correct players.										
Users cannot create more than 3 leagues.										

Appendix I: Example API 1 – ‘faulsgetplayers’ API



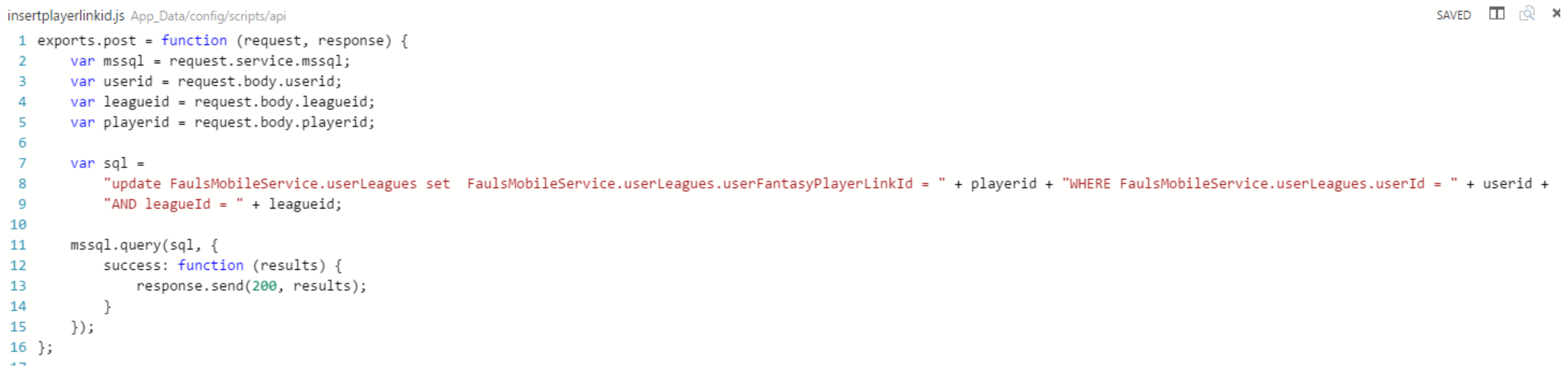
```
exports.post = function (request, response) {
    var mssql = request.service.mssql;
    var user_id = request.body.userid;
    var lid = request.body.leagueid;

    var sql = "SELECT * FROM FaulsMobileService.userTeam INNER JOIN FaulsMobileService.fantasyPlayers ON FaulsMobileService.fantasyPlayers.leagueID =
    FaulsMobileService.userTeam.leagueId AND FaulsMobileService.userTeam.userId IN (" + user_id + ") AND FAULS.FaulsMobileService.userTeam.leagueId IN (" + lid + ") WHERE
    FaulsMobileService.fantasyPlayers.Id IN (FaulsMobileService.userTeam.player1) union all SELECT * FROM FAULS.FaulsMobileService.userTeam INNER JOIN
    FAULS.FaulsMobileService.fantasyPlayers ON FAULS.FaulsMobileService.fantasyPlayers.leagueID = FAULS.FaulsMobileService.userTeam.leagueId AND
    FAULS.FaulsMobileService.userTeam.userId IN (" + user_id + ") AND FAULS.FaulsMobileService.userTeam.leagueId IN (" + lid + ") WHERE  FaulsMobileService.fantasyPlayers.Id IN
    (FaulsMobileService.userTeam.player2)union all SELECT * FROM FaulsMobileService.userTeam INNER JOIN FaulsMobileService.fantasyPlayers ON
    FAULS.FaulsMobileService.fantasyPlayers.leagueID = FAULS.FaulsMobileService.userTeam.leagueId AND FAULS.FaulsMobileService.userTeam.userId IN (" + user_id + ") AND
    FAULS.FaulsMobileService.userTeam.leagueId IN (" + lid + ") WHERE  FaulsMobileService.fantasyPlayers.Id IN (FaulsMobileService.userTeam.player3) union all SELECT * FROM
    FaulsMobileService.userTeam INNER JOIN FaulsMobileService.fantasyPlayers ON FAULS.FaulsMobileService.fantasyPlayers.leagueID = FaulsMobileService.userTeam.leagueId AND
    FAULS.FaulsMobileService.userTeam.userId IN (" + user_id + ") AND FAULS.FaulsMobileService.userTeam.leagueId IN (" + lid + ") WHERE  FaulsMobileService.fantasyPlayers.Id IN
    (FaulsMobileService.userTeam.player4) union all SELECT * FROM FaulsMobileService.userTeam INNER JOIN FaulsMobileService.fantasyPlayers ON
    FaulsMobileService.fantasyPlayers.leagueID = FAULS.FaulsMobileService.userTeam.leagueId AND FaulsMobileService.userTeam.userId IN (" + user_id + ") AND
    FAULS.FaulsMobileService.userTeam.leagueId IN (" + lid + ") WHERE  FaulsMobileService.fantasyPlayers.Id IN (FaulsMobileService.userTeam.player5) union all SELECT * FROM
    FaulsMobileService.userTeam INNER JOIN FAULS.FaulsMobileService.fantasyPlayers ON FAULS.FaulsMobileService.fantasyPlayers.leagueID = FAULS.FaulsMobileService.userTeam.leagueId AND
    FaulsMobileService.userTeam.userId IN (" + user_id + ") AND FAULS.FaulsMobileService.userTeam.leagueId IN (" + lid + ") WHERE  FaulsMobileService.fantasyPlayers.Id IN
    (FaulsMobileService.userTeam.player6) union all SELECT * FROM FAULS.FaulsMobileService.userTeam INNER JOIN FAULS.FaulsMobileService.fantasyPlayers ON
    FAULS.FaulsMobileService.fantasyPlayers.leagueID = FaulsMobileService.userTeam.leagueId AND FAULS.FaulsMobileService.userTeam.userId IN (" + user_id + ") AND
    FAULS.FaulsMobileService.userTeam.leagueId IN (" + lid + ") WHERE FaulsMobileService.fantasyPlayers.Id IN (FaulsMobileService.userTeam.player7) union all SELECT * FROM
    FAULS.FaulsMobileService.userTeam INNER JOIN FAULS.FaulsMobileService.fantasyPlayers ON FAULS.FaulsMobileService.fantasyPlayers.leagueID =
    FAULS.FaulsMobileService.userTeam.leagueId AND FAULS.FaulsMobileService.userTeam.userId IN (" + user_id + ") AND FAULS.FaulsMobileService.userTeam.leagueId IN (" + lid + ") WHERE
    FaulsMobileService.fantasyPlayers.Id IN (FaulsMobileService.userTeam.player8) union all SELECT * FROM FAULS.FaulsMobileService.userTeam INNER JOIN
    FaulsMobileService.fantasyPlayers ON FAULS.FaulsMobileService.fantasyPlayers.leagueID = FAULS.FaulsMobileService.userTeam.leagueId AND FaulsMobileService.userTeam.userId IN (" +
    user_id + ") AND FAULS.FaulsMobileService.userTeam.leagueId IN (" + lid + ") WHERE  FaulsMobileService.fantasyPlayers.Id IN (FaulsMobileService.userTeam.player9) union all SELECT
    * FROM FAULS.FaulsMobileService.userTeam INNER JOIN FAULS.FaulsMobileService.fantasyPlayers ON FAULS.FaulsMobileService.fantasyPlayers.leagueID =
    FAULS.FaulsMobileService.userTeam.leagueId AND FaulsMobileService.userTeam.userId IN (" + user_id + ") AND FAULS.FaulsMobileService.userTeam.leagueId IN (" + lid + ") WHERE
    FaulsMobileService.fantasyPlayers.Id IN (FaulsMobileService.userTeam.player10) union all SELECT * FROM FAULS.FaulsMobileService.userTeam INNER JOIN
    FaulsMobileService.fantasyPlayers ON FAULS.FaulsMobileService.fantasyPlayers.leagueID = FAULS.FaulsMobileService.userTeam.leagueId AND FaulsMobileService.userTeam.userId IN (" +
    user_id + ") AND FAULS.FaulsMobileService.userTeam.leagueId IN (" + lid + ") WHERE  FaulsMobileService.fantasyPlayers.Id IN (FaulsMobileService.userTeam.player11)";

    mssql.query(sql, {
        success: function (results) {
            response.send(200, results);
        }
    });
};
```

Appendix J: Example API 2 – ‘calculatepoints’ API



```
exports.post = function (request, response) {
    var mssql = request.service.mssql;
    var leagueid = request.body.leagueid;

    var sql = "SELECT FaulsMobileService.userTeam.userId, (SUM(FaulsMobileService.fantasyPlayers.rulePointIncrement)) AS pointsthisweek FROM FaulsMobileService.userTeam JOIN
    FaulsMobileService.fantasyPlayers ON FaulsMobileService.fantasyPlayers.leagueID = FaulsMobileService.userTeam.leagueId WHERE FaulsMobileService.fantasyPlayers.Id IN  (
    FaulsMobileService.userTeam.player1, FaulsMobileService.userTeam.player2, FaulsMobileService.userTeam.player3, FaulsMobileService.userTeam.player4,
    FaulsMobileService.userTeam.player5, FaulsMobileService.userTeam.player6,FaulsMobileService.userTeam.player7, FaulsMobileService.userTeam.player8,
    FaulsMobileService.userTeam.player9,  FaulsMobileService.userTeam.player10, FaulsMobileService.userTeam.player11) AND  FaulsMobileService.userTeam.leagueId = " + leagueid + "
    GROUP BY FaulsMobileService.userTeam.userId, FaulsMobileService.userTeam.userPoints  UNION ALL   SELECT FaulsMobileService.userTeam.userId,    (SUM
    (FaulsMobileService.fantasyPlayers.rulePointIncrement)) AS pointsthisweek    FROM    FaulsMobileService.userTeam JOIN FaulsMobileService.fantasyPlayers ON
    FaulsMobileService.fantasyPlayers.leagueID = FaulsMobileService.userTeam.leagueId    WHERE    FaulsMobileService.fantasyPlayers.Id IN (FaulsMobileService.userTeam.captain)    AND
      FaulsMobileService.userTeam.leagueId = " + leagueid + " GROUP BY (FaulsMobileService.userTeam.userId), FaulsMobileService.userTeam.userPoints UNION ALL SELECT
    FaulsMobileService.userTeam.userId,    (SUM(FaulsMobileService.fantasyPlayers.rulePointIncrement * 0.5)) AS pointsthisweek    FROM    FaulsMobileService.userTeam JOIN
    FaulsMobileService.fantasyPlayers ON FaulsMobileService.fantasyPlayers.leagueID = FaulsMobileService.userTeam.leagueId WHERE FaulsMobileService.fantasyPlayers.Id IN
    (FaulsMobileService.userTeam.vicecaptain) AND FaulsMobileService.userTeam.leagueId = " + leagueid + " GROUP BY (FaulsMobileService.userTeam.userId),
    FaulsMobileService.userTeam.userPoints";

    mssql.query(sql, {
        success: function (results) {
            response.send(200, results);
        }
    });
};

```

Appendix K: Example API 3 – ‘Merge Player / User Accounts’ API


insertplayerlinkid.js App_Data/config/scripts/api

SAVED

```
exports.post = function (request, response) {
    var mssql = request.service.mssql;
    var userid = request.body.userid;
    var leagueid = request.body.leagueid;
    var playerid = request.body.playerid;

    var sql =
        "update FaulsMobileService.userLeagues set  FaulsMobileService.userLeagues.userFantasyPlayerLinkId = " + playerid + "WHERE FaulsMobileService.userLeagues.userId = " + userid +
        "AND leagueId = " + leagueid;

    mssql.query(sql, {
        success: function (results) {
            response.send(200, results);
        }
    });
};
```

Appendix L: Original Risk Assessment

Risk	Severity (L/M/H)	Likelihood (L/M/H)	Significance (Sev. x Like.)	How to Avoid	How to Recover
Data loss	H	L	HL	Routinely backup project code, project documentation and other critical data. Ensure to do so after long periods of working.	Restore lost data from the backups
Loss of backups	H	L	HL	Maintain multiple backups and use different technologies for storing backups.	Recover lost data from an alternative backup solution.
Bonus objective too ambitious	H	M	HM	Focus on delivering the core aim of the project (the phone application) and assess whether or not time is available to deliver.	Cut the bonus objective, focus on enhancing the ability to manage player scores through the application.
Multiple deadline/working commitments	H	H	HH	Manage time effectively by ordering work commitments in terms of priority. Allow for additional time in the plan nearer 'deadline/exam' periods.	Focus efforts on delivering the essentials to the best standard possible. If required cut unessential / lower priority features.
Lack of platform understanding / knowledge	M	M	MM	Consider the platform and technologies required to deliver the project and spend time researching these.	Find an alternative solution to the problem. Other technologies available? Is the feature essential to the aim? Consider these possibilities.
Poor User Experience	M	M	MM	Test and gain feedback on a number of different possible designs. Research existing products and asses their design pros / cons.	Action user feedback to provide a better user experience. Consider the application 'flow' and whether or not it's logical.
Device Failure	L	L	LL	Provide sufficient care of devices and personal computer.	Should the PC fail, use computers located in throughout the university to limit project downtime until a fix can be found. Regularly commit work onto SVN / other cloud storage backups.

Appendix M: Sample Test Data (Experimental Testing)

Adding Rules			
Rule Name	**Test 1 Rule Value**	**Test 2 Rule Value**	**Test 3 Rule Value**
Goals	25	10	40
Assists	10	5	25
Appearences	5	3	20
BUCS Appearences	5	5	20
Man of the Match	50	25	75
Starting Line-up	2	1	5
Red Card	-50	-15	-75
Yellow Card	-25	-10	-40
Green Card	-10	-5	-24
Clean Sheets	30	40	45
Updating Player Statistics			
RULE	**TEST**		
R1	**Test 1**	**Test 2**	**Test 3**
R2	**1**	**-2**	0
R3	0	0	0
R4	0	0	**3**
R5	**3**	**4**	0
R6	0	0	0
R7	0	**7**	0
R8	0	0	0
R9	**5**	0	**2**
R10	0	**7**	**6**

References

Lee K.Farquhar and Robert Meeds, (2007). *Types of Fantasy Sports Users and Their Motivations*. Journal of Computer-Mediated Communication, Volume 12, Issue 4. Available online: http://onlinelibrary.wiley.com/doi/10.1111/j.1083-6101.2007.00370.x/full
[Accessed: 01/05/2015]

Oxford Dictionaries, 2016., Fantasy Football Definition. [Online]
Available at: http://www.oxforddictionaries.com/definition/english/fantasy-football
[Accessed 10 January 2016]

International Data Corporation, (2015). Smartphone OS Market Share, 2015 Q2
Available Online: http://www.idc.com/prodserv/smartphone-os-market-share.jsp
[Accessed 08 October]

Microsoft Azure (2016). What is Azure
Available Online: *https://azure.microsoft.com/en-gb/overview/what-is-azure/*
[Accessed 05 May 2016].

Google Cloud (n.d.). Google App Engine
Available Online: https://cloud.google.com/appengine/
[Accessed 10 May 2016].

Android (2015). *Platform Versions*
Available Online: https://developer.android.com/about/dashboards/index.html
[Accessed 09 October 2015].

Williams, R, (2015). *Britain turns its back on Android in favour of iOS.* [Online]
Available Online: http://www.telegraph.co.uk/technology/apple/iphone/11329767/Britain-turns-its-back-on-Android-in-favour-of-iOS.html
[Accessed 08 October 2015].

Charland, Andre and Brian Leroux. (2011) *Mobile Application Development: Web Vs. Native.*
Communications of the ACM 2011: Volume 54 Issue 5, Page 49-53.
Available at: http://dl.acm.org/citation.cfm?id=1941504
[Last Accessed: 12 October]

Xamarin, 2016, UI Form controls [Online]
Available at: https://www.xamarin.com/forms
[Accessed 18/04/2016]

Xamarin 2016, Platforms [Online]
Available at: https://www.xamarin.com/platform
[Accessed 18/04/2016]

Techcrunch, 2010., Google Cloud Launch. [Online]
Available at: http://techcrunch.com/2010/05/18/google-to-launch-amazon-s3-competitor-google-storage-at-io/
[Accessed 12 January 2016]

Google Cloud, n.d, Google Cloud Customers. [Online]
Available at: https://cloud.google.com/customers/
[Accessed 12 January 2016]

Facebook Developers, 2013, Welcoming Parse to Facebook. [Online]
Available at: https://developers.facebook.com/blog/post/2013/04/25/welcoming-parse-to-facebook/
[Accessed 15 January 2016]

Fantastar, 2005, Features. [Online]
Available at: http://www.fantastar.com/General/fantastar-fantasy-leagues.aspx
[Accessed 15 January 2016]

Fantasizr, 2015, Homescreen. [Online]
Available at: http://www.fantasizr.com
[Accessed 15 January 2016]

Hernandez, M. J. (1997) *Database design for mere mortals: a hands-on guide to relational database design*.
Reading, Mass.: Addison-Wesley Developers.
Page 16

Azure, 2016, Developer Documentation [Online]
Available at: https://azure.microsoft.com/en-gb/documentation/articles/mobile-services-android-how-to-use-client-library/
[Accessed 14 May 2016]

Material Design, n.d., Material Design. [Online]
Available at: https://www.google.com/design/spec/material-design/introduction.html
[Accessed 15 January 2016]

BUCS, 2016, University Team Membership. [Online]
Available at: http://www.bucs.org.uk/page.asp?section=16983&sectionTitle=About+Us
[Accessed 09 April 2016]

Hull University Union, 2016, Hull Athletic Union Membership [Online]
Available at: https://hullstudent.com/activities/sports-teams
[Accessed 10/04/2016]

Develop Android, 2016, ViewPager [Online]
Available at: http://developer.android.com/reference/android/support/v4/view/ViewPager.html
[Accessed 16/04/2016]

Develop Android, 2016, Intents [Online]
Available at: http://developer.android.com/reference/android/content/Intent.html
[Accessed 16/04/2016]

Develop Android, 2016, Bundles [Online]
Available at: http://developer.android.com/reference/android/os/Bundle.html
[Accessed 17/04/2016]

Develop Android, 2016, Using the AsyncTask object [Online]
Available at: http://developer.android.com/reference/android/os/AsyncTask.html
[Accessed 17/04/2016]