

COM549

Computing Systems Project

BSc (Hons) Computing Systems

FINAL REPORT

|  |  |
| --- | --- |
| Student Name: | Ryan Butler |
| Student Number: | B00583770 |
| Supervisor Name: | Dr Philip Catherwood |
| Date: | April 2018 |

Abstract  
  
The motivation for this project came when listening to a close friend’s frustrations with limitations in his place of work. As a youth coach for Juventus in Bahrain, he felt restricted with the information he could use for coaching, team set-up and performance analysis.

This caused this project to be aimed at solving this problem by creating a match/player statistic collecting mobile application to help track player and team performance across a season.

The initial stages of this project were to gather information surrounding the subject area. This was completed as an extensive Literature Review of current systems on the market, software implementation options and several software development methodologies.

Requirements Analysis was carried out to focus the objectives of the project and guarantee the project fulfils its aim to a satisfactory standard for all stakeholders. This resulted in complete and concise functional and non-functional requirements being documented.

Once requirements were set Design of the system began incorporating high-level and low-level architectures and designs.

Implementation followed using a modified waterfall approach which allowed the flexibility to enable all deadlines to be met.

The system was tested throughout to ensure quality at every stage. Extensive testing was also carried out once implementation was complete to identify any errors before presenting the finished product to the end user.

Finally, an evaluation of the final system was required to assess the success of the project. Overall the stakeholder was very impressed with the complexity of the system I was able to create in such a tight timeframe.

Although the stakeholder is happy I do intend to work on this project in the future to make further enhancements to increase its functionality and efficiency.

Acknowledgements  
  
I would like to take this opportunity to thank the people that have assisted and supported me while completing this project as well as supporting me through my time in university.

First and foremost, I would like to give my thanks to my project supervisor Dr Philip Catherwood. Your support and guidance throughout this year has been essential to my success and the quality of the overall project. Having someone there, who has vast experience and knowledge has been priceless, and for this, I thank you.

I would also like to thank the Course Co-ordinator, Dr Donald McFall, and the Module Co-ordinator, Dr Kenneth Adamson. I’d like to thank both for their tireless support during this intense final year.

I would especially like to show my gratitude to Dr Donald McFall for accepting my application to enrol in the course with advanced entry. I hope I have proved this decision to be worthwhile and I will be forever appreciative of the opportunity you have given me.

My family deserve my thanks for their assistance and encouragement over the last 12 months which has kept me motivated and calm during the completion of this project.

To my close friends, Jordan Stephens and Ryan Clarke, I thank you both for your assistance throughout this project. Jordan has been open and critical with everything I brought him ensuring a high quality was maintained. Whereas, Ryan aided my thought processes and was my “go-to” when struggling with programming concepts. Without both of you the project could not have been completed.

My girlfriend, Roisin Jackson, has been a pillar of support and without her this would not have been possible. The smallest gestures from her went a long way in helping me achieve my goals and perform to the best of my ability.

To everyone above, I am eternally grateful

Table of Contents

**CHAPTER 1 – INTRODUCTION ………………………………………………… 001-003**

* 1. Background ………………………………………………………... 1
  2. Project Aims ………………………………………………………… 2
  3. Project Objectives ……………………………………………….. 2
  4. Dissertation Overview ……………………………………….. 3

**CHAPTER 2 – LITERATURE REVIEW ………………………………………… 004-019**

2.1 Current System Available ……………………………………….. 4

2.1.1 Great Coach Football ………………………………. 4

2.1.2 Performa Sports ……………………………………….. 5

2.1.3 SportEasy ………………………………………………… 5

2.1.4 Comparison of Systems ………………………………. 6

2.1.5 Conclusion ………………………………………………… 7

2.2 Implementation Options ……………………………………….. 7

2.2.1 Target Platforms ……………………………………….. 7

2.2.1.1 Android ……………………………………….. 7

2.2.1.2 iOS ………………………………………………… 8

2.2.1.3 Windows ……………………………………….. 8

2.2.1.4 Comparison of Target Platforms ……………... 9

2.2.2 Development Tools and Languages ……..……………… 10

2.2.2.1 Android Studio (Android) ………………………. 10

2.2.2.2 Xcode/Swift (iOS) ………………………………. 11

2.2.2.3 Visual Studio (Windows) ………………………. 11

2.2.2.4 Cross-Platform ………………………………. 11

2.2.2.4.1 PhoneGap ……………………………… 11

2.2.2.4.2 Intel XDK ………………………………. 12

2.2.2.4.3 Xamarin ………………………………… 12

2.2.3 Chosen Target Platform ………….…………………… 12

2.2.4 Database Options ……………………………………….. 13

2.2.4.1 SQLite ……………………………………….. 13

2.2.4.2 Firebase ……………………………………….. 13

2.2.4.3 AWS Mobile ………………………………. 13

2.2.4.4 Azure Mobile ………………………………. 14

2.2.4.5 IBM Cloud ………………………………. 14

2.2.4.6 Chosen Database Option ……………………… 14

2.2.5 Conclusion ………………………………………………… 14

2.3 Software Development Lifecycles ………………………………. 15

2.3.1 Waterfall Model ……………………………………….. 16

2.3.2 Agile Model ………………………………………………… 16

2.3.3 Iterative Model ……………………………………….. 17

2.3.4 Spiral Model ………………………………………………… 18

2.3.5 Comparison of SDLC’s ………………………………. 18

2.3.6 Conclusion ………………………………………………… 19

**CHAPTER 3 – ANALYSIS …………………………………………………………. 020-025**

3.1 Requirements Gathering ……………………………………….. 20

3.1.1 Questionnaires ……………………………………….. 21

3.1.2 Observations ……………………………………….. 21

3.1.3 Interviews ………………………………………………… 21

3.1.4 End Users ………………………………………………… 22

3.2 Requirements …………………………………………………………. 22

3.2.1 Functional Requirements ………………………………. 22

3.2.2 Non-Functional Requirements ……………………… 23

3.2.3 Business Requirements ………………………………. 23

3.3 Risk Analysis …………………………………………………………. 24

3.3.1 Probability/Impact Matrix ………………………………. 25

3.4 Conclusion …………………………………………………………. 25

**CHAPTER 4 – DESIGN …………………………………………………………….. 026-038**

4.1 Use Case Diagrams ………………………………………………… 26

4.1.1 Coach Interaction ……………………………………….. 26

4.1.2 Player Interaction ……………………………………….. 27

4.1.3 Adding a Player Interaction ……………………… 27

4.1.4 Logging a Match Day Interaction ……………………… 28

4.2 User Interface Designs ……………………………………….. 28

4.2.1 Application Map ……………………………………….. 29

4.2.2 Logo Design ……………………………………….. 30

4.2.3 Login Screen ……………………………………….. 30

4.2.4 Registration Screen ………………………………. 30

4.2.5 Home Screen ……………………………………….. 31

4.2.5.1 Coach ……………………………………….. 31

4.2.5.2 Player ……………………………………….. 31

4.2.6 My Account Screen ………………………………. 31

4.2.7 Add Players Screen (Coach) ……………………… 32

4.2.8 View Players Screen (Coach) ……………………… 32

4.2.9 Log Match Day Screen (Coach) ……………………… 33

4.2.10 View Team Stats Screen ……………………… 33

4.2.11 View Individual Stats Screen ……………………… 34

4.2.12 Compare Team Performances Screen …………….. 34

4.2.13 Compare Individual Performances Screen ……. 34

4.3.14 Video Upload Screen ………………………………. 35

4.2.15 Chat Screen ……………………………………….. 35

4.3 Database Design ………………………………………………… 35

4.3.1 Users ………………………………………………… 36

4.3.2 Teams ………………………………………………… 36

4.3.3 Dates ………………………………………………… 36

4.3.4 Matches ………………………………………………… 37

4.3.5 Messages ………………………………………………… 37

4.3.6 Videos ………………………………………………… 38

4.4 Conclusion …………………………………………………………. 38

**CHAPTER 5 – IMPLEMENTATION ………………………………………………** **039-081**

5.1 Development Platform Used ………………………………………... 39

5.2 Development Technologies Used ……………………………….. 39

5.2.1 Android Studio ………………………………………… 39

5.2.2 Firebase …………………………………………………. 39

5.3 Implementation Strategy ………………………………………… 40

5.4 Coach – Register …………………………………………………. 40

5.5 Coach – Login …………………………………………………. 43

5.6 Coach – Home …………………………………………………. 45

5.6.1 Coach Navigation Menu ……………………………….. 47

5.6.2 Calendar Implementation ……………………………….. 48

5.7 Coach – My Account …………………………………………………. 52

5.8 Coach – Add Players …………………………………………………. 56

5.9 Coach – View Players …………………………………………………. 58

5.10 Player – Login ………………………………………………………….. 60

5.11 Player – Home ………………………………………………………….. 61

5.12 Player – My Account …………………………………………………. 63

5.13 Coach – Log Match Day ………………………………………… 63

5.14 View Team Statistics …………………………………………………. 67

5.15 View Player Statistics ………………………………………… 69

5.16 Compare Team Performance ……………………………….. 69

5.17 Compare Player Performance ……………………………….. 71

5.18 Chat Facility …………………………………………………………. 74

5.19 Video Upload …………………………………………………………. 77

5.20 Video View …………………………………………………………. 80

**CHAPTER 6 – TESTING …………………………………………………………… 082-086**

6.1 Unit Testing …………………………………………………………. 82

6.2 System Testing …………………………………………………………. 83

6.2.1 Issues Encountered ……………………………….. 84

6.2.1.1 Deleted User Login ………………………. 84

6.2.1.2 Video Progress Error ………………………. 85

6.2.1.3 Chat Notifications ………………………. 85

6.2.1.4 Samsung Galaxy S5 Error ……………… 85

6.3 User Acceptance Testing ………………………………………… 85

6.4 Conclusion ………………………………………………………….. 86

**CHAPTER 7 – EVALUATION AND RESULTS …………………………………. 087-097**

7.1 Reflection on Technologies Used ……………………………….. 87

7.1.1 Android Studio ………………………………………… 87

7.1.2 Firebase …………………………………………………. 87

7.2 Requirements Evaluation ………………………………………… 88

7.2.1 Functional Requirements ……………………………….. 88

7.2.2 Non-Functional Requirements …………………….. 89

7.2.3 Business Requirements ……………………………… 90

7.3 End User Evaluation ………………………………………. 90

7.3.1 Coaches (Jordan Stephens) …………………….. 91

7.3.2 Players ……………………………………………….. 92

7.4 Reflection on Project Aims and Objectives …………….. 96

**CHAPTER 8 – CONCLUSION ……………………………………………………. 098-099**

8.1 Potential Future Enhancements ……………………………… 98

8.2 Project Reflection and Conclusion ……………………………… 98

**REFERENCES ………………………………………………………………………. 100-104**

**APPENDICES ………………………………………………………………………... 105-129**

Listing of Figures

|  |  |  |  |
| --- | --- | --- | --- |
| **FIGURE** | **TITLE** | **PAGE** |  |
| Figure 1.1 | Example of Current System | 1 |  |
| Figure 2.1 | Great Coach Football Screen Examples | 4 |  |
| Figure 2.2 | Performa Sports Screen Examples | 5 |  |
| Figure 2.3 | SportEasy Screen Examples | 5 |  |
| Figure 2.4 | Worldwide Smartphone OS Market Share | 10 |  |
| Figure 2.5 | Waterfall Model | 16 |  |
| Figure 2.6 | Agile Model | 16 |  |
| Figure 2.7 | Iterative Model | 17 |  |
| Figure 2.8 | Spiral Model | 18 |  |
| Figure 3.1 | Probability/Impact Matrix | 25 |  |
| Figure 4.1 | Use Case Diagram - Coach User Interaction | 26 |  |
| Figure 4.2 | Use Case Diagram - Player User | 27 |  |
| Figure 4.3 | Use Case Diagram – Adding a Player Interaction | 27 |  |
| Figure 4.4 | Use Case Diagram – Match Day Interaction | 28 |  |
| Figure 4.5 | Application Map – Coach | 29 |  |
| Figure 4.6 | Application Map – Player | 29 |  |
| Figure 4.7 | Mock Logo Designs | 30 |  |
| Figure 4.8 | Login Screen Design | 30 |  |
| Figure 4.9 | Register Screen Design | 30 |  |
| Figure 4.10 | Coach Home Screen | 31 |  |
| Figure 4.11 | Player Home Screen | 31 |  |
| Figure 4.12 | My Account – Coach | 32 |  |
| Figure 4.13 | My Account - Player | 32 |  |
| Figure 4.14 | Add Players Screen | 32 |  |
| Figure 4.15 | View Players Screen | 32 |  |
| Figure 4.16 | Log Match Day Screen | 33 |  |
| Figure 4.17 | View Team Stats | 33 |  |
| Figure 4.18 | View Ind Stats Screen | 34 |  |
| Figure 4.19 | Compare Team Perf Screen | 34 |  |
| Figure 4.20 | Compare Player Perf | 34 |  |
| Figure 4.21 | Video Upload Screen | 35 |  |
| Figure 4.22 | Chat Screen | 35 |  |
| Figure 4.23 | Users Data Structure | 36 |  |
| Figure 4.24 | Teams Data Structure | 36 |  |
| Figure 4.25 | Dates Data Structure | 36 |  |
| Figure 4.26 | Matches Data Structure | 37 |  |
| Figure 4.27 | Messages Data Structure | 37 |  |
| Figure 4.28 | Videos Data Structure | 38 |  |
| Figure 5.1 | Register Layout | 41 |  |
| Figure 5.2 | Database Reference Code | 41 |  |
| Figure 5.3 | Register Data Verification Code | 42 |  |
| Figure 5.4 | Coach Email Verification | 42 |  |
| Figure 5.5 | Authentication Data Example | 42 |  |
| Figure 5.6 | Authenticating a User Code | 43 |  |
| Figure 5.7 | Coach Data Example | 43 |  |
| Figure 5.8 | Login Layout | 44 |  |
| Figure 5.9 | Login Code Examples | 44 |  |
| Figure 5.10 | Auto-Login Code | 45 |  |
| Figure 5.11 | OnStart code for Auto Login | 45 |  |
| Figure 5.12 | Coach Home Layout | 45 |  |
| Figure 5.13 | User Information Java Class Code | 46 |  |
| Figure 5.14 | Retrieving User Data Example Code | 47 |  |
| Figure 5.15 | Coach Menu Layout | 47 |  |
| Figure 5.16 | Navigation Bar Code | 48 |  |
| Figure 5.17 | Navigation Bar Profile Photo Code | 48 |  |
| Figure 5.18 | Adding Dates to Calendar | 49 |  |
| Figure 5.19 | Collecting Date Selected Code Snippet | 49 |  |
| Figure 5.20 | Add Date Code Snippet | 50 |  |
| Figure 5.21 | Add Match Day Code | 50 |  |
| Figure 5.22 | Date Added Example | 51 |  |
| Figure 5.23 | Loading Dates to Calendar | 51 |  |
| Figure 5.24 | Coach My Account Screen Examples | 52 |  |
| Figure 5.25 | Team Information Java Class Code | 52 |  |
| Figure 5.26 | Checking Profile Image | 53 |  |
| Figure 5.27 | ProGalleryIntent Code | 53 |  |
| Figure 5.28 | Selecting a Profile Picture Code | 54 |  |
| Figure 5.29 | Uploading a Profile Picture Code | 54 |  |
| Figure 5.30 | Alert Dialog 1 | 55 |  |
| Figure 5.31 | Changing Password Code | 55 |  |
| Figure 5.32 | Adding Player Screen Examples | 56 |  |
| Figure 5.33 | Creating a new FirebaseAuth Instance Code | 56 |  |
| Figure 5.34 | Adding Player Details Code | 57 |  |
| Figure 5.35 | Player Details within Firebase Console | 57 |  |
| Figure 5.36 | View Players Layout | 58 |  |
| Figure 5.37 | CardView Layout | 58 |  |
| Figure 5.38 | Populating Player List Code | 59 |  |
| Figure 5.39 | Deleting a Player Code | 59 |  |
| Figure 5.40 | Editing a Player Code | 60 |  |
| Figure 5.41 | Directing Different Users Code | 61 |  |
| Figure 5.42 | Player Home Layout | 61 |  |
| Figure 5.43 | Updating Player Calendar Code | 62 |  |
| Figure 5.44 | Player – My Account Layout | 63 |  |
| Figure 5.45 | Formation Selector | 63 |  |
| Figure 5.46 | RecyclerView Formatting Code | 64 |  |
| Figure 5.47 | Populating Player Grid Code | 64 |  |
| Figure 5.48 | Event OnClick Code | 65 |  |
| Figure 5.49 | Process of Logging Data | 66 |  |
| Figure 5.50 | Team Statistics Fragments | 67 |  |
| Figure 5.51 | Loading a Fragment Code | 67 |  |
| Figure 5.52 | Collecting an Event Total Code | 68 |  |
| Figure 5.53 | Player Stat Layout | 69 |  |
| Figure 5.54 | Compare Menu | 69 |  |
| Figure 5.55 | putExtra Code Snippet | 69 |  |
| Figure 5.56 | Team Performance Layout | 70 |  |
| Figure 5.57 | Collecting Team Performance Data Code Snippet | 70 |  |
| Figure 5.58 | Data Collection if-else Code Snippet | 71 |  |
| Figure 5.59 | OnStart Default List Adapter | 72 |  |
| Figure 5.60 | SetData Code Snippet | 72 |  |
| Figure 5.61 | Data Collection for Player Graphs Code Snippet | 73 |  |
| Figure 5.62 | Player Performance Layout | 74 |  |
| Figure 5.63 | Player Chat Select for Coach | 74 |  |
| Figure 5.64 | Chat Intent Example | 74 |  |
| Figure 5.65 | Chat Facility Layout | 75 |  |
| Figure 5.66 | Send Message Code | 75 |  |
| Figure 5.67 | Message Data Gathering Code Snippet | 75 |  |
| Figure 5.68 | Implementing Message Layouts | 76 |  |
| Figure 5.69 | Message Sender and Time Example | 76 |  |
| Figure 5.70 | Video Upload Layout | 77 |  |
| Figure 5.71 | Select Video File Code | 77 |  |
| Figure 5.72 | OnActivityResult Code Snippet | 78 |  |
| Figure 5.73 | OnSuccessListener for Video Upload | 79 |  |
| Figure 5.74 | OnProgressListener and OnFailuerListener Code | 79 |  |
| Figure 5.75 | Video View Layout | 80 |  |
| Figure 5.76 | Video List Creation Code | 80 |  |
| Figure 5.77 | Async Task for Progress Update | 81 |  |
| Figure 6.1 | Deleted User Access | 84 |  |
| Figure 7.1 | Evaluation Results – Question 1 | 92 |  |
| Figure 7.2 | Evaluation Results – Question 2 | 92 |  |
| Figure 7.3 | Evaluation Results – Question 3 | 93 |  |
| Figure 7.4 | Evaluation Results – Question 4 | 93 |  |
| Figure 7.5 | Evaluation Results – Question 5 | 94 |  |
| Figure 7.6 | Evaluation Results – Question 7 | 95 |  |

Listing of Tables

|  |  |  |
| --- | --- | --- |
| **TABLE** | **TITLE** | **PAGE** |
| Table 2.1 | Comparison of Current Systems | 6 |
| Table 2.2 | Comparison of Target Platforms | 9 |
| Table 2.3 | Comparison of SDLC’s | 18 |
| Table 3.1 | Functional Requirements | 22 |
| Table 3.2 | Non-Functional Requirements | 23 |
| Table 3.3 | Business Requirements | 23 |
| Table 3.4 | Risk Analysis | 24 |
| Table 5.1 | Development Platforms | 39 |
| Table 5.2 | Implementation Plan | 40 |
| Table 6.1 | Test Case Example | 83 |
| Table 7.1 | Functional Requirements Evaluation | 88 |
| Table 7.2 | Non-Functional Requirements Evaluation | 90 |

**1.1 BACKGROUND**

CHAPTER 1 - INTRODUCTION

Jordan Stephens is currently a youth football coach for Juventus under 12’s and is currently based in Saar, Bahrain. He has previously studied football at university level and has experience coaching various teams. These include a term coaching Markethill Swifts under 10’s, 8 months spent in California working at soccer camps before initially moving to Bahrain to coach for Arsenal’s youth academy back in 2016.

The academy in which Jordan coaches currently has 20 players in his under 12’s squad. Alongside this commitment, Jordan has taken it upon himself to start an enterprise of one-to-one coaching at the academy and has named this “J-Uno”.

Jordan has grown frustrated with the lack of technology available to him at this level. To date, Jordan has only had access to football related software whilst at university and this was only to provide an example of what top professionals use.

This has caused Jordan to come to me to create a statistic collecting application for him to utilise for the improvement of his team and help him get a better understanding of how his players are performing.

At present, Jordan simply takes notes in a diary whilst his team’s matches are being played. This is far from ideal as Jordan often misses key moments of the game due to time-consuming nature of taking notes.

This method also makes it difficult to track a player’s performance accurately and provide players with detailed analysis of the game. Below is an example of some match day notes Jordan has taken:

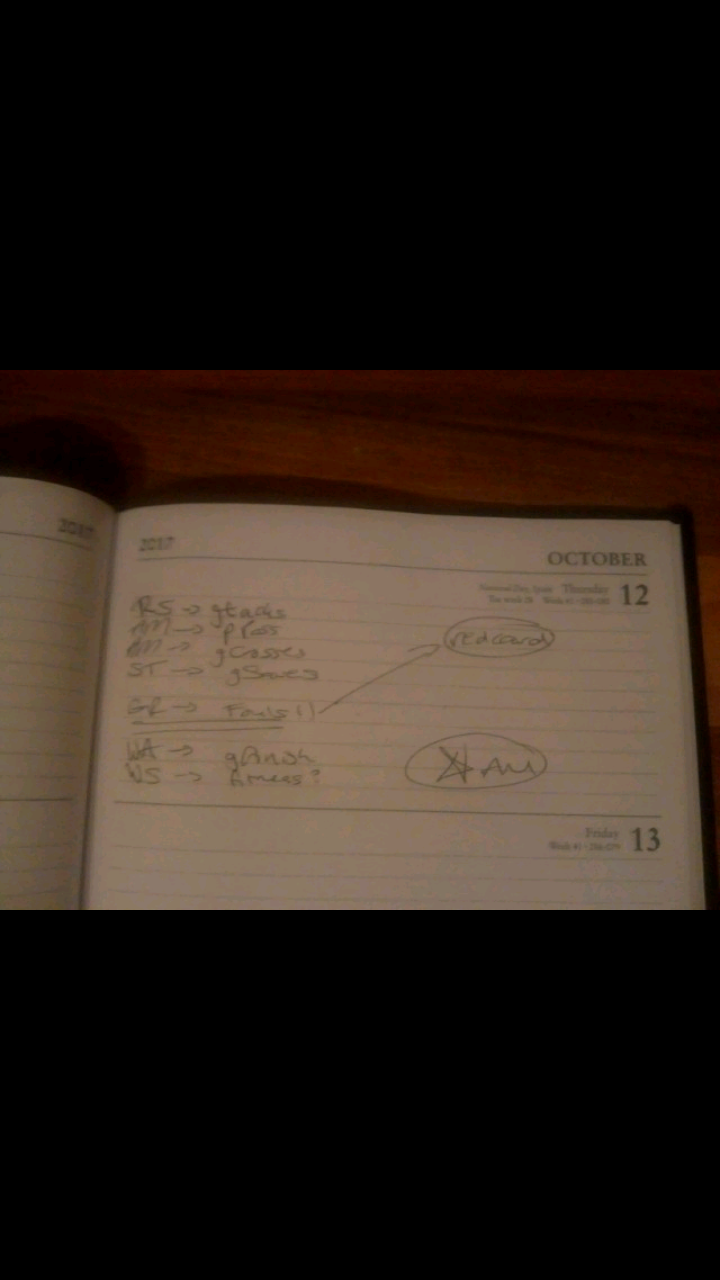


Figure 1.1 – Example of Current System

Text within image for clarity:

* RS 🡪 gtacks
* AM 🡪 pPoss
* AM 🡪 gCrosses
* ST 🡪 gSaves
* GR 🡪 Fouls!! 🡪 Red card
* WA 🡪 gFinish
* WS 🡪 fitness?
* AM

From this example it is clear to see the current system in place:

* Initials used to identify players
* Terms like “gCrosses”, “pPoss”, “gSaves” used as abbreviated version of key actions in game (Good Crosses, Poor Possession, Good Saves)
* Useful information for feedback is circled or underlined to emphasise importance
* Man of the Match indicated by ‘star’ symbol

The main problems that are evident in this system include:

1. Unorganised – there is no order, format or priorities given to the input of data
2. The detail and amount of data is often dictated by the space left on the page (when running out of room less detail is captured)
3. There is no accumulation of player statistics, simply states if player has performed good or bad in key areas throughout the game
4. Difficult to track progress – no system in place to consider previous player performances
5. Unsecure – this technique is simply taking notes in a diary, this diary can be easily read if not kept safe, putting players information at risk
6. Time consuming – taking notes physically by pen and paper causes concentration to be taken away from match/players

The system proposed intends to eliminate all the issues identified above. Its main target will be to improve efficiency and increase credibility of data collected by displaying this in a useful and meaningful way.

For the purposes of development all players created will be completely fictitious.

**1.2 PROJECT AIM**

The aim of this project is to create an application which allows the collection and sharing of football match statistics on each player in the team in the most efficient and practical manner possible

**1.3 PROJECT OBJECTIVES**

The main objectives of the project are:

1. Research and evaluate current match statistic collection software available against the needs and requirements of the client
2. Create an online database to hold coach/player/match day information
3. Create a mobile application to accommodate all users

To achieve these objectives the following activities will be performed:

1. Review similar systems currently available to the client
2. Review mobile application development software
3. Review online database tools for creating the database needed for the application
4. Define system requirements through consultation with the client
5. Review and evaluate various software development lifecycles appropriate to this project
6. Create initial designs for final product, with both coach and player functionalities considered
7. Design Database format and relationships
8. Complete testing throughout the development stages to ensure a fully functional application is created
9. Document all testing completed on the project
10. Create fully functional application to accommodate all users
11. Develop code to link online database to mobile application
12. Create a project report

**1.4 DISSERTATION OVERVIEW**

Below is an overview of what is contained in the remainder of the report:

**Chapter 2**

This chapter contains all research surrounding the project. This includes reviews on current systems available to the user and evaluations of possible target platforms, development environments/technologies, development methods and software development lifecycles. This section ends in a conclusion with the most appropriate approach being selected.

**Chapter 3**

Within this section, the system requirements are outlined with information on techniques used to create functional, non-functional and business requirements. This section also includes the risk analysis of the overall project.

**Chapter 4**

Presented in this chapter are the designs of the system, ranging from high-level designs to UI and database designs.

**Chapter 5**

Implementation is documented in this chapter with an in-depth insight into how all elements of the system were developed. Requirements outlined in Chapter 3 and designs included in chapter 4 are used as references throughout this section.

**Chapter 6**

Testing is contained within this chapter which details the strategies used during and after implementation was complete. All testing methods are defined and explained with reference to how they were used

**Chapter 7**

This chapter evaluates the completed mobile application. Project planning and maintenance is reviewed alongside the feedback from end users. Requirements outlined in Chapter 3 are reviewed against the end-product.

**Chapter 8**

The conclusion of the report evaluates the success of the project and includes the proposal of future enhancements.

The Literature Review contains all research conducted on various aspects of the project. It includes an extensive evaluation of current systems available in the open market to determine problems with current systems and advantages and disadvantages of each.

CHAPTER 2 - LITERATURE REVIEW

Following the investigation into current systems, target mobile phone platforms were reviewed along with the development tools and languages associated with each. It was also essential to evaluate all possible database technologies for the chosen platform, and after this is complete research was conducted to establish which software development lifecycle approach is most suitable for this project.

**2.1 CURRENT SYSTEMS AVAILABLE**

**2.1.1 GREAT COACH FOOTBALL**

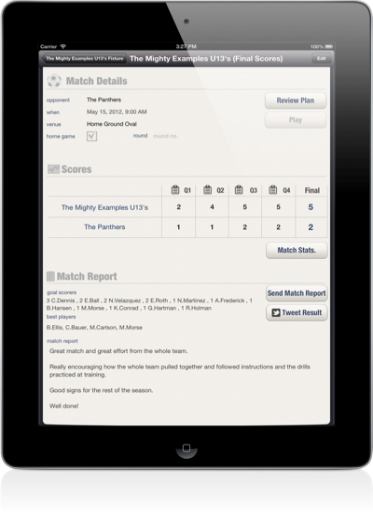


Figure 2.1 – Great Coach Football Screen Examples

(ignitedsoftware.com, 2018)

Great Coach Football a coaching and team management application available on iPad “designed by coaches for coaches”. Developed by Ignited Software who have been designing iPhone and iPad applications since 2008.

This application can be used for teams of any age range and has been especially designed to be easy to learn and produce team improvements quickly. Great Coach Football can be downloaded on the Apple App Store for £12.99.

Some of the key features of this application are:

* Users can plan team line-ups and formations which can be saved and copied for later matches
* Substitutions can be planned, and reminders set to remind coach
* Player details can be stored
* Collects player statistics and game time with indicators and alerts
* Users can review shot maps with goal/miss locations
* Allows the export of data to spreadsheets for analysis
* Social Media aspect with ability to ‘tweet’ match results
* Send match reports via email

**2.1.2 PERFORMA SPORTS**

Performa Sports is a flexible real-time and post-game performance analysis application created for iPad. This application helps users understand the bigger picture of team’s performance.  
Along with this application, Performa Sports offer accredited training courses which are designed in partnership with Ulster University to increase coach and analysts understanding and the effectiveness of the application.  
  
This application is offered through a subscription service which ranged from £299 to £1499 per year with each offering differing functionality. The package that costs £1499 per year offers the following key features:

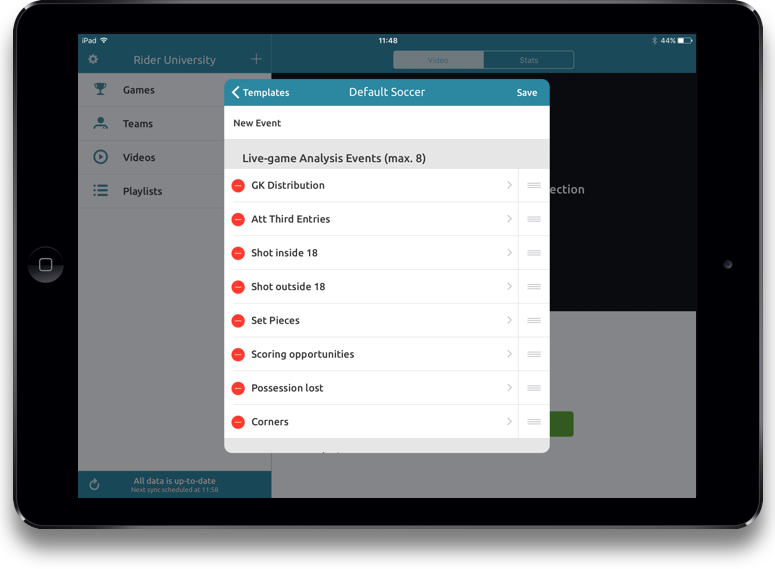
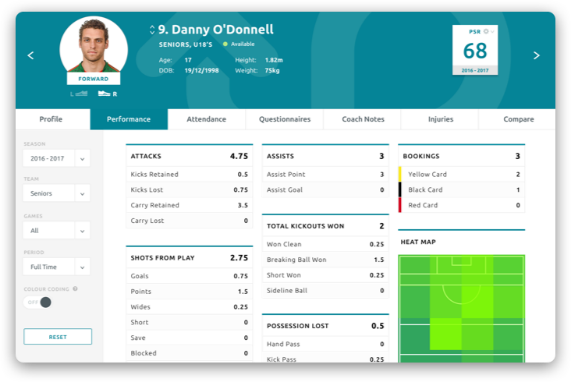


Figure 2.2 – Performa Sports Screen Examples

(Performasports.com, 2018)

* Live Game Reports
* Location Analysis
* Video Analysis
* Video Playlist Editor and Drawing Tool
* 60 Online Player Accounts
* 90 Online Videos - Storage

**2.1.3 SPORTEASY**

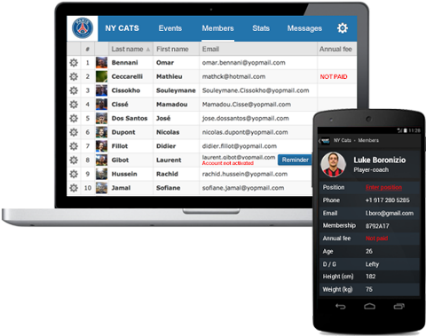


Figure 2.3 – SportEasy Screen Examples

(SportEasy.net, 2018)

“SportEasy is the most comprehensive app to communicate with your team and organise your games and practices, whatever your sport”

SportEasy is currently used by over 600,000 players and coaches around the world. It offers the facilities to manage various sports such as football, baseball, basketball, hockey and man others.

This application is available on both the Google Play Store and Apple App Store and offers various payment packages:

* Free version – contains ads, limited members, limited storage, very limited functionality
* Premium - €5 per team per month or 1 annual payment of €60
* Club - €2 per member per year

Below is a list of some of the key features available within the football version of the application:

* Various user types (coaches, players, occasional players and friends)
* Save player information (photo, position, squad number etc.)
* Offers team members web & mobile access to team information
* Schedule team events (league games, friendlies, practices etc.)
* Track participation, absences and late arrivals
* Statistic collection alongside formation editing
* Facility to vote for player of the game, comments from team members, game reports and performance analysis

**2.1.4 COMPARISONS OF CURRENT SYSTEMS**

After reviewing the current systems available individually, it is important to compare system features to get a better understanding of what is presently on the market

Table 2.1 – Comparison of Current Systems

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No.** | **Comparisons** | **Great Coach Football** | **Performa Sports** | **SportEasy** |
| 1 | Subscription | ✗ | **✗** | **✓** |
| 2 | Cost | £12.99 | £299 - £1499 | €60py or €2 pm/py |
| 3 | Target Platforms | iOS | iOS | iOS  Android |
| 4 | Training Tracker | **✗** | **✗** | **✓** |
| 5 | Pre-Game Preparation | **✓** | **✗** | **✓** |
| 6 | Live Match Statistic Collecting | **✓** | **✓** | **✓** |
| 7 | Location Tracking | **✓** | **✓** | **✓** |
| 8 | Match Reports | **✓** | **✓** | **✓** |
| 9 | Video Editing | **✗** | **✓** | **✗** |
| 10 | Export data | **✗** | **✓** | **✓** |
| 11 | Player access | **✗** | **✓** | **✓** |
| 12 | Social Media Links | **✓** | **✗** | **✗** |

**2.1.5 CONCLUSION**

All three of these systems have their own strengths and weaknesses but could all be used to meet some of the target user’s requirements. One major drawback of each is the cost, none are free to download, and this may present a problem to coaches with a tighter budget, as the more expensive (more functional) versions would be a less feasible investment for smaller teams.

Using Great Coach Football as an example, although the cost is only an upfront download cost of £12.99, it has the least number of features when comparing the three.

One other disadvantage to note during this research is that none of the three systems offer mobile phone applications. All documentation reviewed directs the use of the application to tablets/iPads.

When comparing these three systems Performa Sports and SportsEasy came out equally as well, with both only missing some key features. The addition of player access to the information collected puts these applications above Great Coach Football in respect to meeting the aim and objectives of this project.

**2.2 IMPLEMENTATION OPTIONS**

The application to be developed will be composed of a mobile application and a database to store the large amounts of data that will be collected. The incorporation of both components will ensure that the end-product is easily modified and maintained.

The main technologies that can be used to used during implementation have been reviewed:

**2.2.1 TARGET PLATFORMS**

When developing a mobile application, it is important to select the correct platform to implement it on. Currently, there are 3 main platforms in the UK to choose from; these being Android, iOS and Windows, along with the option to develop cross-platform applications. At first a brief background of each platform (Android, iOS, Windows) was investigated before comparisons were made.

**2.2.1.1 ANDROID**

Android Inc was founded in 2003 in Pao Alto, California by four developers (Rich Miner, Nick Sears, Chris White and Andy Rubin). When it was launched, one of its founders was quoted as saying they would develop “smarter mobile devices that are more aware of its owner’s location and preferences”.

It wasn’t until 2005 that the company was acquired by Google. They decided to use Linux as the basis of the Android OS due to its financial benefits as the OS could be offered to third-party mobile phone manufacturers.

It was in November 2007 that Android launched the public beta of version 1.0 for developers. The first Android smartphone device was not announced until September 2008. This phone was received negatively by the media markets due to the primitive designs and lack of standard hardware features (the handset did not have a 3.5mm headphone jack).

Although the phone itself was not a hit, it was clear with this first launch the trademarks that Google wished to impose onto the OS (Google Maps, YouTube, Google Search Services).

Unlike iOS, Android’s OS has gone through numerous “makeovers” but its most definitive change came in 2014, with the introduction of Lollipop 5.0, which brought forward the new ‘Material Design’ look.

**2.2.1.2 IOS**

The original iPhone and the accompanying OS was first announced in January 2007. Rumours of Apple developing a smartphone had been circulating for years before Steve Jobs took centre stage to make the announcement.

Apple’s decision to extend its iPod brand seemed only natural, taking the media capabilities of the latest iPods and adding mobile phone functionality to produce a device that could outperform competitors in each sector.

Steve Jobs has famously described the iPhone as “a revolutionary and magical product, literally five years ahead of any other mobile phone”.

Even when first released the iPhone divided opinion. Microsoft’s Steve Ballmer laughed at the iPhone on US TV claiming “it does not appeal to business customers because it does not have a keyboard”. Others were not convinced that a “one device fits all” approach was the way forward due to devices being affordable in both electronic markets.

Despite this, iPhone sold one million units in 74 days when it was released in the US, even with its extortionate price of $499 plus contract. These figures instantly established Apple as a major player in the mobile phone market.

**2.2.1.3 WINDOWS**

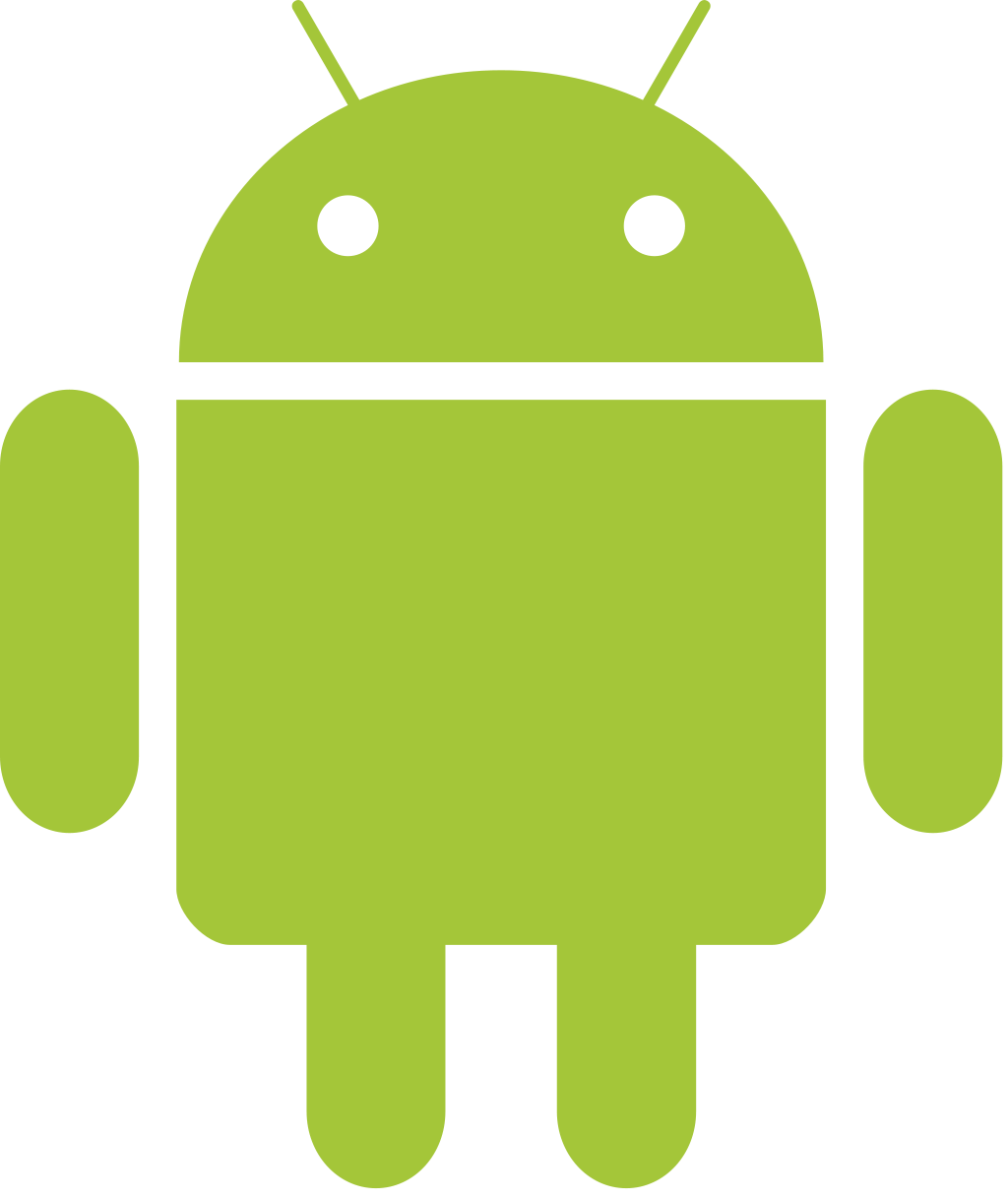
In 2010, Windows were still struggling to establish themselves in the mobile phone market. With previous attempts, such as Windows Mobile and the Kin, performing poorly it was time for Microsoft to stake a claim of the modern smartphone market.

In Barcelona the first iteration of the refreshed Windows Phone as we know it was announced. The changes made for Windows Phone 7 were received with positive feedback, with many regarding it as a great base for an OS platform.

The OS was visually different to the typical smartphone user interfaces. Normally, with systems like Android and IOS the home screens revolve around a row of icons. With this release, Microsoft had redeveloped the home screen into a 4x2 layout of solid coloured live tiles.

While several reviewers first thought this release could spell the start of Windows displacing Android or iOS, it was held back by its development approach. Microsoft had developed a locked down platform, leaving very little wriggle room for developers and users. Applications like video editing, or music player apps could not be created with the API’s currently available. Along with this the multitasking capabilities were not up to scratch with their competitors.

**2.2.1.4 COMPARISON OF TARGET PLATFORMS**

https://upload.wikimedia.org/wikipedia/zh-yue/a/ab/Apple-logo.pngTo begin comparing the three target platform options it is important to gather information on the advantages and disadvantages of each:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **IOS** | | **ANDROID** | | **WINDOWS** | |
| **ADV** | **DISADV** | **ADV** | **DISADV** | **ADV** | **DISADV** |
| Higher App Revenue  Dedicated User Base  Fewer Lines of Code  Better Development Tools  Fewer Bugs | Smaller Market Share | Biggest Market Value  Wider Demographics  Variety of Devices  Fewer Requirements for Development  Easy Publication | Fragmentation  More Potential for Bugs  Longer to Develop | Committed Users  Growing Market Share  Multiplatform amongst all MS devices | Marginal Market Share |

It is also important to account for each platforms market share, as this will become important if the application in this project were to be published. If a platform is chosen with a small market share it means less devices can support the final application and therefore less potential users.

Table 2.2 – Comparison of Target Platforms

Android – when looking at figure 5, Android’s dominance over the worldwide smartphone OS market is evident. Between 2014 and 2017 their market share percentage has not varied dramatically, remaining above 75%. It is also relevant to mention Android’s average selling prices (ASPs). Even with the attractive high-end Android products available on the market, their ASPs are expected to fall from $220 to $198 by 2021.

iOS – Apple’s OS has been runner-up to Android for a considerable time and the trends seen in Figure 5 re-iterate their position in the market. Much like Android, Apple’s market share does not fluctuate much between 2014 and 2017 with figures staying between the 10-20% region. It is also expected that iPhone shipments will grow by 3.8% throughout 2017.

Windows – This platform has a marginal global market share and these numbers continue to fall due to a lack of new hardware partners, developer support and overall enthusiasm for the platform. Within figure 5, Windows Phone market share never rises above 1% and has fallen to 0.1% by the start of 2017.

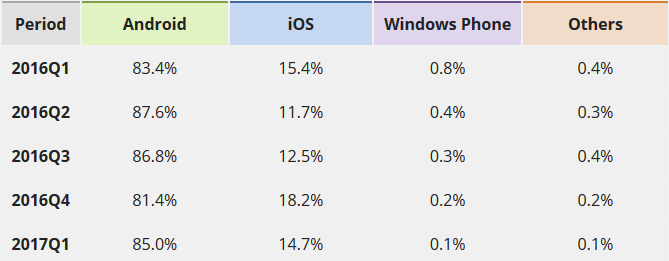
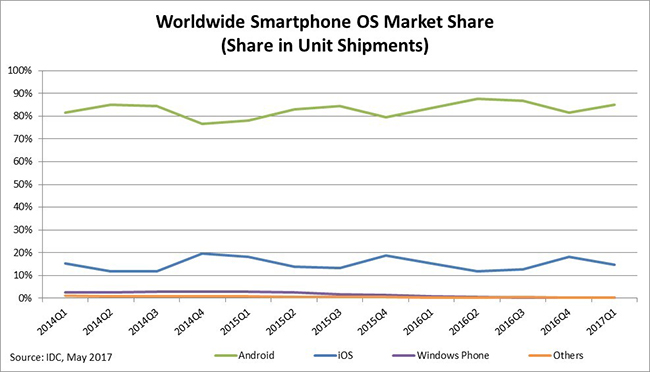


Figure 2.4 – Worldwide Smartphone OS Market Share

(www.idc.com, 2018)

**2.2.2 DEVELOPMENT TOOLS AND LANGUAGES**

When deciding on a platform to implement the application on it is essential to compare the development tools associated with each. This was also a good opportunity to investigate the possibility of developing a cross-platform application.

**2.2.2.1 ANDROID STUDIO (ANDROID)**

Android Studio is Android’s official IDE purpose built to aid developers build the highest-quality applications. It offers custom tools for Android developers including code editing, debugging, testing and profiling tools.

This IDE is for all Android developers and allows then to write in either C or C++.

System Requirements to run Android Studio on Windows:

* Microsoft Windows 7/8/10 (32 or 64-bit)
* 3GB RAM minimum, 8GB recommended
* 2GB of available disk space minimum, 4GB recommended
* 1280x800 screen resolution minimum

**2.2.2.2 XCODE/SWIFT (IOS)**

The Xcode IDE is at the centre of Apple development and is integrated with the Cocoa and Cocoa Touch frameworks. It is a quick and consistently smooth environment even when dealing with large files.

Only available on macOS this free integrated development environment was first released in 2003 and now contains tools for developing software for macOS, iOS, watchOS and tvOS.

Xcode supports development in C, C++, Objective-C, Objective-C++, Jave, AppleScript, Python, Ruby, ResEdit and Swift.

There are no specific systems requirements to note in order to run Xcode as it has been developed to work specifically on all Apple Mac Products.

**2.2.2.3 VISUAL STUDIO (WINDOWS)**

Visual Studio is an extensive integrated development environment for not only Windows phone but for Android an iOS. This IDE is not limited to mobile applications, it also allows developers to create:

* Windows apps and games to reach every device running Windows
* Azure apps that scale to Azure
* Web apps with powerful open tools and an extensible ecosystem
* Cross-platform games with leading engines and frameworks
* SQL Server and Azure SQL databases with ease

System Requirements to run Visual Studio 2017 on Windows:

* Windows 7/8/10
* 2GB RAM minimum, 4GB recommended
* On average 20-50GB of free space required, can require 130GB depending on features installed

**2.2.2.4 CROSS-PLATFORM**

Cross-platform development refers to the creation of apps that are compatible with multiple mobile operating systems. Before the arrival of cross-platform development environments it was often easier to simply build native applications for each mobile OS even though very little code was able to be repurposed.

Cross platform is an enticing possibility for this project as it would not restrict. the devices the users can run the application on. To assess the feasibility of this option 3 cross-platform development approaches were researched.

**2.2.2.4.1 PHONEGAP**

Developed by the team behind Apache Cordova, the Adobe PhoneGap framework which provides the advantages of technology created by a diverse team of pros alongside a robust developer community.

The PhoneGap CLI is the most efficient way to use the PhoneGap framework. It allows the creation, service, compilation and running of apps from the command-line terminal. The CLI can be used on any platform is written in Node.js and distributed on npm.

The only requirements to install the PhoneGap CLI in their documents:

* node.js – a Javascript runtime to build your JavaScript code
* git – used in the background to download assets

**2.2.2.4.2 INTEL XDK**

Although this environment is no longer being updated it is still an option for developers.

Intel XDK is an IDE for creating mobile Internet of Things applications using JavaScript. Originally used to make use of Cordova this cloud-based environment allows developers to create applications for Android, HTML5, Linux, OS X and Windows devices.

**2.2.2.4.3 XAMARIN**

Xamarin is an add-on feature of Visual Studio and allows users to build cross-platform applications using C# programming code. This environment allows the use of the same language, API’s and data structures for 75+% of application code across all mobile platforms.

Xamarin allows developer access to native API’s and results in finished applications not only looking as the user expected but behaving that way too.

The system requirements for this IDE are the same as using Visual Studio. This also means development can be done on either Windows or Mac.

**2.2.3 CHOSEN TARGET PLATFORM**

With all platforms reviewed and their associated development tools critiqued it was decided the implementation of this project would take place on Android Studio and therefore be targeted at android devices.

This decision was driven by 3 factors:

* resources available to the developer
* resources available to the expected end users
* market share and versatility of devices

First and foremost, the ability to develop the application is down to the resources available to the developer. There is only one developer on this project; and with Android Studio being free to download, devices cheap to purchase and system requirements reasonable to meet, this was the only feasible choice in development.

Android’s market share was also a major factor in this decision. With this not falling below 75% in the last 2 years this means an app targeted at these devices has a massive market for potential users.

The end user of this project also advised that within their users the majority would be Android devices. This was critical, and Android’s low ASP’s means any intended user can easily invest to get access to the app.

**2.2.4 DATABASE OPTIONS**

After a decision was made on the platform to implement the application on it was important to assess the database options available to this platform.

**2.2.4.1 SQLITE DATABASE**

This is Android’s default database option. It is a lightweight, transitional database engine which requires very little storage. The option is not feasible for this type of project as it is a native database, meaning only the device the application is on can store and read data.

This project’s aim and objectives means that users on different devices will need to view the same data. For this reason, SQLite will not be considered.

**2.2.4.2 FIREBASE**

Firebase was acquired by Google and has grown to have a developer community of 450,000+. It is a back-end service of huge importance to developers and allows them to store data which is easily accessed.

Firebase is simple and user friendly and offers a simple control dashboard. It also offers services including:

|  |  |
| --- | --- |
| * Realtime Database | * Crashlytics |
| * Crash Reporting | * Cloud Firestore |
| * Authentication | * Cloud Functions |
| * Cloud Storage | * Hosting |
| * Test Lab for Android | * Performance Monitoring |
|  |  |

**2.2.4.3 AWS MOBILE**

“AWS Mobile gives you the tools to rapidly configure and integrate the cloud backend your mobile app needs”

AWS is free, with every additional service having a free tier, it also allows the use of serverless technology to run code without provisioning or managing servers. On top of this it also offers additional cloud services such as:

* Secure Authentication – Sign-in UI with MFA Login with Facebook, Google and Twitter
* Database – Fast and flexible NoSQL databases that scale
* Storage – Store and retrieve user photos, profile information and settings instantly
* Bots – Deep learning conversational interfaces for your app

**2.2.4.4 AZURE MOBIILE**

A comprehensive set of cloud services that is used by developers to build, develop and manage applications. Includes integrated tools, DevOps and a marketplace to support building anything from simple apps to internet-scale solutions.

Azure Mobile allows developers to:

* Store data to the Cloud
* Authenticate Users
* Send push notifications driven by Azure

**2.2.4.5 IBM CLOUD**

Created by IBM, it has been built to enable developers to accelerate and optimize mobile app development. The IBM cloud gives developers the flexibility to choose from the latest cloud architectures and languages, whilst also having the option to extend its capabilities with powerful services like IBM Watson, Weather Company Data and more.

Its key features are:

* A Mobile foundation – secure mobile access that simplifies integration with backend and cloud services
* App ID – the addition of authentication to web and mobile apps
* Mobile Analytics – the ability to monitor performance and usage of the application
* Push Notifications – send real-time notifications

**2.2.4.6 CHOSEN DATABASE OPTION**

After reviewing all database options available it has been decided that Firebase will be used to create the database needed for this project. It is free to use, with the most user-friendly interface and extensive documentation which is easy to understand it came on top in nearly every area.

Firebase also come with the most additional features and seems to fit the specification needed for use in this project.

**2.2.5 CONCLUSION**

With all aspects of the implementation options researched the final decision is to create Android mobile application using Android’s official IDE, Android Studio. This will be accompanied by using Firebase, as a back-end service provider.

**2.3 SOFTWARE DEVELOPMENT LIFECYCLES**

The definition of a software development lifecycle (SDLC) is a framework that defines tasks performed at each step in the software development process. SDLC’s contain detailed plans on how to develop, maintain and replace specific software.

A SDLC typically consists of 5 phases, or variations on these, which include:

Analysis

SDLC’s tend to begin with Requirement Analysis. This includes discussing requirements with stakeholders to capture the detail of each requirements of the proposed software.

It is also common practice to detail how each requirement will be tested so that testers on a project add value to the requirement gathering process.

Design

High-level designs are created to deliver each requirement discussed during the analysis phase. This is often when design approaches are discussed and depending on the triple constraint of scope, time and cost of the project, the best approach is selected.

Implementation

This is when the developers can begin coding according to the requirements and design specified in the previous phases. Databases are created, interfaces and GUI’s are defined, and developers also write unit tests for each component to test new code.

This is often the longest phase of the SDLC.

Testing

This is the last phase before the software is delivered to the customer. During testing the finished software is tested against the original requirements set in the analysis stage.

The aim is to find defects within the systems whist also verifying the behaviour of the overall system. Any defects that are identified are reported and developers fix and create a new version of software.

This phase is repeated until all requirements have been tested and all defects have been resolved.

Deployment/Maintenance

After the system is successfully tested, it is deployed ready for customer to use. A maintenance team is usually in place to oversee any post-production issues.

There are many approaches to following a SDLC and for this project the following were reviewed:

* The Waterfall Model
* The Agile Model
* The Iterative Model
* The Spiral Model

**2.3.1 WATERFALL MODEL**

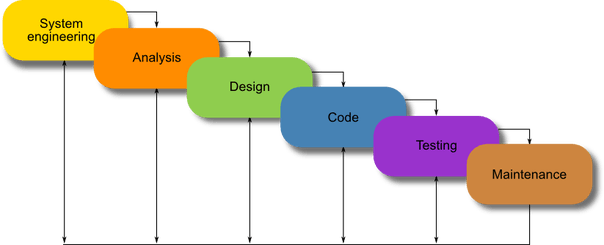


Figure 2.5 – Waterfall Model

(Powell-Morse, 2018)

Introduced in 1970 by Dr Winston W. Royce, the waterfall model emphasizes a logical progression through the phases of the SDLC. Popularity of this approach has dwindled recently with developers favouring more agile approaches.

With this said, the logical nature of the sequential process used in the waterfall method remains desirable to many developers.

|  |
| --- |
| ADVANTAGES |
| * Adapts to shifting teams * Forces structured organization * Allows for early design changes * Suited for milestone-focused development |
| DISADVANTAGES |
| * Non-adaptive design constraints * Ignores mid-process user/client feedback * Delayed testing period |

**2.3.2 AGILE MODEL**

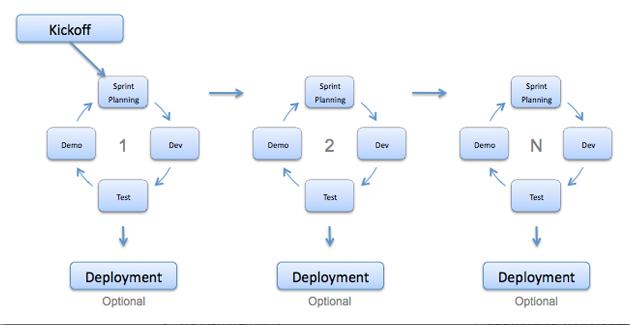


Figure 2.6 – Agile Model

(www.tutorialspoint.com, 2018)

The Agile SDLC model is an iterative, incremental model which focuses on adaptability and customer satisfaction by delivering working software in rapid intervals.

These models break a project into small incremental builds which travel through the model in iterations. Each iteration typically lasts up to 3 weeks and at the end of an iteration the working build is presented to the customer for feedback.

|  |
| --- |
| ADVANTAGES |
| * Emphasis on modern techniques * Highly adaptive * Constant customer feedback * Allows for iterative development |
| DISADVANTAGES |
| * Potential for increased “Technical Debt” * Difficult to make additions within an iteration * Minimal emphasis on documentation |

**2.3.3 ITERATIVE MODEL**

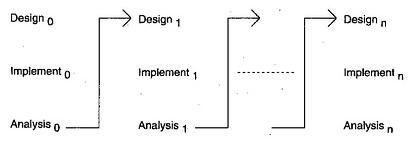


Figure 2.7 – Iterative Model

(Istqbexamcertification.com, 2018)

This model does not start with a full specification of requirements, but instead, specifies and implements a part of the software before it is reviewed to identify further requirements.

This process is then repeated, producing a new version of the software for every cycle.

|  |
| --- |
| ADVANTAGES |
| * Results are obtained early and periodically * Parallel development can be planned * Less costly to change scope/requirements * Easier to manage risk * Better suited for large and mission critical projects |
| DISADVANTAGES |
| * Cost is lesser for changing requirements, but model is not well suited to do so * Not suitable for smaller projects * End of project may not be known, which is a risk * More management attention is required |

**2.3.4 SPIRAL MODEL**

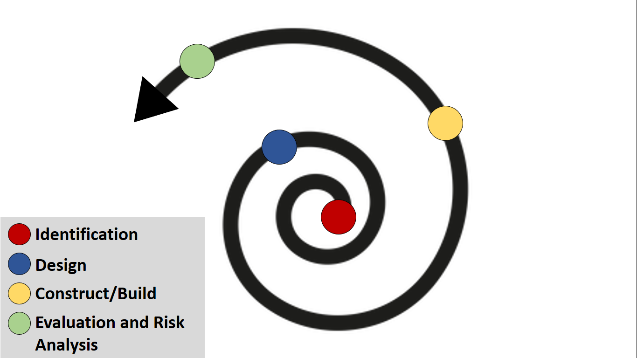


Figure 2.8 – Spiral Model

Much like an incremental model with more emphasis on risk analysis, the spiral model has 4 stages:

* Identification
* Design
* Construct/Build
* Evaluation and Risk Analysis

A project repeatedly travels through these phases in iterations (called spirals in this model).

|  |
| --- |
| ADVANTAGES |
| * Risk avoidance is enhanced * Good for large and mission-critical projects * String approval and documentation control * Software is produced early in the software cycle |
| DISADVANTAGES |
| * Can be a very costly model * High levels of risk analysis require specific expertise * Does not work well for smaller projects |

**2.3.5 COMPARISON OF SDLC’S**

Table 2.3 represents a comparison of all SDLC’s mentioned above:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Model** | | **Waterfall** | **Agile** | **Iterative** | **Spiral** |
| **Long term Project** | | Inappropriate | Appropriate | Appropriate | Appropriate |
| **Complex Project** | Inappropriate | Appropriate | Appropriate | Appropriate |
| **Frequently Changed Requirements** | Inappropriate | Appropriate | Appropriate | Appropriate |
| **Cost** | Not Costly | Costly | Costly | Costly |
| **Cost Estimation** | Easy | Difficult | Difficult | Difficult |
| **Flexibility** | Not Flexible | Flexible | Flexible | Less Flexible |
| **Simplicity** | Simple | Intermediate | Intermediate | Intermediate |
| **Guarantee of Success** | Less | High | High | High |
| **Customer Involvement** | Low | High | High | Low |
| **Testing** | Late | After each iteration | After each iteration | End of each phase |
| **Ease of Implementation** | Easy | Easy | Easy | Complex |

Table 2.3 – Comparison of SDLC’s

**2.3.6 CONCLUSION**

Having compared the advantages and disadvantages of each software development lifecycle detailed in this report it was decided the best approach to adopt for this project is, instead, a modified waterfall model.

This model has similar phases to the traditional waterfall model but reduces the defects in the original design. The main change is that phases are permitted to overlap.

This means this approach allows a structured focus on phases whilst still allowing any changes to requirements to be made with less disruption to development.

The main drawback to using this approach is the tendency to move back-and-forth between phases, fine tuning small elements which could cause a delay in project completion. This disadvantage is less concerning with the introduction of benchmarks to ensure the project does not significantly fall behind schedule.

This Analysis chapter contains the requirements gathering and finalising along with an in-depth risk analysis for the project.

CHAPTER 3 - ANALYSIS

A software requirement is a description of a feature or functionality of the target system. They can also be described as “descriptions of the services that a software system must provide and the constraints under which it must operate”.

**3.1 REQUIREMENTS GATHERING**

Gathering requirements is an essential part of any project and is key to the management of the project. Understanding fully what is expected by the end user is critical to success. Although this process seems like common sense it is often over-looked which causes issues further in the Software Development Lifecycle.

To ensure the process of gathering requirement is successful, the following 10 rules must be adhered to:

1. Don’t assume you know what the customer wants – always ask
2. Involve the users from the start
3. Define and agree on the scope of the project
4. Make sure requirements are specific, measurable, agreed upon, realistic and time-based (SMART)
5. Gain clarity if there is any doubt
6. Create a clear, concise and thorough requirements document and share it with the customer
7. Confirm understanding with the customer
8. Avoid talking about solution until requirements are fully complete and understood
9. Ensure requirements are agreed with the customer before the project starts
10. Create a prototype, if necessary, to confirm requirements

During this process there are often mistakes made, such as:

* Basing a solution on cutting edge technology, only to realise this cannot be rolled in the “real-world”
* Not prioritising requirements
* Insufficient customer involvement
* Solving the problem before fully understanding what the problem is
* Making assumptions, rather than asking

The requirements for this project were collected using various methods over a period of time. The intended system will have a total of 21 users (1 coach and 20 players). To guarantee the quality of requirements all end users have been included in this process.

Requirements were then categorised into functional and non-functional lists created and presented to the stakeholder.

There were 3 methods used to gather sufficient requirements:

**3.1.1 QUESTIONNAIRES**

To gain a better understanding of the processes in place at present it was decided that Questionnaires would be created for the players to complete. Questionnaires were chosen due to the location of the customer and the number of players the information was needed from.

Questions within the questionnaire are easy to understand and are created to give an overall insight into how match and team information is conveyed amongst the team. Care was taken to allow players the chance to express their own opinions within the questionnaire.

A copy of the original Questionnaire can be found in Appendix A, alongside a summary of results in Appendix B.

**3.1.2 OBSERVATIONS**

With observations, it is possible to identify opportunities for improvements as well as clarifying process steps and uncovering pain points or awkward steps. Observations are a great way to gather a greater understanding of existing business processes.

In the case of this project, active observations of the customer and their team was not a feasible option due to their location. Instead, other teams around Northern Ireland were observed to identify what systems are in place (if any) and to investigate how the proposed system could be implemented in a similar working environment.

Observations took place over a 3-week period with teams such as Markethill Swifts under 18’s, Armagh City under 16’s and Richhill under 16’s agreeing to facilitate the process.

These observations were a valuable experience as it granted a better perspective of the environment the system would be used in and gave the opportunity to ask questions to people in a similar field to the customer.

**3.1.3 INTERVIEWS**

Critical to creating a satisfactory software, an interview was scheduled with the main stakeholder. Due to their location, this was scheduled for when they travelled back from Bahrain, so it could be done face-to-face.

This interview was carried out in an informal setting and allowed the stakeholder to feel comfortable to give honest, more accurate responses. Open-ended questions were used to allow the customer the freedom to express concerns and opinions.

This was an opportunity to clarify any confusion regarding the project and to ask questions directly to the intended user of the system to gain a better understanding of what they expected as a result.

**3.1.4 END USERS**

Due to the nature of the proposed system, there will be two types of user for the mobile application. These being Coaches and Players.

* Coaches – will act as an ‘admin-type’ user. They will be able to register and log in to the application. They can add player, start match days and collect statistics. Data will be available for them to compare team and player performances.
* Players – login details will be given to players by their coach. Once details are created the player will be able to log in and view statistics on their own performance as well as the overall team performances.

**3.2 REQUIREMENTS**

After gathering extensive information about the needs of the customer and end users a complete list of requirements was drawn up.

**3.2.1 FUNCTIONAL REQUIREMENTS**

Functional requirements specify something a system ‘should’ do, typically they identify a behaviour or function and reflect the tasks the end user will require the system to do. Table 4 contains the functional requirements defined for this project.

|  |  |
| --- | --- |
| **No.** | **REQUIREMENT** |
| F01 | New coaches should be able to register a new account with the system |
| F02 | Registered Coaches should be able to log in to the system |
| F03 | Coaches should be able to log out of the system |
| F04 | Coaches should be able to upload a profile photo |
| F05 | Coaches should be able to upload a club badge photo |
| F06 | Coaches should be able to add new Team members to the system |
| F07 | Coaches should be able to view players information |
| F08 | Coaches should be able to view their team calendar |
| F09 | Coaches should be able to add Match Days to their team calendar |
| F10 | Coaches should be able to add Training sessions to their team calendar |
| F11 | Coaches should be able to log statistics to Match Day entries |
| F12 | Coaches should be able to view overall team statistics for completed match day entries |
| F13 | Coaches should be able to view individual player statistics for completed match day entries |
| F14 | Coaches should be able to review team performances of completed match day entries |
| F15 | Coaches should be able to review individual player performances of completed match day entries |
| F16 | Coaches should be able to send/receive messages to their players |
| F17 | Coaches should be able to upload videos to the system |
| F18 | Coaches should be able to view uploaded videos |
| F19 | Players should be able to log in to the system once created by their coach |
| F20 | Players should be able to log out of the system |
| F21 | Players should be able to upload a profile photo |
| F22 | Players should be able to view their team calendar |
| F23 | Players should be able to view overall team statistics for completed match day entries |
| F24 | Players should be able to view individual statistics for completed match day entries |
| F25 | Players should be able to review individual performances of completed match day entries |
| F26 | Players should be able to send/receive messages to their Coach |
| F27 | Players should be able to view uploaded videos |
| F28 | All users should be able to reset a forgotten password |
| F29 | All users should be able to change their password |

Table 3.1 – Functional Requirements

**3.2.2 NON-FUNCTIONAL REQUIREMENTS**

Non-functional requirements simply describe ‘how’ the system should work. They describe how the system should behave and can be referred to as quality attributes of a system. Table 5 contains the non-functional requirements defined for this project.

|  |  |
| --- | --- |
| **No.** | **REQUIREMENT** |
| NF01 | Application should have a consistent theme throughout |
| NF02 | Application should display correctly on all varying screen sizes |
| NF03 | Application should react appropriately to orientation changes |
| NF04 | The application should be easily navigated |
| NF05 | The application should not crash |
| NF06 | The application should be capable of running in the background |
| NF07 | Users should not be able to access data without logging in |
| NF08 | Passwords of all users should be stored securely |
| NF09 | The application should conform to best practice and design principles for chosen devices |

Table 3.2 – Non-Functional Requirements

**3.2.3 BUSINESS REQUIREMENTS**

Table 6 contains the business requirements defined for this system.

|  |  |
| --- | --- |
| **No.** | **REQUIREMENT** |
| B01 | The finished system should be completed by 22nd April 2018 |

Table 3.3 – Business Requirements

**3.3 RISK ANALYSIS**

Risk analysis is simply identifying those risks which could compromise the success of the project. A risk is simply a potential problem that may occur during the software development lifecycle. Any risks that are identified need to be managed to prevent damage, corruption or total loss of the project.

Risk analysis entails the identification of all potential risks to the project and the development of resolutions in the case the risk becomes apparent.

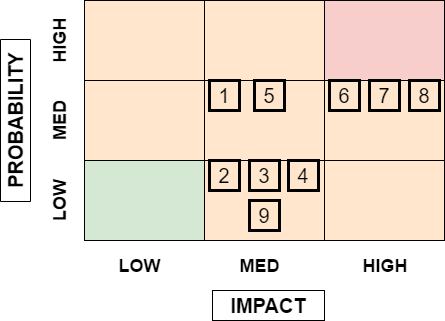
Table 7 contains the results from the risk analysis.

|  |  |  |  |
| --- | --- | --- | --- |
| **RISK NO.** | **RISK** | **IMPACT** | **RESOLUTION** |
| 1 | Falling behind schedule | Missing key deadlines, incomplete work and reduction in quality | Utilise project plan to monitor progress, plan review after requirements are finalised |
| 2 | Change in Requirements | Triple constraint of time, cost and scope could all be affected | Regular contact with client throughout development to ensure requirements are set correctly and met when completed |
| 3 | Unclear Specification | Continuous change/editing of requirements prolonging development of application and running project beyond its due date | Ample time allocated to collecting Specification during Analysis to ensure quality information gathered and all requirements are clear, concise and accurate. |
| 4 | Compromising on Design | Rushing design stages to begin implementation sooner could lead to more programming hours in later stages | Design has been allocated 4 weeks to complete within the desired timescales. |
| 5 | Productivity Issues | Projects with larger timescales can lead to developers ‘taking it easy’ to begin with. This can lead to falling behind schedule later in the project | Strict adherence to the project plan and utilisation of project monitoring techniques to ensure progress is recorded and any warning signs can be identified |
| 6 | Developer illness | This would affect the productivity and the working hours put into the project as only one developer will be completing the work | Buffers are key to reducing this risk, trying to get ahead of schedule will be vital to ensuring this risk does not affect the project due to the tight timelines |
| 7 | Loss of Data | Losing work completed and having to repeat completed work | Backing up of project data regularly |
| 8 | Unavailable/Unidentified Resources | Possibility of it impacting the developer constructing desired software. Hindering timescales and cost constraints | Ensure all resources are identified and available from the early stages of the project |
| 9 | First time project for developer | Inexperience could lead to unidentified risks, costs, resources etc. Project could fall behind schedule and/or go over budget | Communication with stakeholders is key to ensuring this risk is minimised |

**3.3.1 PROBABILITY/IMPACT MATRIX**

Table 3.4 – Risk Analysis

Once all risks were identified the risks were placed into a probability/impact matrix. This was to prioritise the list of risks into a rank order of significance. Figure 10 displays the outcome of this process.



**3.4 CONCLUSION**

Figure 3.1 – Probability/Impact Matrix

Once the requirements were specified it allowed the Design stage to begin. It is important to note that, even though careful consideration has been taken into these requirements, these may be changed throughout the development of the project. The flexible nature of the Modified Waterfall Model means this is not a major issue, but all changes or amendments to requirements will be closely monitored and managed throughout the duration of development.

The main aim of this phase is to transform the requirements, identified in the previous phase, into a detailed system architecture which is both feasible and robust.

CHAPTER 4 - DESIGN

This chapter contains high-level designs of the interaction of both users (coach and player), user interface designs for both users and a plan for the back-end database to be implemented on Firebase.

All designs within this chapter are based around the requirements defined in Chapter 3.

Design is imperative to the success of a project and because of this it is important that all designs are accurate and complete. Poor design can lead to requirements not being met and the customer being unsatisfied with the end-product. With this type of application efficiency is key, and this will come from the design of the system.

**4.1 USE CASE DIAGRAMS**

The first step to design this mobile application was to create Use Case Diagrams. These are a valuable method to visualize the functional requirements of a system.

A Use Case Diagram is a behaviour diagram in UML. The diagrams model the functionality of the system using actors and use cases. These Use Cases are a set of actions, services and functions that a system needs to perform. The ‘actors’ are people or entities which will interact with the system.

For the use case diagrams within this report the system is encased in a blue circle. This is to reduce clutter within the diagram as users will be able to travel between every screen from any screen through a navigation bar.

**4.1.1 COACH USER INTERACTION**

Figure 11 displays a Use Case Diagram overview of how a coach will interact with the system.

Figure 4.1 – Use Case Diagram - Coach User Interaction

**4.1.2 PLAYER USER INTERACTION**

Figure 12 displays an overview of how a player will interact with the system.



Figure 4.2 – Use Case Diagram - Player User Interaction

A major difference in this diagram is that players can not register themselves with the system. Players can only log in after a coach has added their details to the system. Once logged in a player can change their password.

From designing both these diagrams (Figure 11 and Figure 12), it is evident that once logged in a user is directed to their home screen. Once directed to their home screen the user has full access to the application and can go to any area highlighted in yellow. Every screen has its own functionalities which are highlighted in green.

**4.1.3 ADDING A PLAYER INTERACTION**

To get a better understanding on how some aspects of the system would need to behave a use case diagram was created to illustrate how a coach would add a player to their squad (Figure 13).



Figure 4.3 – Use Case Diagram – Adding a Player Interaction

Figure 13 shows a coach will navigate from the ‘home screen’ to an ‘add player’ screen. From here they will enter the details of the player and once complete the information will be submitted. A confirmation of the details is required and once confirmed the coach is redirected back to their home screen.

**4.1.4 LOGGING A MATCH DAY INTERACTION**

It was important to also identify the interaction required for logging a match day. This interaction is illustrated in Figure 14.



Figure 4.4 – Use Case Diagram – Match Day Interaction

This use case diagram shows a coach will navigate to the ‘log a match’ screen and once a match day is selected they will be able to enter some match details. From here they will select a player and an event, and this will be added to the database. This process can be repeated until the match day is complete. Once complete the coach is redirected to their ‘home screen’.

**4.2 USER INTERFACE DESIGNS**

When designing the application interface, it is important to consider design principles. Design principles, often referred to as heuristics, are loose guidelines designers can follow to improve the quality of a user interface design.

Below are some details on the most commonly used designed principles:

* The Structure Principle – design should organize the UI purposefully, in a meaningful and useful ways based on clear models that are recognisable to users. This principle is concerned with overall user interface architecture
* The Simplicity Principle – design should make common tasks easy, communicating clearly and simply in the user’s own language. Shortcuts should be provided that are meaningfully related to longer procedures
* The Visibility Principle – design should make all options visible without distracting the user with redundant information
* The Feedback Principle – design should keep users informed of actions, changes or errs that are relevant to the user with unambiguous explanations
* The Tolerance Principle – design should be flexible and tolerant, reducing the cost of mistakes and misuse allowing undoing and redoing
* The Reuse Principle – design should reuse internal and external components and behaviours, maintaining consistency with purpose to reduce the need for users to rethink and remember

All these principles have been considered and applied throughout this stage of development.

**4.2.1 APPLICATION MAP**

To further distinguish the layouts needed for this application it was necessary to create an application map for both coach (Figure 15) and Player (Figure 16). These designs focus on application content and what functions will be visible where.



Figure 4.5 – Application Map – Coach

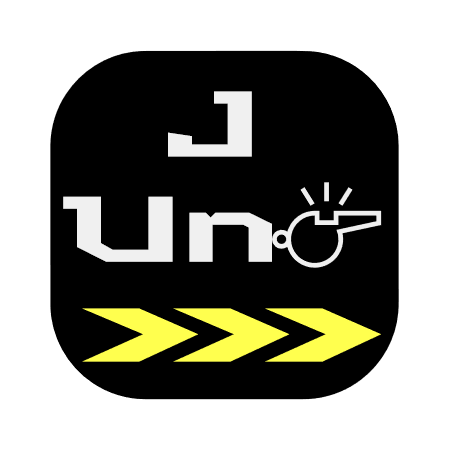
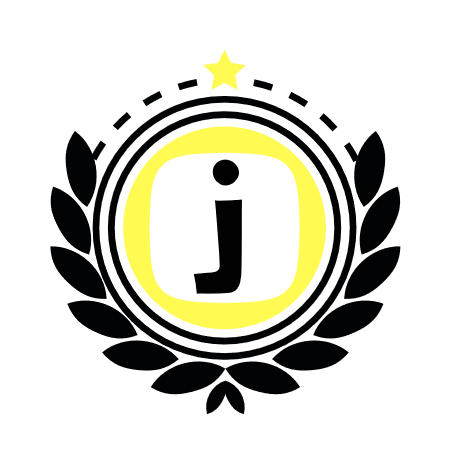
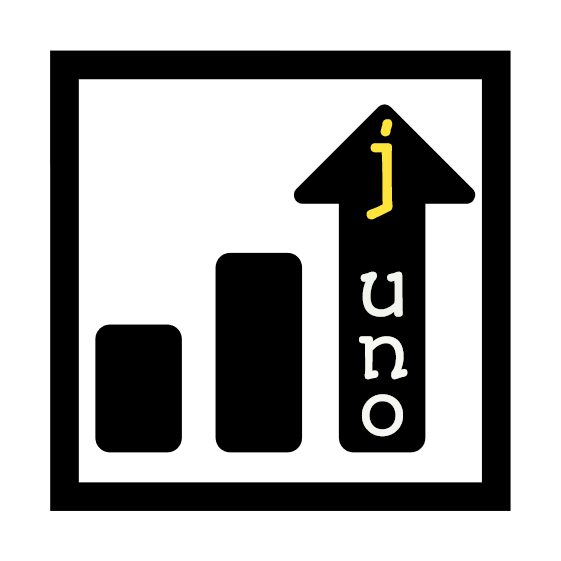


Figure 4.6 – Application Map – Player

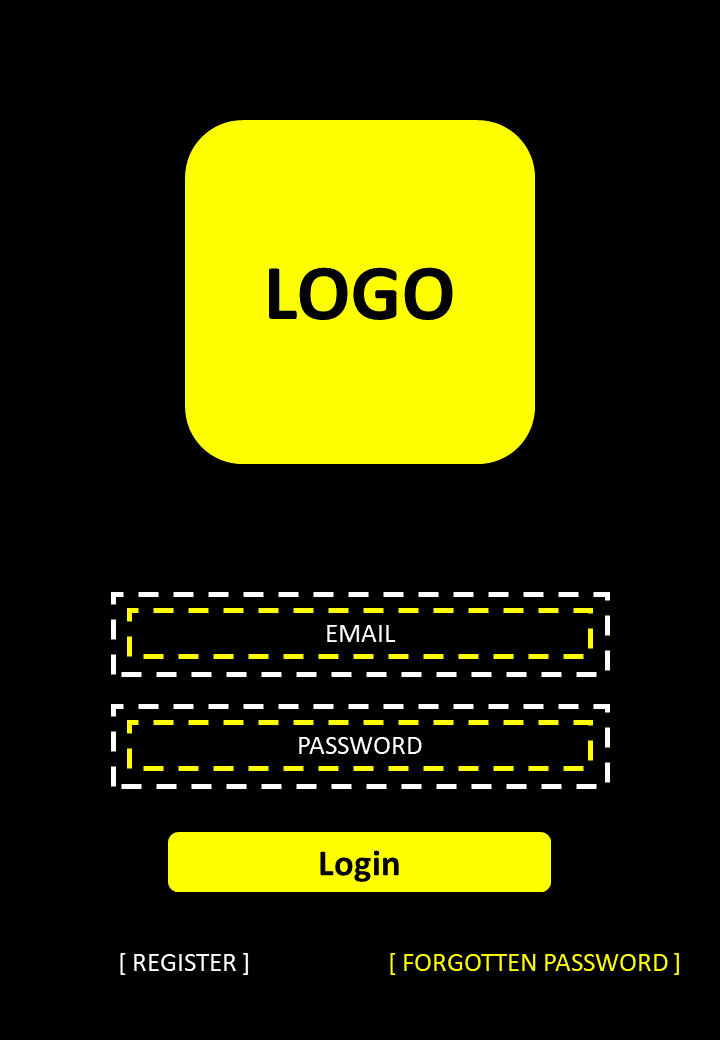
**4.2.2 LOGO DESIGN**

Figure 17 illustrates some mock logo designs created for the mobile application. At this stage it was decided the mobile application would be targeted at Jordan’s solo endeavour of one-to-one training. For this reason, the application will be named ‘J Uno’ and will follow a colour scheme specified by the stakeholder (black, yellow and white).

Figure 4.7 – Mock Logo Designs



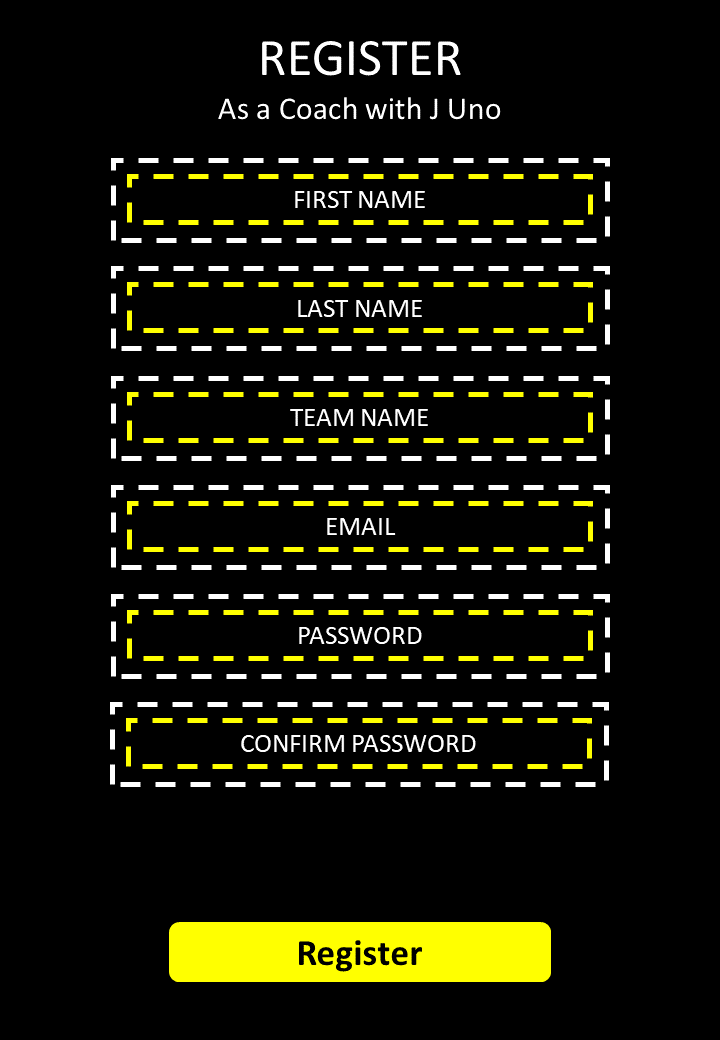
**4.2.3 LOGIN SCREEN**

Once the app is opened on a mobile device the user will be presented with this login screen. This screen gives the option for new coaches to go to the register screen and provides a link for current users to reset forgotten passwords.

An existing user will enter their email and password which will be verified once the ‘Login’ button is pressed. Log in attempts will only be made if both fields are not empty. Feedback will be given to users for unsuccessful attempts to log in.

**4.2.4 REGISTRATION SCREEN**

Figure 4.8 – Login Screen Design

If a new Coach wishes to register to use the application, they will be directed to this screen. Details are entered into text fields and verified before submission.

The user is asked to enter their password twice to guarantee correctness. The password field will not be visible once entered to improve privacy. Feedback will be given to the user if registration is unsuccessful. Once registered the user’s password will be encrypted and stored in the Database.

Figure 4.9 – Register Screen Design

**4.2.5 HOME SCREEN**

Once a user has successfully logged in they will be directed to their home screen. This screen will differ depending on the type of user. Once logged in the user will have access to a navigation bar which will be their main method of navigating through the application.



**4.2.5.1 COACH**

When a Coach successfully logs in they are greeted with their calendar. From here they will be able to add match day fixtures and training days to their team calendar.

Once a button is pressed the coach will be presented with a Calendar Picker allowing them to select a date easily and avoiding any formatting errors. Once a date is selected the calendar will be updated automatically and dates added to the database.

The coach must add a match day here to interact with it (add data) in other areas of the application.

Figure 4.10 – Coach Home Screen



**4.2.5.2 PLAYER**

When a player has successfully logged in there are greeted by their team calendar. This is updated and changed by their coach, the players cannot interact with the team calendar.

There will be a key present to explain icons within the calendar.

**4.2.6 MY ACCOUNT SCREEN**

Figure 4.11 – Player Home Screen

This screen is another screen which will differ depending on the user type. It will have the same overall functionality apart from one exception – the coach will be able to upload and save a photo from their device to use as their team’s club badge, the players will not have this facility.

Both users can upload and save a photo from their device to use as a profile photo for the application. They can both view their personal information and have access to changing their own password.

All photo data will be uploaded into the database once the user selects the ‘Submit’ button.

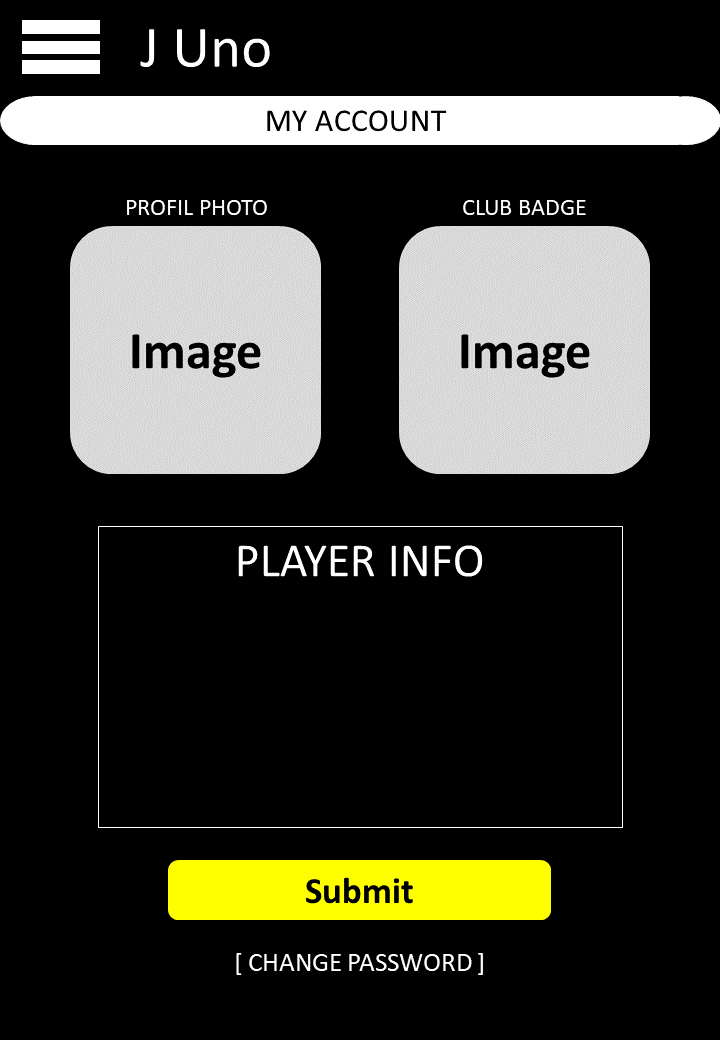
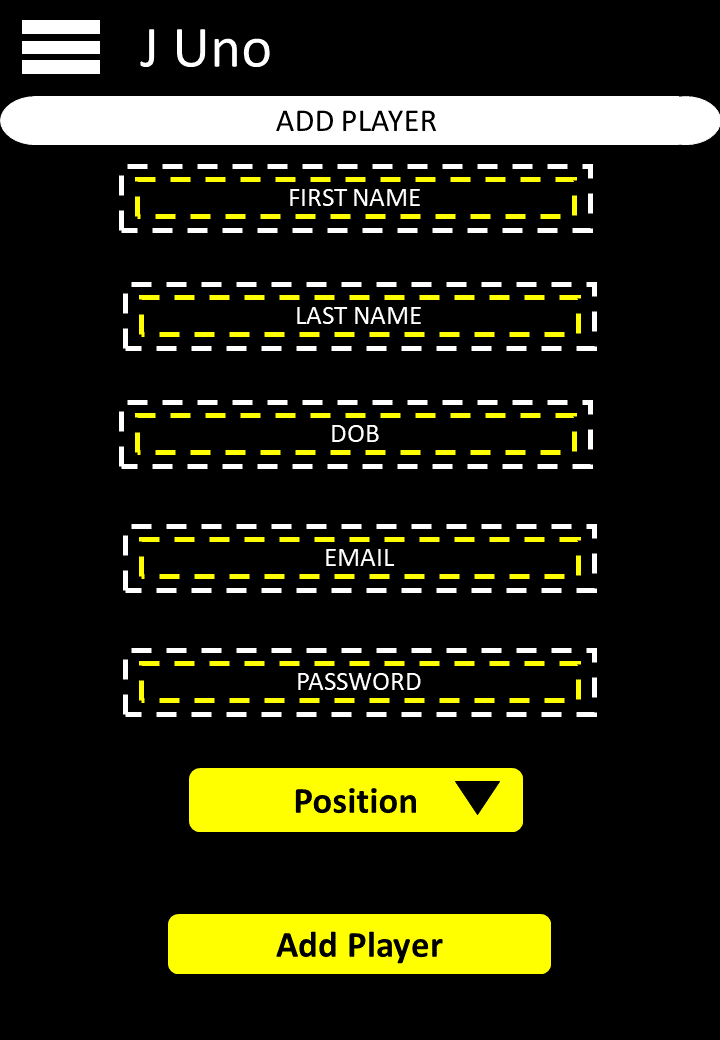


Figure 4.13 – My Account - Player

Figure 4.12 – My Account - Coach

**4.2.7 ADD PLAYERS SCREEN (COACH)**

A coach can add players to their squad through this facility. This screen is only accessible by coaches. A coach will enter all details on their player and select the ‘Add Player’ button.

The coach will create a password for that user which the player can change once logged in successfully. Details will be verified for errors before adding the players details to the database.

The coach will receive feedback on the success or failure of this process.

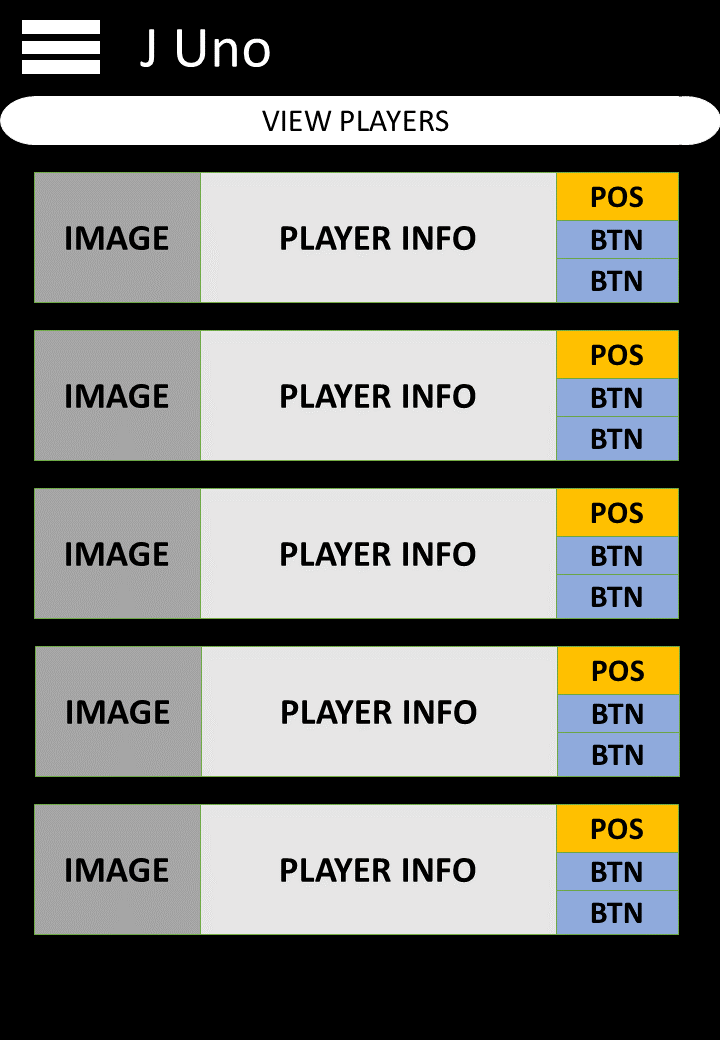
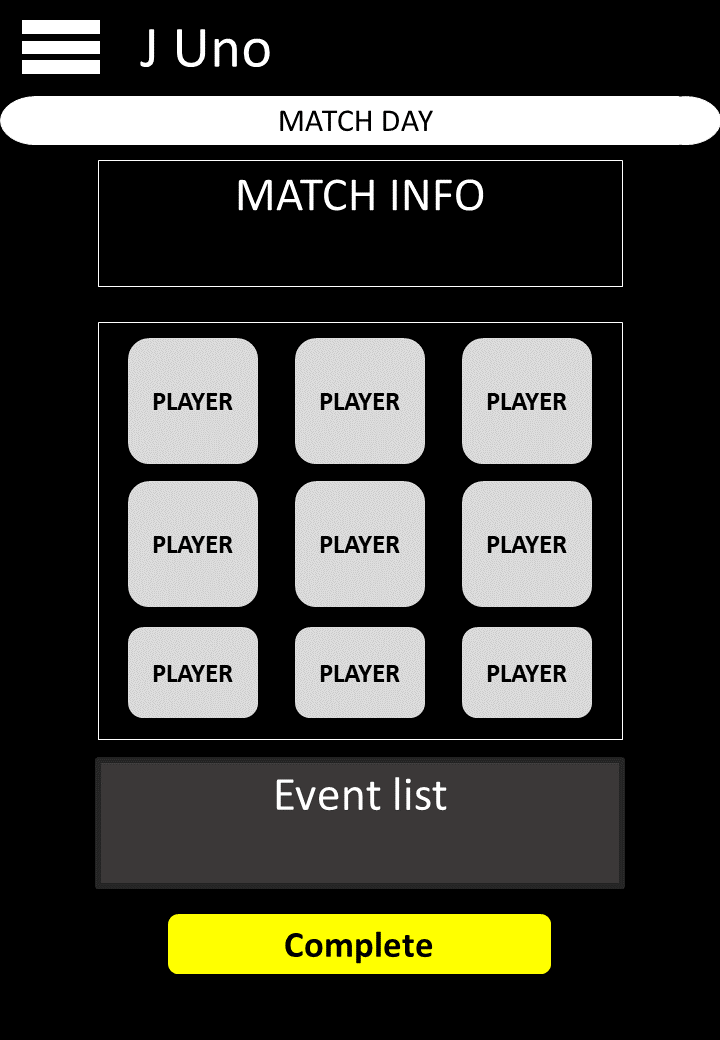
  
  
  
 **4.2.8 VIEW PLAYERS SCREEN (COACH)**

Figure 4.14 – Add Players Screen

This screen allows a coach to view details on all players in their squad. This is another screen which is only available to coaches, not players.

A coach can view personal information on their players alongside the facility to edit or delete those players.

Figure 4.15 – View Players Screen

**4.2.9 LOG A MATCH DAY SCREEN (COACH)**

This will be the screen with the most functionality. After selecting a match day from the list of dates added to their calendar, a coach will be able to enter match da details such as ‘Opponent’, ‘Home or Away’ and ‘Formation’.

There will be a list of a coach’s players in the centre of this screen and when a player is selected an ‘Alert Dialogue’ will appear with all the possibly events. These will be buttons and when selected the information will be added to the database.

There is the addition of and ‘Events’ list below the players which will provide real-time feedback to the user to reflect what has been added to the database.

Figure 4.16 – Log Match Day Screen

Options that will appear in the Alert Dialogue are:

Goal Keeper

Shot Parried/Caught

Goal Conceded

Cross Caught

Defending

Tackle Won/Lost

Foul/Yellow Card/Red Card

Interception/Block/Clearance

Header Won/Lost

Attacking

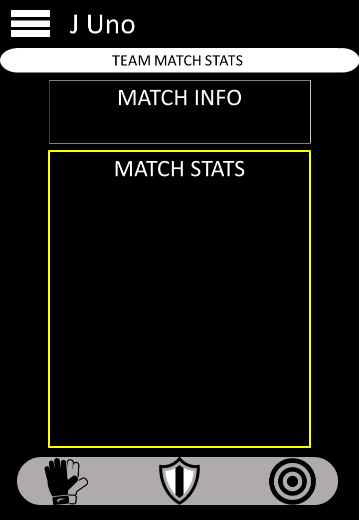
Pass Complete/Failed

Cross Complete/Failed

Shot On/Off Target

Assist/Goal

Player Fouled/Successful 1v1

**4.2.10 VIEW TEAM STATS SCREEN**

This screen will be the same for both users. It will give them access to the totals for the whole team for each event for a selected match day.

It will make use of a Bottom Navigation Bar to switch between Goal Keeper, Defending and Attacking statistics.

Percentages will be calculated for the following figures:

Tackles Won

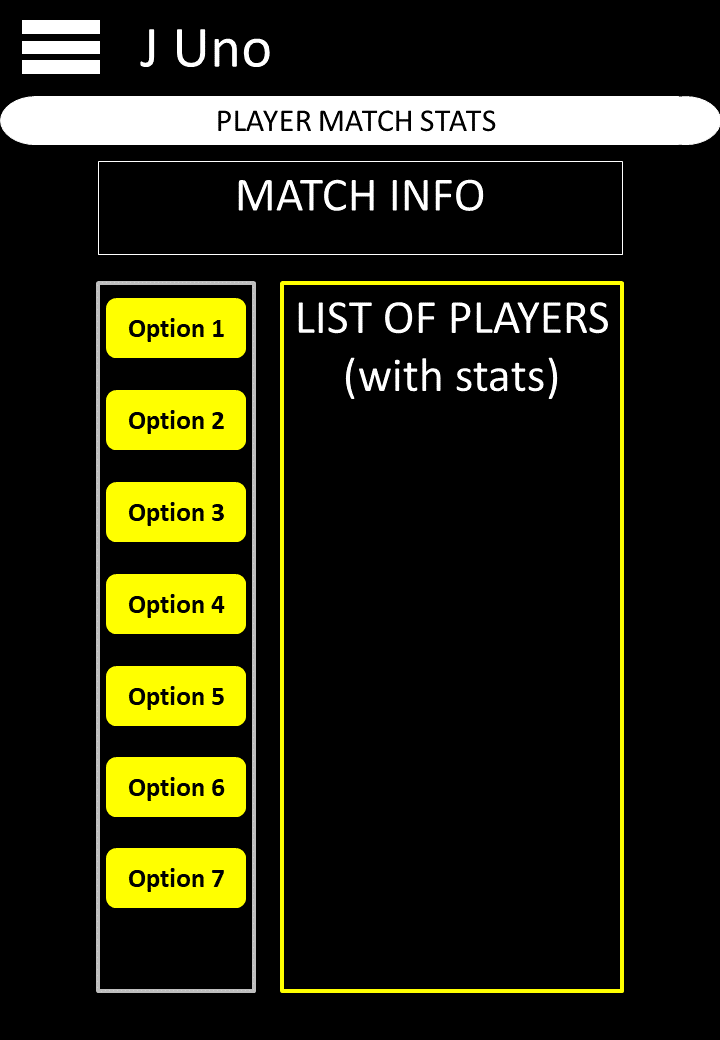
Headers Won

Pass Completion

Cross Completion

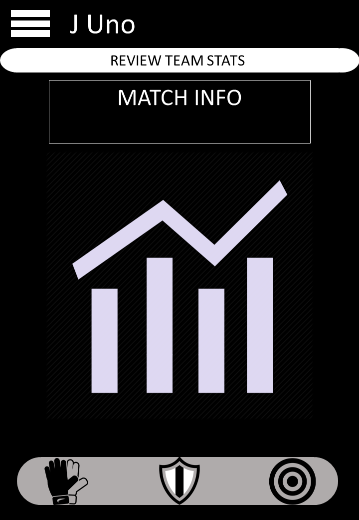
Shot Accuracy

Figure 4.17 – View Team Stats Screen

**4.2.11 VIEW INDIVIDUAL STATS**

This screen will allow the user to select a set of statistics to view from the buttons on the left and a list of players with their appropriate figures will be shown on the right.

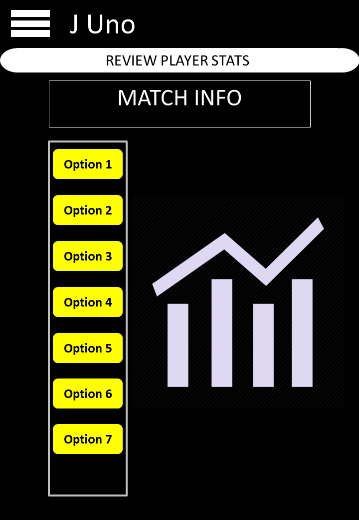
Figure 4.18 – View Ind Stats Screen

**4.2.12 COMPARE TEAM PERFORMANCE**

A user will select 2 of the completed match days to review. Once selected and confirmed the data will be displayed in a graph to illustrate variations in performance.

Again, the use of a Bottom Navigation Bar will allow users switch between goal keeper, defending and attacking statistics.

Figure 4.19 – Compare Team Perf Screen

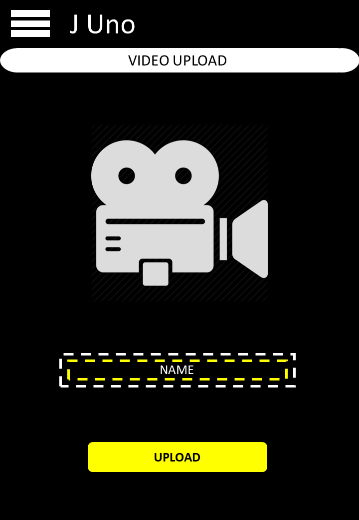
**4.2.13 COMPARE PLAYER PERFORMANCE**

This screen is the same for both users, with an exception with how it is accessed. For coaches, they will have to select a player, then 2 completed match days to view performance variances for that specific player.

For players, they will simply select 2 match days to review their own performance variances for specific match days.

Buttons will again show options for the user to display different graphs for the different statistics which will be displayed to the left of the options.

Figure 4.20 – Compare Player Perf Screen

**4.2.14 VIDEO UPLOAD SCREEN**

This screen will be accessed by the coach. They will first select an appropriate match day for the video and this will open the facility to upload a video from the mobile device.

Once a file is chosen and the user submits their choice, an upload progress bar will appear to illustrate to the user how far along the upload is. There will also be feedback to failed uploads. If successful, the video data will be saved to the database.

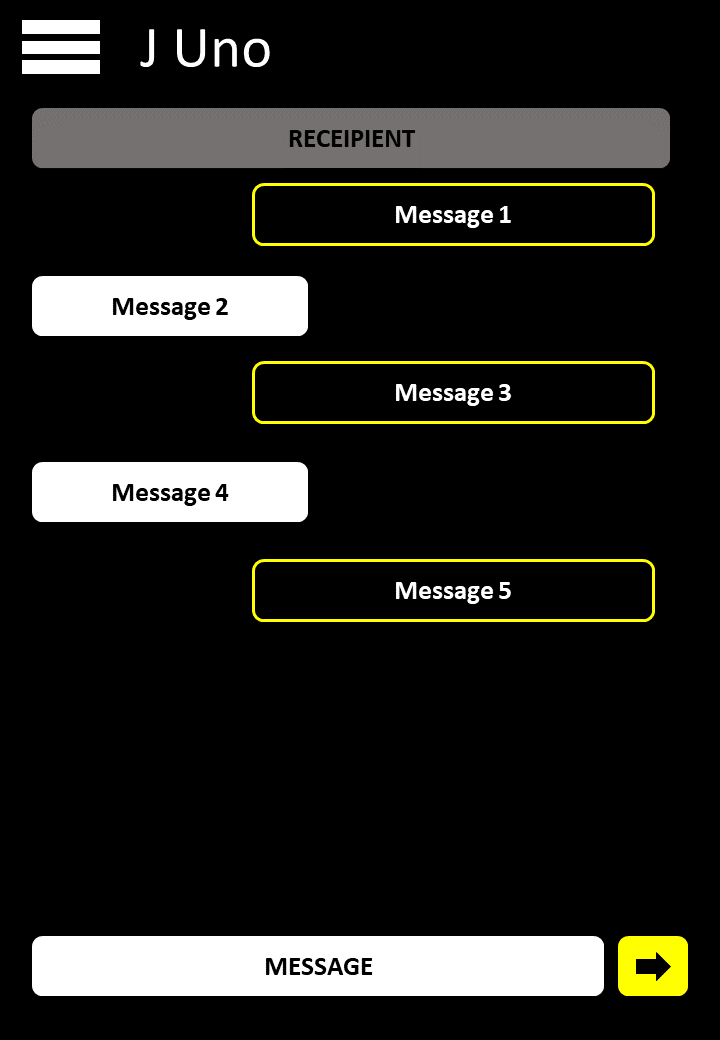
Players will be able to view videos, this will be accessed via a list of match days with available video files.   
  
  
  
  
**4.2.15 CHAT SCREEN**

Figure 4.21 – Video Upload Screen

The chat screen will look like a conventional chat service but with the application colour theme considered.

The coach will select the player to send a message to, whereas players can only send and receive messages to/from their coach.

**4.3 DATABASE DESIGN**

Figure 4.22 – Chat Screen

The design of the database in this project is imperative to the efficiency that data can be retrieved or stored. Both users of the application will need to be able to store data in and retrieve data from the database.

Previously in this report it was decided that Firebase would be used to facilitate an online database. With Firebase, data is stored as JSON and synchronized in real-time to every connected client. This database allows developers to build rich, collaborative applications by allowing secure access to the database directly from client-side code.

**4.3.1 USERS**

This section of the database will hold information on all users authenticated to use the application. The structure of the data will be:

├── Users

│ ├── Coaches

│ │ ├── UserID

│ │ │ └── FirstName

│ │ │ └── LastName

│ │ │ └── Email

│ │ │ └── Image

│ │ │ └── UserType

│ ├── Players

│ │ ├── UserID

│ │ │ └── FirstName

│ │ │ └── LastName

│ │ │ └── DOB

│ │ │ └── Email

│ │ │ └── Position

│ │ │ └── Image

│ │ │ └── TeamID

│ │ │ └── UserType

Figure 4.23 – Users Data Structure

**4.3.2 TEAMS**

This section of the database will simply hold data regarding the different teams registered by coaches. The structure will be:

├── Teams

│ ├── CoachID

│ │ └── TeamName

│ │ └── Badge

Figure 4.24 – Teams Data Structure

**4.3.3 DATES**

This section will store data regarding dates added by the coaches from their home screen. It will keep organize match days into complete and incomplete groups. The structure is shown below:

├── Dates

│ ├── CoachID

│ │ ├── MatchDays

│ │ │ └── Date1

│ │ │ └── Date2 ..

│ │ ├── Complete

│ │ │ └── Date1

│ │ │ └── Date2 ..

│ │ ├── Incomplete

│ │ │ └── Date1

│ │ │ └── Date2 ..

│ │ ├── Training

│ │ │ └── Date1

│ │ │ └── Date2 ..

Figure 4.25 – Dates Data Structure

**4.3.4 MATCHES**

This section contains several sub-sections to ensure efficiency when collecting data. This section will contain all data regarding match days, this includes match information (e.g. Opponent, Home/Away, Formation) and all events that took place.

The structure of this data will be:

├── Matches

│ ├── CoachID

│ │ ├── Date

│ │ │ ├── MatchInfo

│ │ │ │ └── Formation

│ │ │ │ └── Opponent

│ │ │ │ └── HomeAway

│ │ │ ├── Events

│ │ │ │ └── EventDescription1

│ │ │ │ └── EventDescription2

│ │ │ │ └── EventDescription3 ..

│ │ │ ├── PlayerEvents

│ │ │ │ ├── PlayerID

│ │ │ │ │ ├── Event

│ │ │ │ │ │ └── PushedNode1

│ │ │ │ │ │ └── PushedNode2 ..

│ │ │ ├── TeamStats

│ │ │ │ ├── Event

│ │ │ │ │ └── PushedNode1

│ │ │ │ │ └── PushedNode2 ..

Figure 4.26 – Matches Data Structure

* ‘Events’ will be used to hold Strings describing the event captured by the user. This will then be used to display real-time feedback within the Log Match Day screen.
* ‘PlayerEvents’ will be used for retrieving individual statistics when using the application
* ‘TeamStats’ will be used for collecting total statistics for a whole team when capturing match day data

**4.3.5 MESSAGES**  
This section will store all messages sent and received by all users. The structure of this data will be:

Figure 4.27 – Messages Data Structure

├── Messages

│ ├── CoachID

│ │ ├── PlayerID

│ │ │ ├── MessageID1

│ │ │ │ └── messageText

│ │ │ │ └── messageTime

│ │ │ │ └── messageUser

│ │ │ ├── MessageID2

│ │ │ │ └── messageText

│ │ │ │ └── messageTime

│ │ │ │ └── messageUser

**4.3.6 VIDEOS**

This section will store data needed to download videos uploaded using the application. The structure will be:

├── Videos

│ ├── CoachID

│ │ ├── Date

│ │ │ └── Video

Figure 4.28 – Videos Data Structure

**4.4 CONCLUSION**

With the design stage complete, it allows implementation of the application to begin. It is important to note that these designs are not final, these may change during the process of implementation, but this process has given insight into the effort required for each element.

This chapter will document the implementation of the proposed mobile application. It will consist of a walkthrough of the creation of the application with code snippets of the system’s most significant features included. Any issues encountered along with solutions are also included in this chapter.

CHAPTER 5 - IMPLEMENTATION

**5.1 DEVELOPMENT PLATFORM USED**

Table 5.1 presents an overview of the platform used to implement the application:

|  |  |  |  |
| --- | --- | --- | --- |
| HARDWARE | | SOFTWARE | |
| CPU | AMD Dual-Core Processor A9-9420  3.6GHz | OPERATING SYSTEM | Windows 10 |
| RAM | 8GB |
| HARD DRIVE | 1TB | DEVELOPMENT ENVIRONMENT | Android Studio,  Firebase |
| SCREEN | 1280x1024 |
| MOBILE DEVICES | Samsung Galaxy S5,  Moto C Plus |

Table 5.1 – Development Platforms

**5.2 DEVELOPMENT TECHNOLOGIES USED**

The technologies used in the development of the application are detailed below:

**5.2.1 ANDROID STUDIO**

The official Integrated Development Environment (IDE) for Android app development, based on IntelliJ IDEA. Android Studio offered features such as:

* A flexible Gradle-based build system
* Instant run to push changes to your running app without building a new APK
* Extensive testing tools and frameworks
* Built-in support for Google Cloud Platform, making it easy to integrate Google Cloud Messaging and App Engine

**5.2.2 FIREBASE**

Firebase will be used to implement the online database and authentication of the applications users. Features detailed online include:

* analytics, databases, messaging and crash reporting so you can move quickly and focus on your users
* Firebase is built on Google infrastructure and scales automatically, for even the largest apps

**5.3 IMPLEMENTATION STRATEGY**

In addition to the Gantt Chart, which can be found in Appendix C, a work breakdown structure was also created to track and monitor the implementation of the project, included in Appendix D.

As well as creating these, a basic plan of sequential tasks was created and can be seen below in Figure 5.1. This plan shows an overview of the order elements of the application will be developed.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| TIME | | | | | | | | | | | | | | | |
| Register (Coach) | Login (Coach) | Home (Coach) | My Account (Coach) | Add Players (Coach) | View Players (Coach) | Login (Player) | Home (Player) | My Account (Player) | Log Match Day (Coach) | View Team Stats  (Coach & Player) | View Player Stats  (Coach & Player) | Compare Team Performance  (Coach & Player) | Compare Player Performance  (Coach & Player) | Chat Facility  (Coach & Player) | Video Upload/View  (Coach & Player) |
| Database Creation/Editing | | | | | | | | | | | | | | | |

The Implementation Plan (Table 5.2) shows each element will be developed sequentially. One element will be completed before moving on to the next. Elements of displaying data for both Coach and Player can be developed parallel to each other as these functions will be very similar.

Table 5.2 – Implementation Plan

Player features can not begin until the coach can successfully add players to the system.

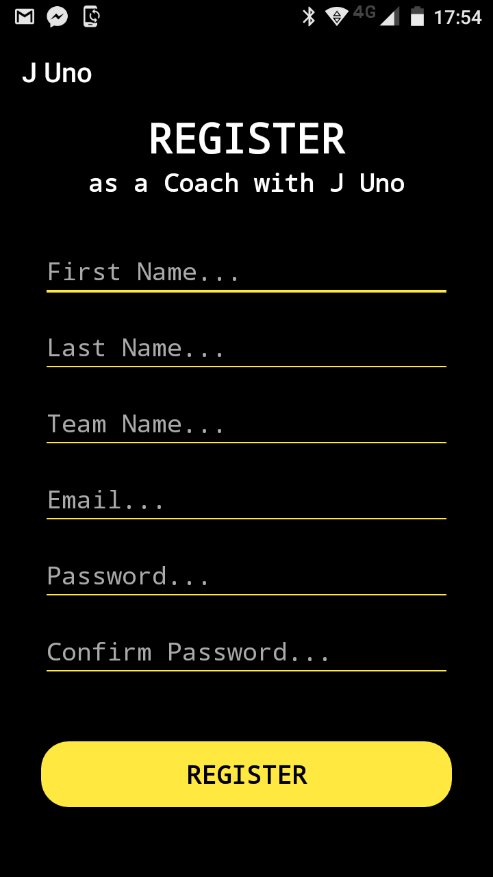
**5.4 COACH – REGISTER**

Requirement F01 states that ‘new coaches should be able to register a new account with the system’, and this is the first element to be implemented.

It is important to note that a blank activity for logging in was created when starting the project. This is used as the Launcher Activity and is what greets the user when first opening the application and will be used as the Login screen.

At first, only a ‘TextView’ was added to this screen to allow testing of the Register activity. The rest of the Login screen layout will be developed later in the implementation stage.

As specified in the design stage, the Registration screen will allow new users to enter their information and create an account which can later be used to log into the system.

Figure 5.1 presents the completed layout for the Registration screen. This layout allows the user to enter information required for setting up an account. This includes:

* First Name
* Last Name
* Team Name
* Email
* Password
* Confirm Password

This screen also includes a Register button. The user will click this button when all information is completed.

When beginning the implementation of authenticating a user within the Firebase Database, it was made apparent that code was required to make a reference to the database in question.

Figure 5.1 – Register Layout

This code is shown in Figure 5.2. This is included within all activities in the project to allow interaction with the online Firebase Database. The ‘.child()’ function allows developers to direct a reference to a specific node within the database. Using this knowledge, the code within Figure 5.2 is a reference for the Coaches section within the Users node.

Figure 5.2 – Database Reference Code

**mAuth** = FirebaseAuth.*getInstance*();  
**dbReference** = FirebaseDatabase.*getInstance*().getReferenceFromUrl(**"https://juno-97ab9.firebaseio.com/"**).child(**"Users"**).child(**"Coaches"**);

The ‘mAuth’ variable within Figure 5.2 is a shared instance of the FirebaseAuth object. This is required to facilitate Firebase’s ‘createUserWithEmailAndPassword’ function.

Before calling this function to authenticate a user, it was important to ensure all data was complete and accurate. Figure 5.3 contains the code used to ensure no field was left empty and that the email entered by the user follows the correct format. The function used for checking the email is contained within Figure 5.4.

Figure 5.3 – Register Data Verification Code

Figure 5.4 – Coach Email Verification Code

**public static boolean** isEmailValid(String email) {  
 String expression = **"^[\\w\\.-]+@([\\w\\-]+\\.)+[A-Z]{2,4}$"**;  
 Pattern pattern = Pattern.*compile*(expression, Pattern.***CASE\_INSENSITIVE***);  
 Matcher matcher = pattern.matcher(email);  
 **return** matcher.matches();  
}

**If** (!TextUtils.*isEmpty*(fName)&& !TextUtils.*isEmpty*(sName)&& !TextUtils.*isEmpty*(tName) && !TextUtils.*isEmpty*(email)&&!TextUtils.*isEmpty*(password1)&& !TextUtils.*isEmpty*(password2)){

**if** (*isEmailValid*(email) == **true**) {

**if** (password1.equals(password2)) {

**//authenticate user**

} **else**{  
 Toast.*makeText*(register.**this**, **"Passwords do not match"**, Toast.***LENGTH\_LONG***).show();  
}

} **else**{  
 Toast.*makeText*(register.**this**, **"Please enter a valid email address"**, Toast.***LENGTH\_LONG***).show();  
}

} **else**{  
 Toast.*makeText*(register.**this**, **"Please ensure all fields are complete"**, Toast.***LENGTH\_LONG***).show();  
}

Once all data is validated the email address and password is used to authenticate the user within the Firebase Database.

Using this function will not save user details such as their First Name, Last Name and Team Name, this is added to the database manually using another database reference. While adding this information, for every user, a default is set for their ‘Image’ and Team Badge, and their ‘UserType’ is set to ‘Coach’.

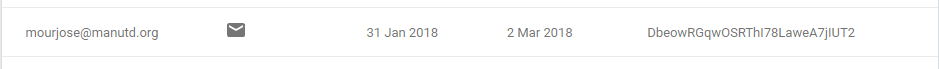
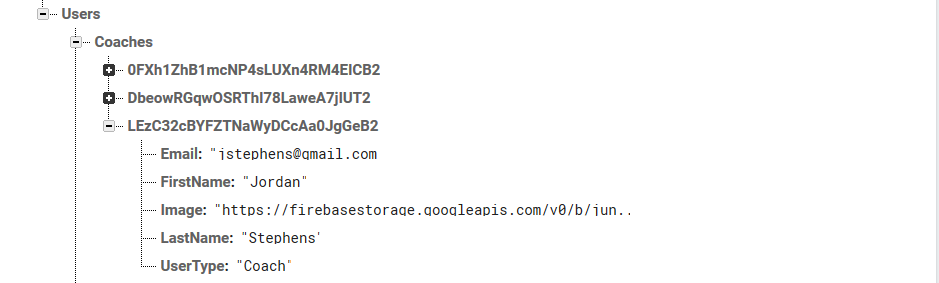
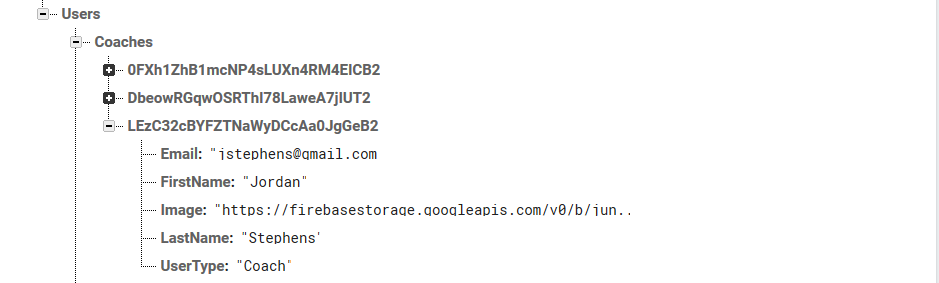
The code used to implement this is present in Figure 5.6. Below, in Figure 5.5, is an example of how data appears within the Firebase Console’s Authentication panel. Passwords are saved after being encrypted by Firebase. Figure 5.5 also shows that each user authenticated is given a unique identifier, this has been highlighted by a red box.

Figure 5.5 – Authentication Data Example



“default”

Figure 5.7 – Coach Data Example

Figure 5.6 – Authenticating a User Code

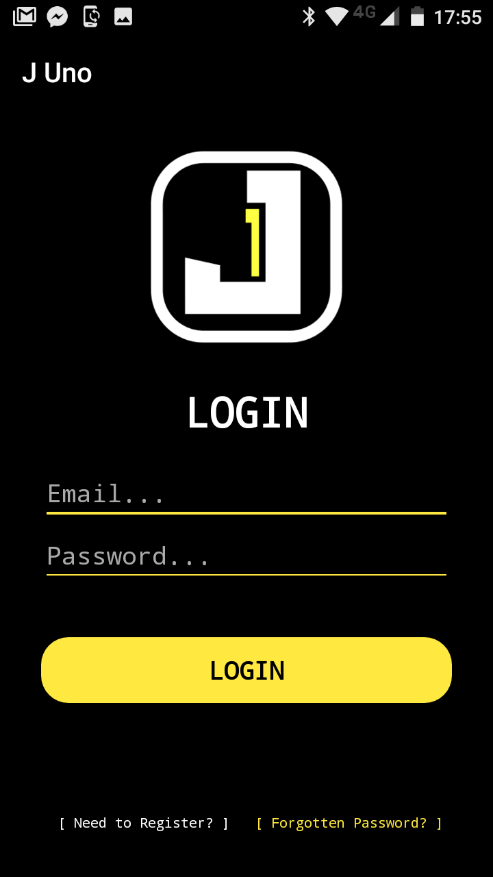
**//authenticate user**

**mProgress**.setMessage(**"Setting up Account..."**);  
**mProgress**.show();  
  
**mAuth**.createUserWithEmailAndPassword(email, password1).addOnCompleteListener(**new** OnCompleteListener<AuthResult>() {  
 @Override  
 **public void** onComplete(@NonNull Task<AuthResult> task) {  
 **if** (task.isSuccessful()) {  
  
 String userID = **mAuth**.getCurrentUser().getUid();  
  
 DatabaseReference currentUserDB = **dbReference**.child(userID);  
 DatabaseReference teamDB = **teamDBreference**.child(userID);  
  
 *//add to coaches db* currentUserDB.child(**"FirstName"**).setValue(fName);  
 currentUserDB.child(**"LastName"**).setValue(sName);  
 currentUserDB.child(**"Email"**).setValue(**mAuth**.getCurrentUser().getEmail());  
 currentUserDB.child(**"Image"**).setValue(**"default"**);  
 currentUserDB.child(**"UserType"**).setValue(**"Coach"**);  
  
 *//add to team db, coachid is team's id* teamDB.child(**"TeamName"**).setValue(tName);  
 teamDB.child(**"Badge"**).setValue(**"default"**);  
  
 **mProgress**.dismiss();  
  
 Intent coachIntent = **new** Intent(register.**this**, coach\_Home.**class**);  
 coachIntent.addFlags(Intent.***FLAG\_ACTIVITY\_CLEAR\_TOP***);  
 startActivity(coachIntent);

Figure 5.7 is an example of how data is stored and displayed within the Firebase Console’s Database Panel. Once a user has registered and been authenticated they are navigated to the Coaches home screen.

**5.5 COACH – LOGIN**

Requirement F02 states ‘Registered Coaches should be able to log in to the system’, so this was implemented next. The Login screen layout was completed, shown in Figure 5.8, before coding to begin.

 This screen is simple but allows a user to access many options. A user can:

* Enter their email and password and attempt to log in.
* Navigate to register as a user
* Request a reset email to be sent if they have forgotten their password.

If a user clicks the Login button the first thing that must happen is the ‘email’ and ‘password’ fields are checked to ensure they are not empty.

If all data is valid a log in attempt is started using the ‘mAuth.signInWithEmailAndPassword( )’ using the email and password entered by the user as the inputs.

If this log in attempt is successful, the coach is directed to the Coach Home Screen. Any errors during this process will be displayed to the user in the form of Toasts.

Figure 5.8 – Login Layout

If a user has forgotten their password, they can request a password reset email by selecting the appropriate option. An Alert Dialog is used to present the interface for entering an email address. Once an email address is entered an email is sent that contains a link to reset a password.

Figure 5.9 contains the code used for both logging a user into the application and requesting a password reset email.

Figure 5.9 – Login Code Examples

**mAuth**.signInWithEmailAndPassword(email, password).addOnCompleteListener(**new** OnCompleteListener<AuthResult>() {  
 @Override  
 **public void** onComplete(@NonNull Task<AuthResult> task) {  
 **if** (task.isSuccessful()) {  
 checkUserExist();  
  
  
 } **else** {  
 **mProgress**.dismiss();  
 Toast.*makeText*(Login.**this**, **"Incorrect Details. Try Again"**, Toast.***LENGTH\_LONG***).show();  
 }  
 }  
});

**\_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_**

**mAuth**.sendPasswordResetEmail(emailToBeReset);

During implementation it was discovered that automatic sign-in was possible. This was implemented using a ‘FirebaseAuth.AuthStateListener( )’. The code to implement this is contained within Figure 5.10.

Figure 5.10 – Auto-Login Code

**mAuthListener** = **new** FirebaseAuth.AuthStateListener() {  
 @Override  
 **public void** onAuthStateChanged(@NonNull FirebaseAuth firebaseAuth) {  
  
 **if** (firebaseAuth.getCurrentUser() == **null**) {  
 ***// user not logged in***  
 } **else** {  
 ***//direct user***  
  
 }  
 }

This code simply uses FirebaseAuth to get Current User data, if this function returns with ‘null’ it can only mean no one is currently logged in on that device and no action is taken. If it returns anything else the Coach is directed to their Home screen. This section of code becomes more important when Players are added by the coach.

For this function to operate correctly an ‘onStart( )’ method was created which can be seen in Figure 5.11.

Figure 5.11 – onStart code for Auto Login

@Override  
**protected void** onStart() {  
 **super**.onStart();  
 **mAuth**.addAuthStateListener(**mAuthListener**);  
}

**5.6 COACH – HOME**

This is the first screen with personalised information, and as acknowledged in Chapter 4, the Home screen for the coach will allow them to view and interact with their Team Calendar. This relates to requirement F08 which states ‘Coaches should be able to view their team calendar’.

Before implementing the calendar, it was crucial to implement a way of gathering user data. When using Firebase, the easiest method of collecting batches of data from a node is by creating a separate Java Class which contains a constructor, along with ‘Getter and Setters’ for each piece of data required. Simply put, a Java Class was required to collect and store the data. The Java class for collecting a coach’s details is presented in Figure 5.13.

Figure 5.12 – Coach Home Layout

Figure 5.13 – UserInformation Java Class Code

**public class** UserInformation {  
 **private** String **FirstName**;  
 **private** String **LastName**;  
 **private** String **Image**;  
 **private** String **UserType**;  
 **private** String **Email**;  
  
 **public** UserInformation() {  
 }  
  
 **public** String getFirstName() {  
 **return FirstName**;  
 }  
  
 **public void** setFirstName(String firstName) {  
 **FirstName** = firstName;  
 }  
  
 **public** String getLastName() {  
 **return LastName**;  
 }  
  
 **public void** setLastName(String lastName) {  
 **LastName** = lastName;  
 }  
  
 **public** String getImage() {  
 **return Image**;  
 }  
  
 **public void** setImage(String image) {  
 **Image** = image;  
 }  
  
 **public** String getUserType() {  
 **return UserType**;  
 }  
  
 **public void** setUserType(String userType) {  
 **UserType** = userType;  
 }  
  
 **public** String getEmail() {  
 **return Email**;  
 }  
  
 **public void** setEmail(String email) {  
 **Email** = email;  
 }  
}

This Java class, ‘UserInformation’, can then be referenced inside an activity class to retrieve data. See Figure 5.14 for the code used to reference this Java class and store the results in variables.

This code will be repeated on every activity used by a coach to retrieve the user’s data and populate the Navigation Menu interface. The code snippet within Figure 5.14 includes the method updateMenu( ), this method updates the Navigation Menu which was the first instance of utilising user data.

Figure 5.14 – Retrieving User Data Example Code

**dbReferenceUsers**.addValueEventListener(**new** ValueEventListener() {  
 @Override  
 **public void** onDataChange(DataSnapshot dataSnapshot) {  
 **if** (dataSnapshot.hasChild(**user\_id**)) {  
 *//get data* UserInformation uInfo = **new** UserInformation();  
 uInfo.setFirstName(dataSnapshot.child(**user\_id**).getValue(UserInformation.**class**).getFirstName());  
 uInfo.setLastName(dataSnapshot.child(**user\_id**).getValue(UserInformation.**class**).getLastName());  
 uInfo.setImage(dataSnapshot.child(**user\_id**).getValue(UserInformation.**class**).getImage());  
  
 **c\_nav\_FirstName** = uInfo.getFirstName();  
 **c\_nav\_LastName** = uInfo.getLastName();  
 **disImage** = uInfo.getImage();  
 **c\_nav\_Email** = **mAuth**.getCurrentUser().getEmail();  
  
 updateMenu(**c\_nav\_FirstName**, **c\_nav\_LastName**, **c\_nav\_Email**, **disImage**);  
 }  
 }  
  
 @Override  
 **public void** onCancelled(DatabaseError databaseError) {  
  
 }  
});

**5.6.1 COACH NAVIGATION MENU**

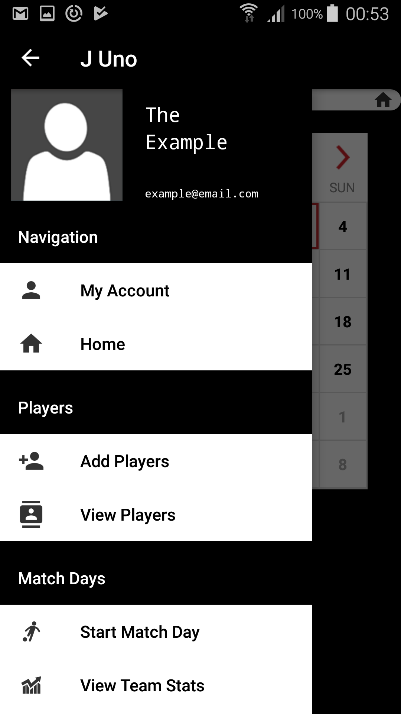
This menu is available to the user on all screens, allowing them access to all facilities from any screen.

Figure 5.15 – Coach Menu Layout

The design consists of a User Information Banner, followed by each navigation option which has been divided into logical sections.

The updateMenu( ) method within Figure 5.14 uses data retrieved and stored in variables to update the Banner with the correct user’s details.

Each user will have the option to upload a profile picture, this facility will be implemented once the Home screen is completed.

The functionality of this navigation bar was implemented using a switch-case scenario. This can be seen within Figure 5.16.

Figure 5.16 – Navigation Bar Code

@Override  
**public boolean** onNavigationItemSelected(@NonNull MenuItem item) {  
 *//handle navigation view item clicks* **switch**(item.getItemId()) {  
  
 *//navigation* **case** R.id.***nav\_my\_account***: {  
 Intent c\_account = **new** Intent(coach\_Home.**this**, coach\_setup.**class**);  
 c\_account.addFlags(Intent.***FLAG\_ACTIVITY\_CLEAR\_TOP***);  
 startActivity(c\_account);  
 **break**;  
 }  
  
 **case** R.id.***nav\_home***: {  
 Intent c\_home = **new** Intent(coach\_Home.**this**, coach\_Home.**class**);  
 c\_home.addFlags(Intent.***FLAG\_ACTIVITY\_CLEAR\_TOP***);  
 startActivity(c\_home);  
 **break**;  
 }**...**

The code within Figure 5.16 identifies which tab was selected and begins an activity to navigate the user to the appropriate screen.

Figure 5.17 highlights the code used to upload a user’s profile photo into the Navigation Bar. This element was implemented using a API named Picasso which downloads and scales the image from Firebase and places inside the ImageView within the Navigation Bar. A progress bar is shown until this process is complete.

If there is an error whilst obtaining the profile picture, the ImageView reverts back to the default image and an error message is displayed to the user.

Figure 5.17 – Navigation Bar Profile Photo Code

**Picasso.*with*(coach\_Home.this).load(Uri.*parse*(profile)).into(navProfile, new Callback() {  
 @Override  
 public void onSuccess() {  
 mProgress.setVisibility(View.*GONE*);  
  
 }  
  
 @Override  
 public void onError() {  
 navProfile.setImageResource(R.drawable.*def*);  
 Toast.*makeText*(coach\_Home.this, "Error Loading Profile Photo", Toast.*LENGTH\_LONG*).show();  
 }**

Now that a method has been created to retrieve user data, and that data has been used successfully the Coaches Home screen could now be completed.

**5.6.2 CALENDAR IMPLEMENTATION**

The Calendar for this screen was added using and API named Caldroid. After some research, this was the best option for the functionality required for the calendar.

When a user selects either button on their Home screen (Add Match Day or Add Training Session) a DatePickerDialog is presented. This is an Android default method of collecting a Date object. This method ensures no errors are encountered when entering a date and that it follows the correct format to be used in the Calendar.

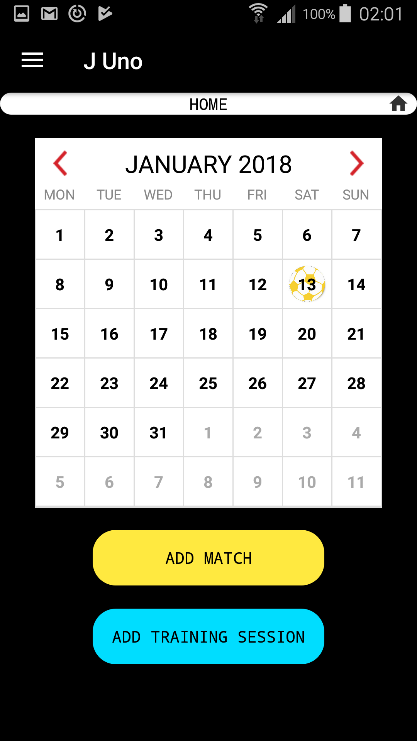
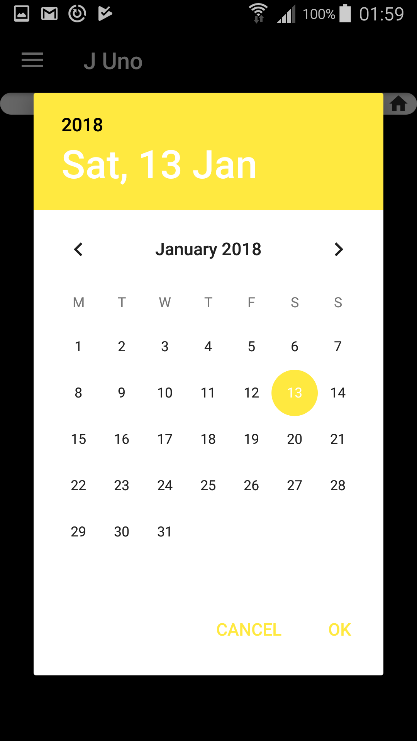


Figure 5.18 shows the basic interfaces a coach will use to add a date to their team calendar. One a Date is selected it is identified within the team calendar using an appropriate icon. Although this seems like a relatively easy process, the implementation of it proved quite difficult. Some of the issues with this process were:

Figure 5.18 – Adding Dates to Calendar

* How to save the dates within the database (“/” can not be used in the naming of nodes within the database)
* How to convert from a Date object, returned by the DatePicker, to a String which can be stored in the database
* How to convert a String version of a date back to a Date object when loading a user’s calendar

Figure 5.19 – Collecting Date Selected Code Snippet

**private** DatePickerDialog.OnDateSetListener **dpickerListener** = **new** DatePickerDialog.OnDateSetListener() {  
 @Override  
 **public void** onDateSet(DatePicker view, **int** year, **int** month, **int** dayOfMonth) {  
  
 Calendar cal = Calendar.*getInstance*();  
 cal.set(Calendar.***YEAR***, year);  
 cal.set(Calendar.***MONTH***, month);  
 cal.set(Calendar.***DATE***, dayOfMonth);  
  
 **date** = cal.getTime(); *// get date object* **chosenDate** = dayOfMonth + **"/"** + (month+1) + **"/"** + year;*//get date string*

**...**}

Figure 5.20 – Add Date Code Snippet

**...**

**final** DatabaseReference matchDays = **dbReferenceDates**.child(**user\_id**).child(**"MatchDays"**);  
matchDays.addListenerForSingleValueEvent(**new** ValueEventListener() {  
 @Override  
 **public void** onDataChange(DataSnapshot dataSnapshot) {  
 **if** (dataSnapshot.hasChildren()) {  
 Map<String, Object> td = (HashMap<String, Object>) dataSnapshot.getValue();  
  
 List<Object> values = **new** ArrayList<>(td.values()); **//create list of dates**  
 **if** (!values.contains(**chosenDate**)) { **//check if date exists**  
 addMatchDay(**chosenDate**, **date**); **//add date to database**  
 } **else** {  
 Toast.*makeText*(coach\_Home.**this**, **"Error:"** + **"\n"** + **"Match Day "** + **chosenDate** + **" already exists"**, Toast.***LENGTH\_LONG***).show();  
  
 }  
 }  
 **else** {  
 addMatchDay(**chosenDate**, **date**); **//add date to database**  
 }  
 }  
  
 @Override  
 **public void** onCancelled(DatabaseError databaseError) {  
  
 }  
});

Figure 5.19 shows how the date selected is collected from the resulting object. It is gathered in separate elements (day, month, year) and then combined to create a recognisable date format.

Figure 5.20 highlights that error handling is also required in this process. When adding a date, first a list of current dates is created. This list is checked for the ‘chosenDate’ and if it is present an error is displayed to the user.

If everything is correct, the ‘addMatchDay( )’ method is called.

Figure 5.21 – Add Match Day Code

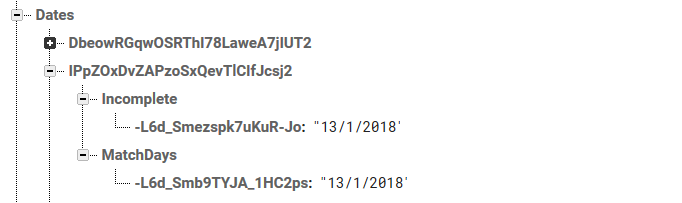
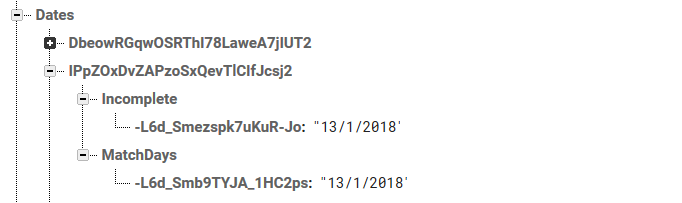
**private void** addMatchDay(String chosenDate, Date setDate) {  
  
 DatabaseReference currentUserDB = **dbReferenceDates**.child(**user\_id**).child(**"MatchDays"**);  
 currentUserDB.push().setValue(chosenDate);  
 DatabaseReference currentuserCompMatch = **dbReferenceDates**.child(**user\_id**).child(**"Incomplete"**);  
 currentuserCompMatch.push().setValue(chosenDate);  
  
 **caldroidFragment**.refreshView();  
  
}

This code snippet (Figure 5.21) shows the ‘chosenDate’ being added to the database in two places:

* MatchDays – used for retrieving dates for displaying in users Calendar
* Incomplete – used to differentiate user Match Days later in the application

When added to the database, the date can be viewed in Firebase as shown in Figure 5.22.

Figure 5.22 – Date added Example



An updateCalendar( ) method is called within this activity’s onCreate( ) to load all existing dates for user into their calendar. Within this method a similar technique used for validating the adding of a date is used.

First, a list of all dates within the database is created. A for-loop is then used to perform tasks on each item within the list. The task performed in this instance is converting the date String into an object which can be used by the Caldroid fragment (the calendar). This process can be seen in Figure 5.23.

Figure 5.23 – Loading Dates to Calendar

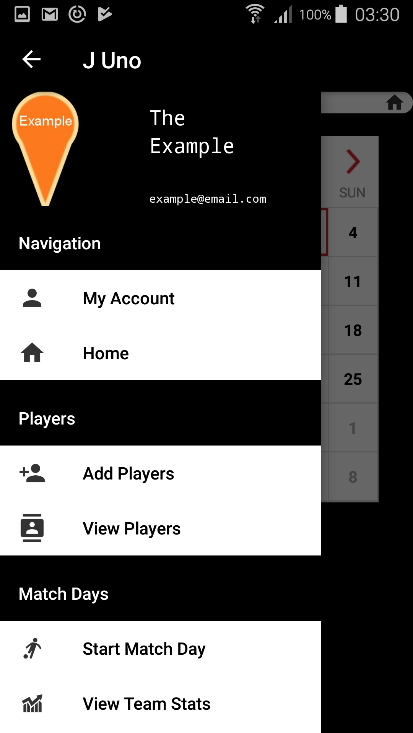
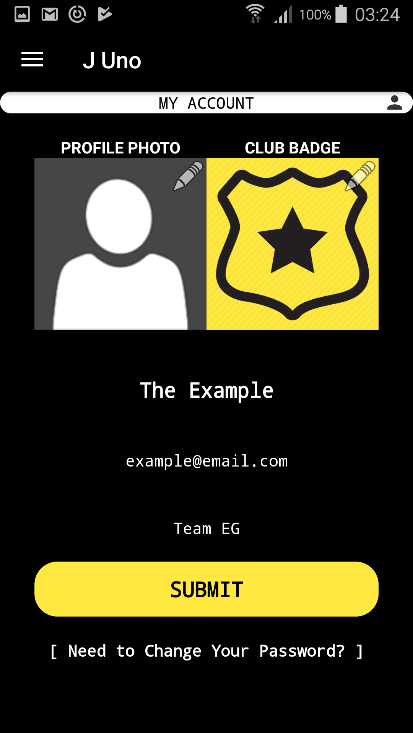
String a = values.get(i).toString() + **" 07:00:00"**; *//time added to help conversion*SimpleDateFormat format = **new** SimpleDateFormat(**"dd/MM/yyyy HH:mm:ss"**);  
**try** {  
 **newdate** = format.parse(a);  
} **catch** (ParseException e) {  
 *//* ***TODO Auto-generated catch block*** e.printStackTrace();  
}

**caldroidFragment**.setBackgroundDrawableForDate(**drawableTrain**, **newdate**);

The calendar is refreshed after all icons are added to display all dates added by the coach.

**5.7 COACH – MY ACCOUNT**

The requirement F04 states ‘Coaches should be able to upload a profile photo’, this activity aims to fulfil this requirement. As stated within the design stage, this screen will display the user’s information and, most importantly, allow them to upload a profile photo and team badge. This activity also includes a facility that allows users to change their password.



**My Account Layout**

**Completed Screen**

**Updated Menu**

Figure 5.24 – Coach My Account Screen Examples

To implement this activity a user’s Team information was needed. This meant the creation of another Java class to retrieve specific data. This Java class can be seen in Figure 5.25.

Figure 5.25 – Team Information Java Class Code

**public class** TeamInformation {  
  
 **private** String **TeamName**;  
 **private** String **Badge**;  
  
 **public** TeamInformation() {  
 }  
  
 **public** String getTeamName() {  
 **return TeamName**;  
 }  
  
 **public void** setTeamName(String teamName) {  
 **TeamName** = teamName;  
 }  
  
 **public** String getBadge() {  
 **return Badge**;  
 }  
  
 **public void** setBadge(String badge) {  
 **Badge** = badge;  
 }  
}

Using the same code detailed in Figure 5.14 to gather user information, a similar technique was used for Team information. Once difference included in this activity is that and ‘if’ statement checks whether the user’s ‘Image’ and Team’s ‘Badge’ is “default”.

If these values are not “default” the value is retrieved, and the image is downloaded into the appropriate ImageButtons (Figure 5.26).

Figure 5.26 – Checking Profile Image

**//performed while gathering userInformation**

**if** (disImage.equals(**"default"**)) {  
 *//do nothing*} **else** {  
 **imgProg**.setVisibility(View.***VISIBLE***);  
 Picasso.*with*(coach\_setup.**this**).load(Uri.*parse*(disImage)).into(**imgProfile**, **new** Callback() {  
 @Override  
 **public void** onSuccess() {  
 **imgProg**.setVisibility(View.***GONE***);  
 }  
  
 @Override  
 **public void** onError() {  
 **imgProfile**.setImageResource(R.drawable.***def***);  
 **imgProg**.setVisibility(View.***GONE***);  
 Toast.*makeText*(coach\_setup.**this**, **"Error Loading Profile Badge"**, Toast.***LENGTH\_LONG***).show();  
 }  
 });  
  
}

When a user selects to upload a profile picture or club badge 3 processes take place. The first is shown in Figure 5.27, it highlights the implementation of a ProGalleryIntent, this is a subset function of Picasso which is an API which has been mentioned previously.

Figure 5.27 – ProGalleryIntent Code

**imgProfile**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
  
 Intent ProGalleryIntent = **new** Intent();  
 ProGalleryIntent.setAction(Intent.***ACTION\_GET\_CONTENT***);  
 ProGalleryIntent.setType(**"image/+"**);  
 startActivityForResult(ProGalleryIntent, ***GALLERY\_REQUEST\_CODE***);  
  
 **ButtonPressed** = 1;  
 }  
});

This intent starts the process of selecting an image. It opens a dialog displaying the folder options to the user. Once an image is selected the second process begins.

The second and third process are contained within an onActivityResult( ) method. Once an image is selected by a user, the second process allows the user to crop the image and restricts the shape of the image to be a square.

The third process stores the final cropped image and a Uri and returns it as a usable variable. This method can be seen in Figure 5.28 (only code for a profile picture change has been included, the ‘buttonPressed’ variable is used as an identifier when both ImageButtons are implemented).

Figure 5.28 – Selecting a Profile Picture Code

**protected void** onActivityResult(**int** requestCode, **int** resultCode, Intent data) {  
 **super**.onActivityResult(requestCode, resultCode, data);

**//second process – cropping image into square**  
 **if** (requestCode == ***GALLERY\_REQUEST\_CODE*** && resultCode == ***RESULT\_OK***) {  
 **if** (**ButtonPressed** == 1) {  
 **mImageUri** = data.getData();  
 CropImage.*activity*()  
 .setGuidelines(CropImageView.Guidelines.***ON***)  
 .setAspectRatio(1, 1)  
 .start(**this**);  
 }  
 }

**//third process – storing cropped image and returing as uri**  
 **if** (requestCode == CropImage.***CROP\_IMAGE\_ACTIVITY\_REQUEST\_CODE***) {  
 CropImage.ActivityResult result = CropImage.*getActivityResult*(data);  
 **if** (resultCode == ***RESULT\_OK***) {  
 **if** (**ButtonPressed** == 1) {  
 Uri resultUri = result.getUri();  
 **imgProfile**.setImageURI(resultUri);  
 **mImageUri** = resultUri;  
 }  
  
 **else if** (resultCode == CropImage.***CROP\_IMAGE\_ACTIVITY\_RESULT\_ERROR\_CODE***) {  
 Exception error = result.getError();  
 }  
 }  
  
}

Only when a user selects the ‘Submit’ button are these changes made permanent. When the button is clicked variables for the profile picture and Badge photo are checked for content. If either is not empty another ‘if’ statement checks both individually.

Boolean variables are used to track the progress of each process and only when both are complete will the user be redirected back to their Home screen.

An example of one of these processes can be seen in Figure 2.29.

Figure 5.29 – Uploading Profile Picture Code

task1Comp = **false**;  
**mProgress**.setMessage(**"Uploading..."**);  
**mProgress**.show();  
StorageReference filepath = **mStorageImage**.child(**mImageUri**.getLastPathSegment());  
filepath.putFile(**mImageUri**).addOnSuccessListener(**new** OnSuccessListener<UploadTask.TaskSnapshot>() {  
 @Override  
 **public void** onSuccess(UploadTask.TaskSnapshot taskSnapshot) {  
  
 String downloadUri = taskSnapshot.getDownloadUrl().toString();  
 **dbReferenceUsers**.child(user\_id).child(**"Image"**).setValue(downloadUri).addOnCompleteListener(**new** OnCompleteListener<Void>() {  
 @Override  
 **public void** onComplete(@NonNull Task<Void> task) {  
 **mProgress**.dismiss();  
 task1Comp = **true**;  
 }

Figure 5.29 shows a Storage filepath being declared. This is essentially a database reference that refers to an area to store image/video data. This code also includes an ‘OnSuccessListener’ which allows a progress bar to appear for the duration of the upload, ensuring the user does not unintentionally cancel it. If an error is encountered an error message is displayed to the user.

If a user needs to change their password the simply click the text at the bottom of the screen. This will display a Alert Dialog (Figure 5.39) which asks them to confirm their current password before asking them to enter their new password twice.

Again, once the button is selected the fields are checked to ensure they are not empty. Once this is confirmed the ‘New Password’ and ‘Confirm Password’ and compared.

Figure 5.30 – Alert Dialog 1

If these match the process of changing the user’s password can begin. Figure 5.31 contains the code used to implement this.

Figure 5.31 – Changing Password Code

mProgress.setMessage(**"Attempting Change..."**);  
mProgress.show();  
AuthCredential authCredential = EmailAuthProvider.*getCredential*(**mAuth**.getCurrentUser().getEmail(), oPassword);  
firebaseUser.reauthenticate(authCredential).addOnSuccessListener(**new** OnSuccessListener<Void>() {  
 @Override  
 **public void** onSuccess(Void aVoid) {  
 mProgress.cancel();  
 firebaseUser.updatePassword(nPassword);  
 Toast.*makeText*(coach\_setup.**this**, **"Password updated"**, Toast.***LENGTH\_LONG***).show();  
  
 }  
});  
firebaseUser.reauthenticate(authCredential).addOnFailureListener(**new** OnFailureListener() {  
 @Override  
 **public void** onFailure(@NonNull Exception e) {  
 mProgress.cancel();  
 Toast.*makeText*(coach\_setup.**this**, **"Error Authenticating account"**, Toast.***LENGTH\_LONG***).show();  
  
 }  
});

The code within Figure 5.31 shows that the application will attempt to reauthenticate the user by their email (gathered programmatically) and the ‘Current Password’ entered in the Alert Dialog.

If this reauthentication is successful, the ‘firebaseUser.updatePassword(nPassword)’ method updates Firebase with the users new password.

If unsuccessful, this means the ‘Current Password’ entered is incorrect and an appropriate error message is displayed.

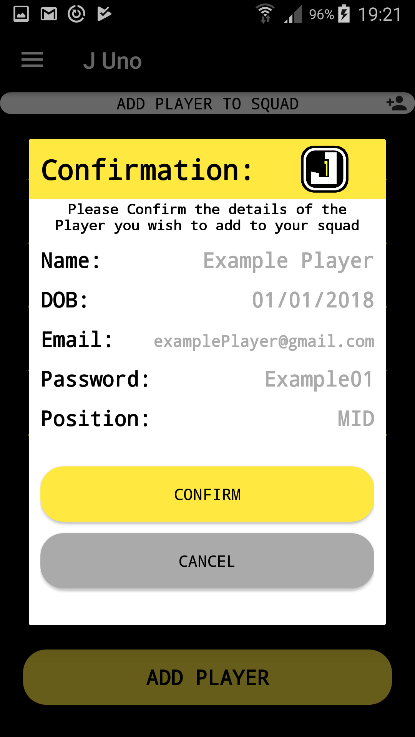
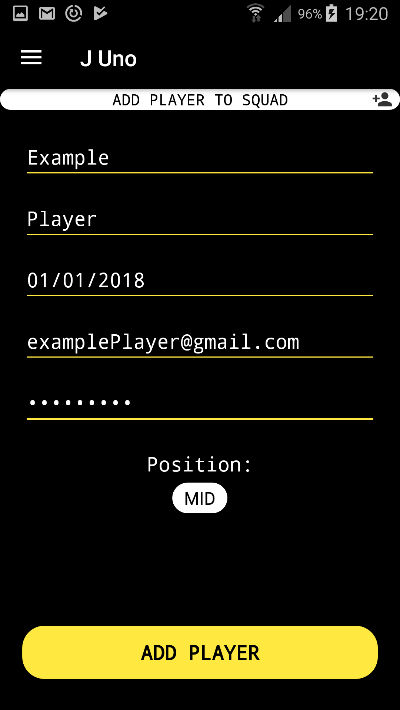
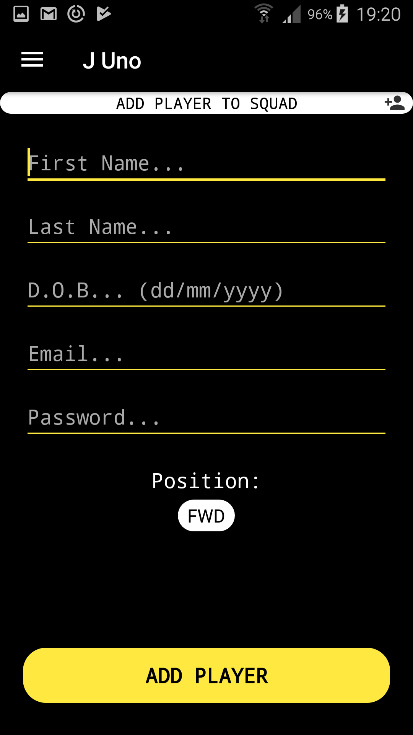
**5.8 COACH – ADD PLAYERS**

The next requirement to be fulfilled is F06, this requirements states ‘Coaches should be able to add new Team members to the system’. Firebase provides a Admin SDK to manage users within the database.

This SDK would be required as the method createUserWithEmailAndPassword( ) automatically signs that user in. This would not be suitable as the coach would have to log out as the player they just created and re-authenticate themselves with every player they add.

Unfortunately, when trying to implement this SDK into the application there was an error when trying to build the gradle file after adding the admin dependency. This error was investigated, but all resolutions were unsuccessful. A work-around was required.

Figure 5.32 – Adding Player Screen Examples



The work around that was implemented was creating a separate FirebaseAuth instance. To do this a separate instance of the app was created with certain options to accommodate the FirebaseAuth. Figure 5.33 displays the code for creating a new FirebaseAuth instance names mAuth2.

Figure 5.33 – Creating a new FirebaseAuth Instance Code

***//////code needed to bypass auto sign-in with mAuth*****firebaseOptions** = **new** FirebaseOptions.Builder()  
 .setDatabaseUrl(**"https://juno-97ab9.firebaseio.com/"**)  
 .setApiKey(**"AIzaSyBzvLW9P5SCBGgZuzvz9uOblS95cYnjFEg"**)  
 .setApplicationId(**"juno-97ab9"**).build();  
  
**myApp** = FirebaseApp.*initializeApp*(getApplicationContext(), **firebaseOptions**,a);  
  
  
  
**mAuth2** = FirebaseAuth.*getInstance*(**myApp**);

When a user wishes to add a player, they enter the players information, this includes their name, date of birth and their email. The coach also creates the player’s password which can later be changed once a player has logged in successfully.

When a coach has entered the details and clicked the ‘Add Player’ button all fields are checked to ensure they are not empty. An alert dialog is shown which confirms all details entered by the coach to ensure all information is correct and that the Password was entered correctly.

If these details are confirmed the information is taken and an account is created for that player using the new FirebaseAuth variable – mAuth2.createUserWithEmailAndPassword( ).

As well as authenticating the new user’s details, their personal details are added to the Players section of the database with their ‘Image’ value set to ‘default’. A Team ID is assigned to every player added and this corresponds to their coaches userID.

When created it is required that this user is signed out. This is done by calling the mAuth2.signout( ) method. Once all components are complete the coach is returned to their Home screen with a Toast confirming the player has been added.

Any errors during this process will evoke an appropriate error message.

Figure 5.34 – Adding Player Details Code

String userID = **mAuth2**.getCurrentUser().getUid();  
String playerEmail = **mAuth2**.getCurrentUser().getEmail();  
  
DatabaseReference currentUserDB = **dbReferencePlayers**.child(userID);  
currentUserDB.child(**"FirstName"**).setValue(playFname);  
currentUserDB.child(**"LastName"**).setValue(playLname);  
currentUserDB.child(**"DOB"**).setValue(playDOB);  
currentUserDB.child(**"Position"**).setValue(playPos);  
currentUserDB.child(**"Email"**).setValue(playerEmail);  
currentUserDB.child(**"UserType"**).setValue(**"Player"**);  
currentUserDB.child(**"Image"**).setValue(**"default"**);  
currentUserDB.child(**"TeamID"**).setValue(**user\_id**);  
currentUserDB.child(**"Badge"**).setValue(**badgeImage**);

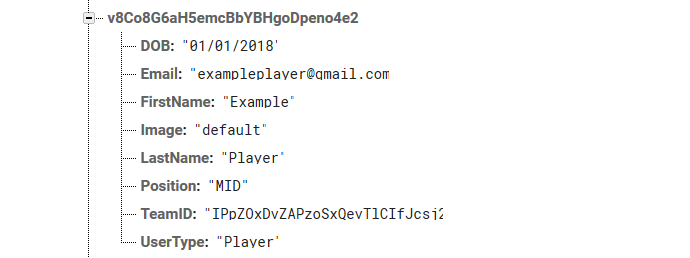
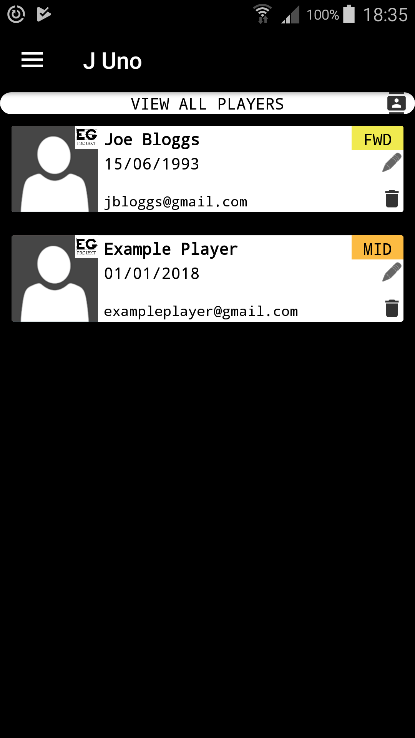


Figure 5.35 – Player Details within Firebase Console

**5.9 COACH – VIEW PLAYERS**

This screen is in addition to the requirements set out in Chapter 3. This screen allows coaches to view a full list of their players, alongside their current profile picture and club badge.

There are buttons to the right of each players information allowing the user to edit the selected players information or delete that player from their squad.

This facility seemed to be an important addition to the application as it can eliminate human error when adding players to the system. Only coaches can change players information and players will need to request this if it is required.

When implementing these facilities, it became apparent that due to the error with using Firebase’s Admin SDK, certain information would not be able to be changed. The information that cannot be changed once saved to the system is the player’s email address and their password.

Figure 5.36 – View Players Layout

The reason for this is that without the Admin SDK the methods used to change these details require the user to be signed in. There was no resolution found for this problem during implementation.

A second error that occurred was that a player could not be removed from the authentication service of Firebase without being signed in. The delete function simply deletes the player from their squad, meaning they will no longer appear as an option throughout the application.

A major issue with this error is that the player may still be able to log into the system even after being deleted by their coach. This will be tested when implementing the player log in.

When implementing this activity a CardView was used to display each individual player’s information (Figure 5.37). This CardView is then utilised within a RecyclerView to create a list of all player belonging to the coach.



Figure 5.37 – CardView Layout

To populate the RecyclerView an ‘onStart( )’ method was created which would be evoked when the activity began. This method would declare the adapter required to retrieve the data which would include a ‘populateViewHolder’ function that places the data into the card view as required. Figure 5.38 displays the code used to populate the list of players.

From the code contained within Figure 5.38 it can be seen that both buttons (Edit Player and Delete Player) are declared and ‘OnClickListeners’ are set at this stage. The code ‘getRef(position).getKey( )’ is used to distinguish the selected player’s ID.

Figure 5.38 – Populating Player List Code

@Override  
 **protected void** populateViewHolder(playersViewHolder viewHolder, players model, **int** position) {  
  
 **final** String player\_id = getRef(position).getKey();  
 **final** String player\_name = model.getFirstName() + **" "** + model.getLastName();  
 **final** String fName = model.getFirstName();  
 **final** String lName = model.getLastName();  
 **final** String dob = model.getDOB();  
  
 viewHolder.setName(model.getFirstName(), model.getLastName());  
 viewHolder.setPos(model.getPosition());  
 viewHolder.setDOB(model.getDOB());  
 viewHolder.setImage(getApplicationContext(), model.getImage());  
 viewHolder.setBadge(getApplicationContext(), **disBadge**);  
 viewHolder.setEmail(model.getEmail());  
  
 viewHolder.**playerListDelete**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 deleteplayer(player\_name, player\_id);  
 }  
 });  
  
 **if**(**playerDeleted** == **true**){  
 notifyItemRemoved(position);  
  
 **playerDeleted** = **false**;  
 }  
  
 viewHolder.**getPlayerListEdit**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 editplayer(player\_id, fName, lName, dob);  
 }  
 });  
  
 }  
};

Figures 3.39 and 3.40 show the code used to delete a player and edit a player’s information, respectively.

Figure 5.39 – Deleting a Player Code.

delplay.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
  
 *//delete player* **dbReferencePlayers**.child(player\_id).removeValue();  
 **playerDeleted** = **true**;  
 dialog.dismiss();  
 }  
});

Deleting a player was easily implemented, the method ‘removeValue( )’ is used to delete the player’s information from the database using their ID as the reference to the node to delete, The Boolean ‘playerDeleted’ within Figure 5.39 is used to trigger the update of the RecyclerView.

Figure 5.40 – Editing a Player Code.

editplay.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 *//get values* String newFname = playFname.getText().toString().trim();  
 String newLname = playLname.getText().toString().trim();  
 String newDOB = playDOB.getText().toString().trim();  
  
 *//edit player details* **dbReferencePlayers**.child(player\_id).child(**"FirstName"**).setValue(newFname);  
 **dbReferencePlayers**.child(player\_id).child(**"LastName"**).setValue(newLname);  
 **dbReferencePlayers**.child(player\_id).child(**"DOB"**).setValue(newDOB);  
 dialog.dismiss();  
 }  
});

Editing a player’s information was implemented by using the player’s ID as a ‘child’ within a database reference. Each field within the alert dialog, provided to enter the changes, is converted into a String variable and sent to the Firebase Database using the ‘setValue( )’ method.

**5.10 PLAYER – LOGIN**

Players can log in once given their credentials by their coach. They use the same screen as the coach (Figure 5.8) but will need directed to a separate Home screen. To do this code was added to the Login java file. This code can be seen in Figure 5.41, for this example the code is placed within the ‘else’ statement contained within Figure 5.10.

This code snippet is used for the automatic log in function. If a user is currently logged in the application firstly collects their ID using ‘mAuth.getCurrentUser.getUid’. Once the ID is established the application checks the coaches’ node for a matching ID.

If it matches an ID within the Coaches’ node it collects more user data before directing them to the Coach Home screen. If the ID is not a match the user must be a Player and is directed to the Player Home screen.

Figure 5.41 – Directing Different Users Code

*//direct user***final** String userID = **mAuth**.getCurrentUser().getUid();  
  
  
*//check if coach***dbReference**.child(**"Coaches"**).addValueEventListener(**new** ValueEventListener() {  
 @Override  
 **public void** onDataChange(DataSnapshot dataSnapshot) {  
 **if** (dataSnapshot.hasChild(userID)) {  
  
 Intent cIntent = **new** Intent(Login.**this**, coach\_Home.**class**);  
 cIntent.addFlags(Intent.***FLAG\_ACTIVITY\_CLEAR\_TOP***);  
 startActivity(cIntent);  
  
 **mProgress**.dismiss();  
  
 }  
 **else** {  
 *//must be player* Intent pIntent = **new** Intent(Login.**this**, player\_Home.**class**);  
 pIntent.addFlags(Intent.***FLAG\_ACTIVITY\_CLEAR\_TOP***);  
 startActivity(pIntent);  
  
 **mProgress**.dismiss();  
 }  
 }  
  
 @Override  
 **public void** onCancelled(DatabaseError databaseError) {  
 **loggedIn** = **false**;  
 }  
});

**5.11 PLAYER – HOME**

Once logged in a player is greeted by their team calendar. A calendar ‘key’ is also included to provide clarification on the icons that may appear. The calendar is updated using dates added by their coach.

This uses the same code as used to update a coach’s calendar, but variables and context are different. A method is called whilst gathering player information and their Team ID is initialised.

This team ID will be used to reference nodes within the database as this is equal to their coach’s user ID.

It is also important to note that, although the player’s navigation bar uses the same format/layout, they will have access to less options than a coach. They will not have the option to add/view players and cannot log match day information.

Figure 5.42 – Player Home Layout

The navigation bar is populated using the same method as used in the coach’s version.

Figure 5.43 – Updating Player Calendar Code

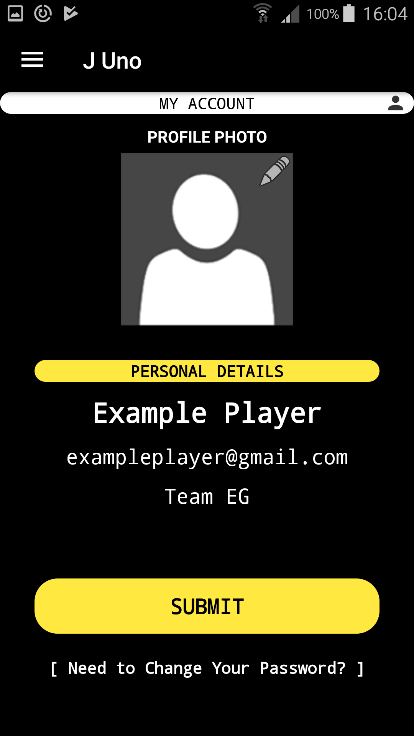
**dbReferenceUsers**.addValueEventListener(**new** com.google.firebase.database.ValueEventListener() {  
 @Override  
 **public void** onDataChange(com.google.firebase.database.DataSnapshot dataSnapshot) {  
 **if** (dataSnapshot.hasChild(**user\_id**)) {  
 PlayerInfo pInfo = **new** PlayerInfo();  
 pInfo.setFirstName(dataSnapshot.child(**user\_id**).getValue(PlayerInfo.**class**).getFirstName());  
 pInfo.setLastName(dataSnapshot.child(**user\_id**).getValue(PlayerInfo.**class**).getLastName());  
 pInfo.setImage(dataSnapshot.child(**user\_id**).getValue(PlayerInfo.**class**).getImage());  
 pInfo.setTeamID(dataSnapshot.child(**user\_id**).getValue(PlayerInfo.**class**).getTeamID());  
  
 **c\_nav\_FirstName** = pInfo.getFirstName();  
 **c\_nav\_LastName** = pInfo.getLastName();  
 **disImage** = pInfo.getImage();  
 **c\_nav\_Email** = **mAuth**.getCurrentUser().getEmail();  
 **c\_nav\_Badge** = pInfo.getTeamID();  
  
 **coachID** = pInfo.getTeamID();  
 **final** Calendar calendar = Calendar.*getInstance*();  
 **day\_x** = calendar.get(Calendar.***DAY\_OF\_MONTH***);  
 **month\_x** = calendar.get(Calendar.***MONTH***);  
 **year\_x** = calendar.get(Calendar.***YEAR***);  
  
  
 updateCalendar(**coachID**);  
  
 updateMenu(**c\_nav\_FirstName**, **c\_nav\_LastName**, **c\_nav\_Email**, **disImage**, **c\_nav\_Badge**);  
  
 }  
 }  
  
 @Override  
 **public void** onCancelled(DatabaseError databaseError) {  
  
 }  
});

Figure 5.43 contains the code used to gather player information and begin updating their calendar. Due to the values within the Coach and Player node differing (a player node contains a field names ‘TeamID’, whereas a Coach node does not), a new Java class was required to gather Player data.This Java class is named ‘PlayerInfo’.

Once data is collected the team ID stored in a variable names ‘coachID’ and this is passed to the ‘updateCalendar( )’ method to be used to create accurate database references.

A player cannot interact with this calendar and dates can only be added by the coach.

**5.12 PLAYER – MY ACCOUNT**

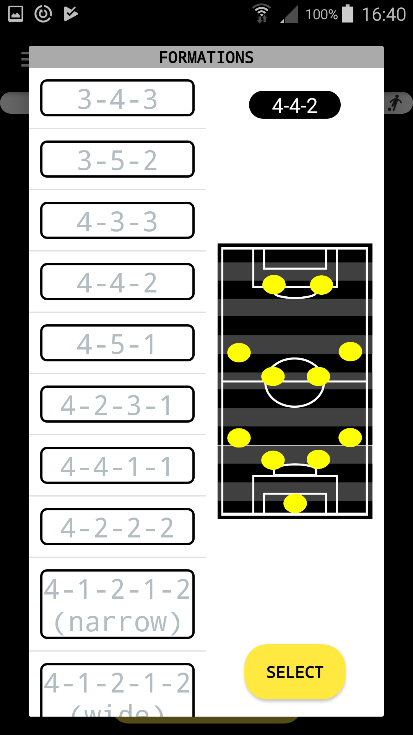
The player’s account set up is a simpler version of the coach side created in section 5.7. It has only one image the user can upload (profile picture) as only a coach can edit a team club badge.

The code is very similar to that contained throughout section 5.7 with only slight changes to the database references – once the team ID is gathered it is utilised within these database references.

The layout also follows a similar template to the coach’s version and facilitates the player’s changing their password. The same ‘updatePassword( )’, included in Figure 5.31, is used to change the password once the user is reauthenticated through the use of an Alert Dialog.

Figure 5.44 – Player – My Account Layout

**5.13 COACH – LOG A MATCH DAY**

This screen would be the one to contain the most functionality. Requirement F11 states “Coaches should be able to log statistics to Match Day entries”, and this is where coaches will log all match day data.

The first stage of implementing this screen was to create a section for adding specific match day information (i.e. Opponent, Formation, Home or Away). This was completed using Edit Texts apart from the Formation option.

For this an Alert Dialog is used to provide the user with several predefined options accompanied by a diagram of the selected formation. The diagrams for the formations were created using Microsoft PowerPoint and then exporting the slides as PNG files. These PNG files are then imported into the Android Studio project. The fully implemented Alert Dialog for this option can be seen in figure 5.45.

Figure 5.45 – Formation Selector

Once a formation is selected it is added to the match day information. The user must hit the save button for this to uploaded to the database.

The next piece of this facility to be implemented was the adding of ‘match day events’. This is the foundation of the application. This is where a coach will collect match data manually in order to be reviewed later.

When contemplating on the best method of entering this data it was decided that a player will be selected, followed by the action they completed. This seemed like the most logical and efficient method of entry.

To select a player a new RecyclerView with a Grid layout was created. This would display the players within the coach’s squad using their profile picture and name. To create the grid layout with 3 columns the code in Figure 5.46 was required to format the RecyclerView.

Figure 5.46 – RecyclerView Formatting Code

**playerGrid**.addItemDecoration(**new** GridSpacingItemDecoration(3, 50, **false**));  
**playerGrid**.setLayoutManager(**new** GridLayoutManager(**this**, 3));

To populate the RecyclerView with player data a similar technique to that used to implement the ‘Coach-View Player’ screen. A ‘FirebaseRecyclerAdapter’ is used to create a list of players with their associated information.

A ‘populateViewHolder’ method is used to assign the data to the items in the Recycler View (Figure 5.47).

Figure 5.47 – Populating Player Grid Code

**final** FirebaseRecyclerAdapter<players, LogMatchDay.playersViewHolder> firebaseRecyclerAdapter = **new** FirebaseRecyclerAdapter<players, LogMatchDay.playersViewHolder>(  
 players.**class**,  
 R.layout.***player\_event\_list\_item***,  
 LogMatchDay.playersViewHolder.**class**,  
 **mQueryPlayers**) {  
  
 @Override  
 **protected void** populateViewHolder(LogMatchDay.playersViewHolder viewHolder, players model, **int** position) {  
  
 **final** String player\_id = getRef(position).getKey();  
 **final** String player\_fname = model.getFirstName();  
 **final** String player\_sName = model.getLastName();  
  
 viewHolder.setfName(model.getFirstName());  
 viewHolder.setsName(model.getLastName());  
 viewHolder.setImage(getApplicationContext(), model.getImage());  
  
 viewHolder.**itemView**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 **//initialise alert dialog**  
 });

Within Figure 5.47, the ‘onClickListener’ is set for each item created in the player grid. Within the ‘onClick’ method the AlertDialog for the choices of ‘events’ is inflated. This means that when a user selects a player a list of events will appear.

The AlertDialog was created in a separate xml file, to code the action required for button click on this alert dialog a separate ‘buttonClicked’ method was created.

This method will run when any button within the AlertDialog is clicked, it will calculate which button was pressed by finding out its ID and will perform an appropriate action for that button.

This calculation is done through a series of ‘else if’ statement. A brief code snippet is provided in Figure 5.48.

Figure 5.48 – Event OnClick Code

**public void** buttonClicked(View v) {  
  
 DatabaseReference currentUserDB = **dbReferenceMatch**.child(**user\_id**).child(**dateKey**).child(**"Events"**);  
 DatabaseReference playerSpecEvent = **dbReferenceMatch**.child(**user\_id**).child(**dateKey**).child(**"PlayerEvents"**).child(**playerSelectedID**);  
 DatabaseReference teamStats = **dbReferenceMatch**.child(**user\_id**).child(**dateKey**).child(**"TeamStats"**);  
  
 *//gk* **if** (v.getId() == R.id.***btnGKshotP***)  
 {  
 Button button = (Button) v.findViewById(R.id.***btnGKshotP***);  
 String btnText = button.getText().toString();  
 currentUserDB.push().setValue(**playerSelectedName** + **" - "** + btnText);  
 playerSpecEvent.child(btnText).push().setValue(**"OK"**);  
 teamStats.child(btnText).push().setValue(**"OK"**);  
 **Eventdialog**.cancel();  
 }  
 **else if** (v.getId() == R.id.***btnGKgoal***)  
 {  
 Button button = (Button) v.findViewById(R.id.***btnGKgoal***);  
 String btnText = button.getText().toString();  
 currentUserDB.push().setValue(**playerSelectedName** + **" - "** + btnText);  
 playerSpecEvent.child(btnText).push().setValue(**"OK"**);  
 teamStats.child(btnText).push().setValue(**"OK"**);  
 **Eventdialog**.cancel();  
 }  
 **else if** (v.getId() == R.id.***btnGKshotC***)  
 {  
 Button button = (Button) v.findViewById(R.id.***btnGKshotC***);  
 String btnText = button.getText().toString();  
 currentUserDB.push().setValue(**playerSelectedName** + **" - "** + btnText);  
 playerSpecEvent.child(btnText).push().setValue(**"OK"**);  
 teamStats.child(btnText).push().setValue(**"OK"**);  
 **Eventdialog**.cancel();  
 }

**...**

Figure 5.48 also contains the code used to upload the event data to the database. Within every ‘else if’ statement the text contained within the button is gathered and stored in a String variable.

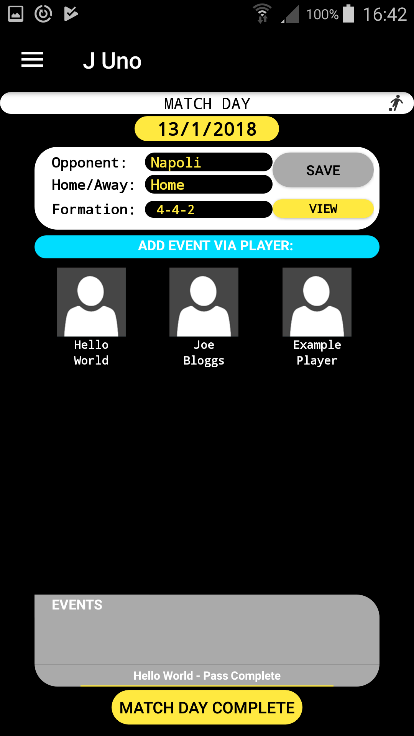
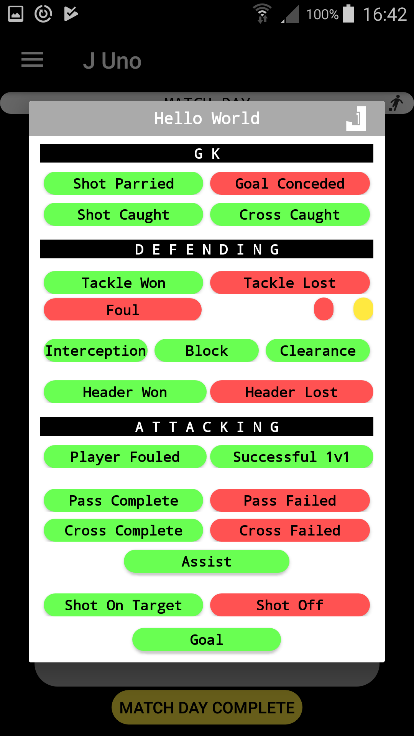
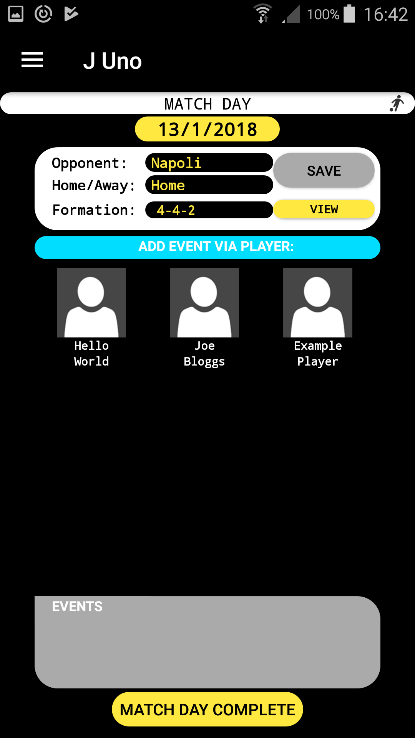
The ‘currentUserDB’ database reference is used to add a String to node in the database which will be used to provide real-time feedback to the user. A String example for this function is “Example Player – Pass Complete”. This is added to the ‘Matches/CoachID/MatchDate/Events’ node.

Next a value of “OK” is pushed onto the node used to store player specific events. The same value of “OK” is also pushed to collective team stats node for the event in question.

To simplify this, an event description is added to the database along with the addition of the event to both team and player nodes. The “OK” variable could be equal to anything, this is added to the node and the nodes children will be counted when collecting data at a later stage.

A ListView was also added to the layout of this activity to provide a list of all actions added to the match day. This was an important addition to ensure the user has immediate feedback adding data. This is done by retrieving all the data added to the ‘Matches/CoachID/MatchDate/Events’ node mentioned previously.

Figure 5.49 shows the fully implemented version of this facility, it shows a user selecting the “Hello World” player to add that they completed a pass.



**MatchDay Layout**

**AlertDialog Layout**

**Event Added**

Figure 5.49 – Process of Logging Data

A coach can leave this screen and all data is still stored, only when the ‘Match Day Complete’ button is clicked can no changes be made. Once this button is clicked the match date is transferred from ‘Incomplete’ to ‘Complete’ Match days.

This small function ensures no further changes can be made and only completed dates will appear as an option when view match data.

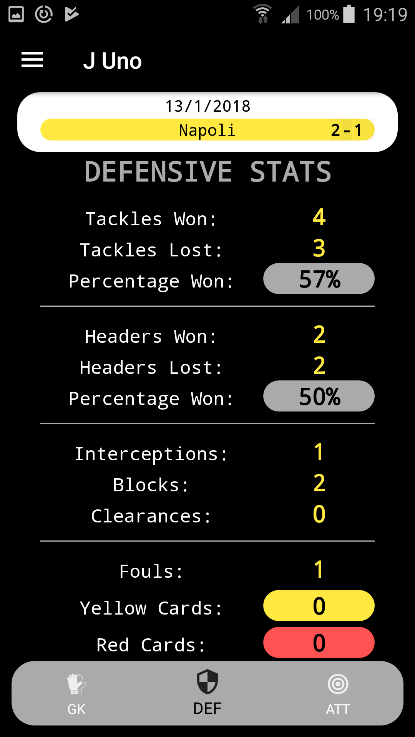
Once implemented this facility turned out better than expected. With the speed of data collection being high and the design being very intuitive, the process of logging match data is a highlight of this project.

**5.14 VIEW TEAM STATISTICS**

The first screen available to both players is viewing team statistics for completed match days. When implementing this screen, a ‘BottomNavigationBar’ was utilised to allow a user to switch between the three categories of statistics (Goal Keeper, Defensive, Attacking).

Fragments are used to implement the code within the changing xml files. Fragments were required as layouts of each category changes drastically. Fragments are held between the Match information and ‘BottomNavigationBar’ and are loaded when an option is selected from the Navigation bar.

Figure 5.50 – Team Statistic Fragments



To start each fragment when selected from the ‘BottomNavigationBar’ an ‘onClickListener’ is set with a ‘switch-case’ statement to determine which category was selected. Figure 5.46 contains a code snippet implementing this.

Figure 5.51 – Loading a Fragment Code

**btmNav**.setOnNavigationItemSelectedListener(**new** BottomNavigationView.OnNavigationItemSelectedListener() {  
 @Override  
 **public boolean** onNavigationItemSelected(@NonNull MenuItem item) {  
 **switch** (item.getItemId()) {  
 **case** R.id.***navigation\_GK***:  
 *//code for fragment* Bundle bundle = **new** Bundle();  
 bundle.putString(**"Matchdate"**, **dateKey**);  
 coachTeamStatsGK fragment = **new** coachTeamStatsGK();  
 fragment.setArguments(bundle);  
 FragmentTransaction fragmentTransaction1 = getSupportFragmentManager().beginTransaction();  
 fragmentTransaction1.replace(R.id.***teamStatContent***, fragment, **"Fragment GK"**);  
 fragmentTransaction1.commit();  
 **return true**;  
 **case** R.id.***navigation\_DEF***:  
 *//code for fragment* **...**

Each Fragment’s Java file contains the code used to collect each category’s statistics. This collection of data is done through the utilisation of the ‘getChildrenCount( )’ method. This will count the children added to each event, in this case, it will count the number of “OK”’s in each event node in the database.

It was also required to check if each event node was present before attempting to count its children. This is to avoid an error as an event node is only added when the event is selected whilst logging the match day data.

Figure 5.52 – Collecting an Event Total Code

**if** (dataSnapshot.hasChild(**"Pass Complete"**)) {  
 DatabaseReference dbRefStat = dbTeamStats.child(**"Pass Complete"**);  
 dbRefStat.addValueEventListener(**new** com.google.firebase.database.ValueEventListener() {  
 @Override  
 **public void** onDataChange(DataSnapshot dataSnapshot) {  
 **pCompAmt** = dataSnapshot.getChildrenCount();  
 String amount = String.*valueOf*(**pCompAmt**);  
 **double** int\_2 = Double.*valueOf*(amount);  
  
 **txt\_pComp** = (TextView) getView().findViewById(R.id.***pCompAmt***);  
 **txt\_pComp**.setText(amount);  
  
 *//work out percentage* **txt\_pFail** = (TextView) getView().findViewById(R.id.***pFailAmt***);  
 String stringpFail =**txt\_pFail**.getText().toString();  
 **double** int\_1 = Double.*valueOf*(stringpFail);  
  
 **double** total = (int\_1 + int\_2);  
 **double** percent = (100/total \* int\_2);  
 DecimalFormat decim = **new** DecimalFormat(**"0"**);  
 String p = decim.format(percent);  
 **txt\_pAcc** = (TextView) getView().findViewById(R.id.***pPerAmt***);  
 **txt\_pAcc**.setText(p + **"%"**);  
 }  
  
 @Override  
 **public void** onCancelled(DatabaseError databaseError) {  
  
 }  
 });  
}

Figure 5.52 shows how one event total is collected. A database reference is checked to see if it contains a child that contains “Pass Complete”. If it does not, not calculations are done as the default for the totals within the fragment layout is set to ‘0’.

If it does contain the specified String, a database reference pointing to that event is created and its children are counted. The method used for this returns a Long variable. This is converted into a String and, in some cases, a double variable for use in calculating averages.

The TextView for the corresponding event is set to the child count and any further calculations are performed if necessary. This snippet of code is repeated for each event in a fragment/category.

**5.15 VIEW PLAYER STATISTICS**

Requirements F13 and F24 state that both coach and player ‘should be able to review individual player performances of completed match day entries’. This activity was created to allow a user to navigate between different statistics and view each player’s totals for those events.

A new GridView was created using another formatted RecyclerView, although this time it was formatted to only have 2 columns. The code within Figure 5.46 was edited to implement this.

Each button edits the adapter used to populate the GridView with different totals. Not all information collected is available from this screen as it made it cluttered and unorganised. Only comparable and related events were included.

Figure 5.53 – Player Stat Layout

The code used for collecting the totals is similar to that included in Figure 5.53 but the database reference used is - “Matches/CoachID/MatchDate/PlayerEvents”.

**5.16 COMPARE TEAM PERFORMANCE**

Requirement F14 states that ‘Coaches should be able to review team performances of completed match day entries’. For this application, it was decided that a coach would be able to review 2 match day entries at a time. A different menu system for all completed match days was required.

The original menu used for simply viewing statistics for matchdays was edited to include a CheckBox for each date.

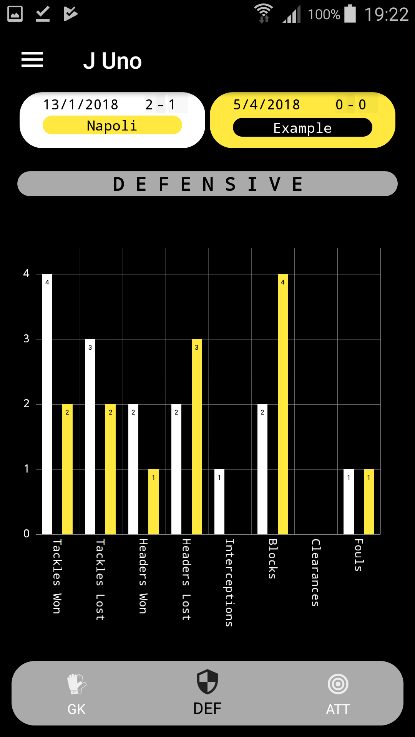
Figure 5.54 – Compare Menu

A user will select 2 dates to compare and select the ‘Compare’ button. If less than 2, or more than 2 dates are selected the user is given understandable feedback.

If only 2 dates are chosen the selected dates are passed to the next Activity using the ‘.putExtra( )’ method when initialising the new Intent, shown in Figure 5.55.

Figure 5.55 – putExtra Code Snippet

Intent compDates = **new** Intent(coach\_compare\_team\_stats.**this**, coach\_compare\_team\_stats\_content.**class**);  
compDates.putExtra(**"dateOne"**, dateOne);  
compDates.putExtra(**"dateTwo"**, dateTwo);  
compDates.addFlags(Intent.***FLAG\_ACTIVITY\_CLEAR\_TOP***);  
startActivity(startMatch);

The layout of the main screen for comparing team performances follows the format of viewing team statistics, in that it utilises a ‘BottomNavigationBar’. This is used to navigate between goal-keeper, defensive and attacking statistics for each match day.

Graphs are displayed to allow a user to easily compare differences in key areas. Fragments were used to handle the initialization of the graphs when an option is selected.

Figure 5.56 – Team Performance Layout

Graphs were created using an API named MPandroid. After some research this API allowed the customization and flexibility required for this application.

MPandroid allows a developer to create numerous graph types using arrays of floats (values) and Strings (labels). For comparing performances a grouped Bar chart was used.

When coding the collection of these data sets (float arrays) it became apparent that the method of collecting data from Firebase is not completed in a procedural fashion (line-by-line). Each snippet of code (Figure 5.57) needed to collect a figure is done simultaneously.

This meant that there needed to be a method of knowing when all data is collected. This is because, during testing, when data was not tracked the application would crash as the graph was attempting to deploy with missing data.

Figure 5.57 – Collecting Team Performance Data Code Snippet

dbTeamEventsOne.addValueEventListener(**new** ValueEventListener() {  
 @Override  
 **public void** onDataChange(DataSnapshot dataSnapshot) {  
 **if** (dataSnapshot.hasChild(**"Pass Complete"**)) {  
 DatabaseReference dbRefStat = dbTeamEventsOne.child(**"Pass Complete"**);  
 dbRefStat.addValueEventListener(**new** com.google.firebase.database.ValueEventListener() {  
 @Override  
 **public void** onDataChange(DataSnapshot dataSnapshot) {  
 **long** count = dataSnapshot.getChildrenCount();  
 **float** passesComp = Float.*valueOf*(count);  
  
 **entriesOne**[0] = passesComp;  
 }  
  
 @Override  
 **public void** onCancelled(DatabaseError databaseError) {  
  
 }  
 });  
 }  
 **else** {  
 **entriesOne**[0] = 0f;  
 }

When collecting the final data for each date a series of ‘if-else’ loops were used to ensure all data is collecting before a graph is drawn. Figure 5.58 displays a code snippet of this process:

Figure 5.58 – Data Collection If-Else Code Snippet

**if** (**entriesOne**[0].equals(**null**) || **entriesOne**[1].equals(**null**) || **entriesOne**[2].equals(**null**) || **entriesOne**[3].equals(**null**) || **entriesOne**[4].equals(**null**) || **entriesOne**[5].equals(**null**) || **entriesOne**[6].equals(**null**) || **entriesOne**[7].equals(**null**))  
{  
 *//do nothing*}  
**else**{  
 **entryOne** = **true**;  
 **if** (**entryOne** && **entryTwo** == **true**)  
 updateGraph(**entriesOne**, **entriesTwo**);  
}

To simplify this code – if any items in the array are equal to ‘null’ nothing is done. When all items are assigned a value a Boolean for the first date is set to true (this is the indicator that this array is complete).

This process is completed for the second date, and when the Booleans for each date are set to ‘true’ the method ‘updateGraph( )’ is called with both arrays included.

This ‘updateGraph( )’ method creates the graph. The code required for this is quite extensive. Totalling 78 lines of code, but includes:

* Graph customization (backgrounds, animations etc)
* Creating List Arrays from the data collected previously
* List Arrays are converted into Bar Data sets
* Labels for each section of the graph are defined
* Bar Data is assigned to the graph
* X-axis data is initialised and customised
* Y-axis data is initialised and customised
* A method ‘invalidate( )’ is called to refresh the graph

**5.17 COMPARE PLAYER PERFORMANCE**

To develop this activity a combination of techniques used in section 5.15 and 5.16 were used.

For this facility, a list of all players within a team is shown, when a user selects an option from the buttons on the left it displays graphs depicting success rates of key events in each game.

To create this, first the list of players is initialised. This is done through generating an ‘onStart’ method which initialises the default adapter for the ListView. This default adapter simply fills the list with all players from the user’s team. It does not collect any match data or create any graphs. The ‘onStart’ method can be seen in Figure 5.59.

Each button has a ‘onClickListener’ set which then handles the changing of colours to indicate which button is selected before replacing the current ‘FirebaseRycyclerAdapter’.

Figure 5.59 – onStart Default List Adapter

@Override  
**protected void** onStart() {  
 **super**.onStart();  
 **final** FirebaseRecyclerAdapter<players, coach\_compare\_ind\_stats\_content.playersViewHolder> firebaseRecyclerAdapter = **new** FirebaseRecyclerAdapter<players, playersViewHolder>(  
 players.**class**,  
 R.layout.***player\_stat\_row***,  
 coach\_compare\_ind\_stats\_content.playersViewHolder.**class**,  
 **mQueryPlayers** ) {  
 @Override  
 **protected void** populateViewHolder(playersViewHolder viewHolder, players model, **int** position) {  
 **final** String player\_id = getRef(position).getKey();  
 **final** String player\_name = model.getFirstName() + **" "** + model.getLastName();  
  
 viewHolder.setName(model.getFirstName(), model.getLastName());  
 viewHolder.setImage(getApplicationContext(), model.getImage());  
 }  
 };  
  
 **playerGrid**.setAdapter(firebaseRecyclerAdapter);  
}

The code to replace this adapter is similar to that in Figure 5.59, only with the addition of one line:

Figure 5.60 – setData Code Snippet

viewHolder.setData(getApplicationContext(),**"Tackle Won"**, **"Tackle Lost"**, **user\_id**, player\_id, **dateKeyOne**, **dateKeyTwo**);

This line is added at the end of the ‘populateViewHolder’ method and this is what begins the creation of each players graph.

This method takes several variables when called:

* Context – ‘*getApplicationContext( )*’ – context is added to help with any error that occur, Toast’s which deliver the error to the user require this to appear.
* Event Titles – ‘*Tackle Won’*, ‘*Tackle Lost’* – these are required to collect the data from firebase within the method
* Identifiers – ‘*user\_id’*, ‘*player\_id’* – these are required for the database references required for data capture within the method
* Date Keys – ‘*dateKeyOne*’, ‘*dateKeyTwo*’ – the dates, without “/”, are required to complete the database references during data collection

A code snippet of data collection can be seen in Figure 5.61.

A database reference is created for each date, using the date keys included in the calling of the method. For each event title it is checked that there is a ‘child’ with that name, in this case, this is done by the line:

Figure 5.61 – Data Collection for Player Graphs Code Snippet

@Override  
**public void** onDataChange(DataSnapshot dataSnapshot) {  
 **if** (dataSnapshot.hasChild(eventWon))  
 {  
 statRefOne.child(playerID).child(eventWon).addValueEventListener(**new** ValueEventListener() {  
 @Override  
 **public void** onDataChange(DataSnapshot dataSnapshot) {  
 **statWon** = dataSnapshot.getChildrenCount();  
 **statWonSet** = **true**;  
  
 **if** (**statWonSet** == **true** && **statLostSet** == **true**)  
 {  
 *//work out percentage* **if** (**statWon** == 0)  
 {  
 statAmount1.setText(**"0"**);  
 statAmount1.setVisibility(View.***VISIBLE***);  
 symbolOne.setVisibility(View.***VISIBLE***);  
 entriesOne[0] = 0f;  
  
 **if** (entriesOne[0] != 101f && entriesOne[1] != 101f) {  
 *createGraph*(mView, entriesOne, ctx);  
 }  
 }  
 **else** {  
 **long** total = (**statWon** + **statLost**);  
 **long** percent = ((100 / total) \* **statWon**);  
 **float** value = (**float**)percent;  
 entriesOne[0] = value;  
 DecimalFormat decim = **new** DecimalFormat(**"0"**);  
 String p = decim.format(percent);  
 statAmount1.setText(p);  
 statAmount1.setVisibility(View.***VISIBLE***);  
 symbolOne.setVisibility(View.***VISIBLE***);  
  
 **if** (entriesOne[0] != 101f && entriesOne[1] != 101f) {  
 *createGraph*(mView, entriesOne, ctx);  
 }  
 }  
 }  
 }  
  
 @Override  
 **public void** onCancelled(DatabaseError databaseError) {  
  
 }  
 });  
 }  
 **else{** { **...**

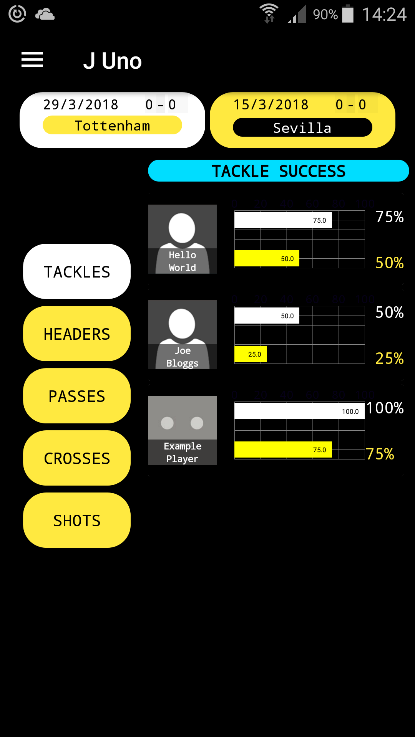
}

“if (datasnapshot.hasChild(eventWon))”

If there is a node with that title it then begins to collect the data. A separate database reference is created with the event title included. “Datasnapshot.getChildrenCount( )” is used to collect the total of the event. Once collected a Boolean is changed to indicate that this was successful.

This Boolean is then checked against the other Boolean for that date, which reference the second event title included in the method.

If both are set a percentage can be calculated and added to the Array of Floats required for creating the graph.

Each entry within the array is checked to ensure it has been set correctly. This is done by checking if it is less than 101.

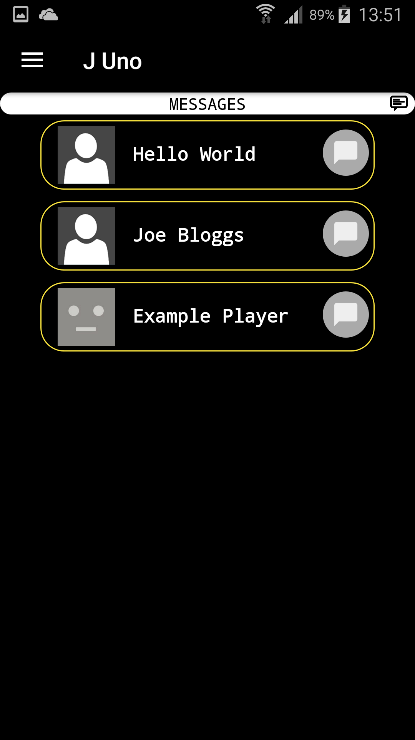
When initialising the array each element is set to 101 as default. Only when data collection is complete will this become less than 101 as all percentages will have a maximum of 100.

Only when this check is successful and all elements are less that 101 will the ‘createGraph( )’ method be called. This method contains all code required to create, customise and refresh the graph as mentioned in section 5.16.

A horizontal Bar Chart was used, this is another option available to the developer when using MPandroid. Figure 5.62 shows the resulting layout from this process.

Figure 5.62 – Player Performance Layout

**5.18 CHAT FACILTY**

As part of the requirements set out for this project it was required to develop a chat facility for both coaches and players. When speaking to stakeholders it was decided that coaches will be able to chat to all players in their team, but players will only be able to chat to the coach of their team.

For a coach, they will select a player from a list to access their messages.

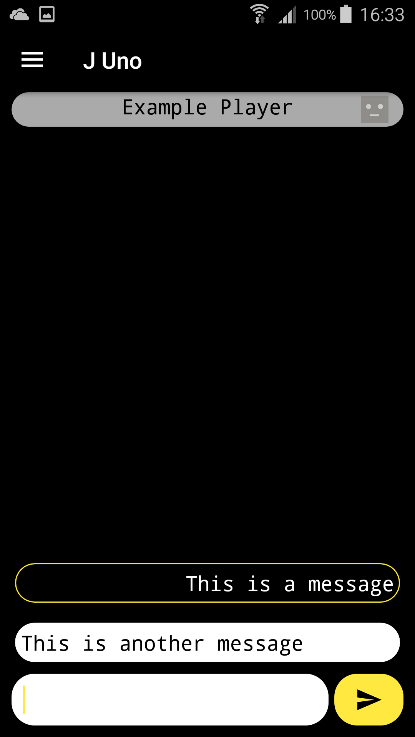
Figure 5.63 – Player Chat Select for Coach

This was achieved by simply creating a list of all the players within their squad. A new list item layout was created to include their profile picture, name and a message icon.

When a coach click the icon on a specific player a new Intent is started and certain data is forwarded using the ‘.putExtra( )’ method, this is highlighted in Figure 5.64.

viewHolder.**playerListChat**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 Intent mIntent = **new** Intent(Coach\_Chat\_Select.**this**, coach\_Chat.**class**);  
 mIntent.putExtra(**"PlayerID"**, player\_id);  
 mIntent.putExtra(**"PlayerName"**, player\_name);  
 mIntent.putExtra(**"PlayerImage"**, player\_image);  
 startActivity(mIntent);  
 }  
});

Figure 5.64 – Chat Intent Example

The player ID, name and image URI are all forwarded to the next Activity to aid in generating layouts.

The layout of the chat facility itself consists of a title Bar (which contains the name and profile picture of the recipient), a ListView (this contains all messages sent/received between the two users) and an Input Bar (this is used to enter and send messages).

The player name is simply transferred to create the title bar for this screen. An ‘if/else’ statement is used to generate their profile picture.

Figure 5.65 – Chat Facility Layout

If the player’s image has not been changed this image is set to the default profile picture. If it has been edited, Picasso (shown in Figure 5.26) is used to download the image from Firebase using the ‘player\_Image’ URI.

When a user types a message and selects the send button the text and name are gathered to store in Firebase. Once collected the values are used within a new Java class names ‘ChatMessage’ (shown in Figure 5.66. The resulting data is pushed onto a node in Firebase which follows the following structure:

‘Messages/Coach ID/Player ID/Pushed Node ID”

Figure 5.66 – Send Message Code

**sendMessage**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 String m = **message**.getText().toString();  
 String u = **c\_nav\_FirstName** + **" "** + **c\_nav\_LastName**;  
  
 **if** (!TextUtils.*isEmpty*(m)){  
 **dbRefMessages**.push().setValue(**new** ChatMessage(m, u));  
 **message**.setText(**""**);  
 **messageList**.invalidate();  
 }  
 }  
});

The ‘ChatMessage’ Java class is used to set the data stored in Firebase. The main difference with this, compared to similar classes used in this project, is that the date the message was sent is gathered without user input. A code snippet of this Java class is included in Figure 5.67.

Figure 5.67 – Message Date Gathering Code Snippet

**public** ChatMessage(String messageText, String messageUser)  
{  
 **this**.**messageText** = messageText;  
 **this**.**messageUser** = messageUser;  
  
 **messageTime** = **new** Date().getTime();  
}

Displaying the messages to a user is done through a ListView. This is updated with the use of a ‘FirebaseListAdapter’. When collecting the data, messages are differentiated using the message sender name.

This differentiation allows messages to be displayed in a conventional message style (one user’s messages on one side). Each message in the list is edited dependant on the sender, background and gravity of the item are utilised to create this layout. The code to implement this is included in Figure 5.68.

Figure 5.68 – Implementing Message Layouts

**if** (sender.equals(**c\_nav\_FirstName** + **" "** + **c\_nav\_LastName**))  
{  
 Drawable drawable = getResources().getDrawable(R.drawable.***btn\_fadedyel***);  
 messageText.setGravity(Gravity.***RIGHT***);  
 messageText.setTextColor(getResources().getColor(R.color.***colorPrimary***));  
 messageText.setBackground(drawable);  
}  
**else** {  
 Drawable drawable = getResources().getDrawable(R.drawable.***bg\_title***);  
 messageText.setGravity(Gravity.***LEFT***);  
 messageText.setTextColor(getResources().getColor(R.color.***colorPrimaryDark***));  
 messageText.setBackground(drawable);  
}  
  
messageTime.setVisibility(View.***INVISIBLE***);  
messageUser.setVisibility(View.***INVISIBLE***);  
  
v.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 **if** (messageTime.getVisibility() == View.***VISIBLE***)  
 {  
 messageTime.setVisibility(View.***INVISIBLE***);  
 messageUser.setVisibility(View.***INVISIBLE***);  
 }  
 **else** {  
 messageTime.setVisibility(View.***VISIBLE***);  
 messageUser.setVisibility(View.***VISIBLE***);  
 }  
 }  
});

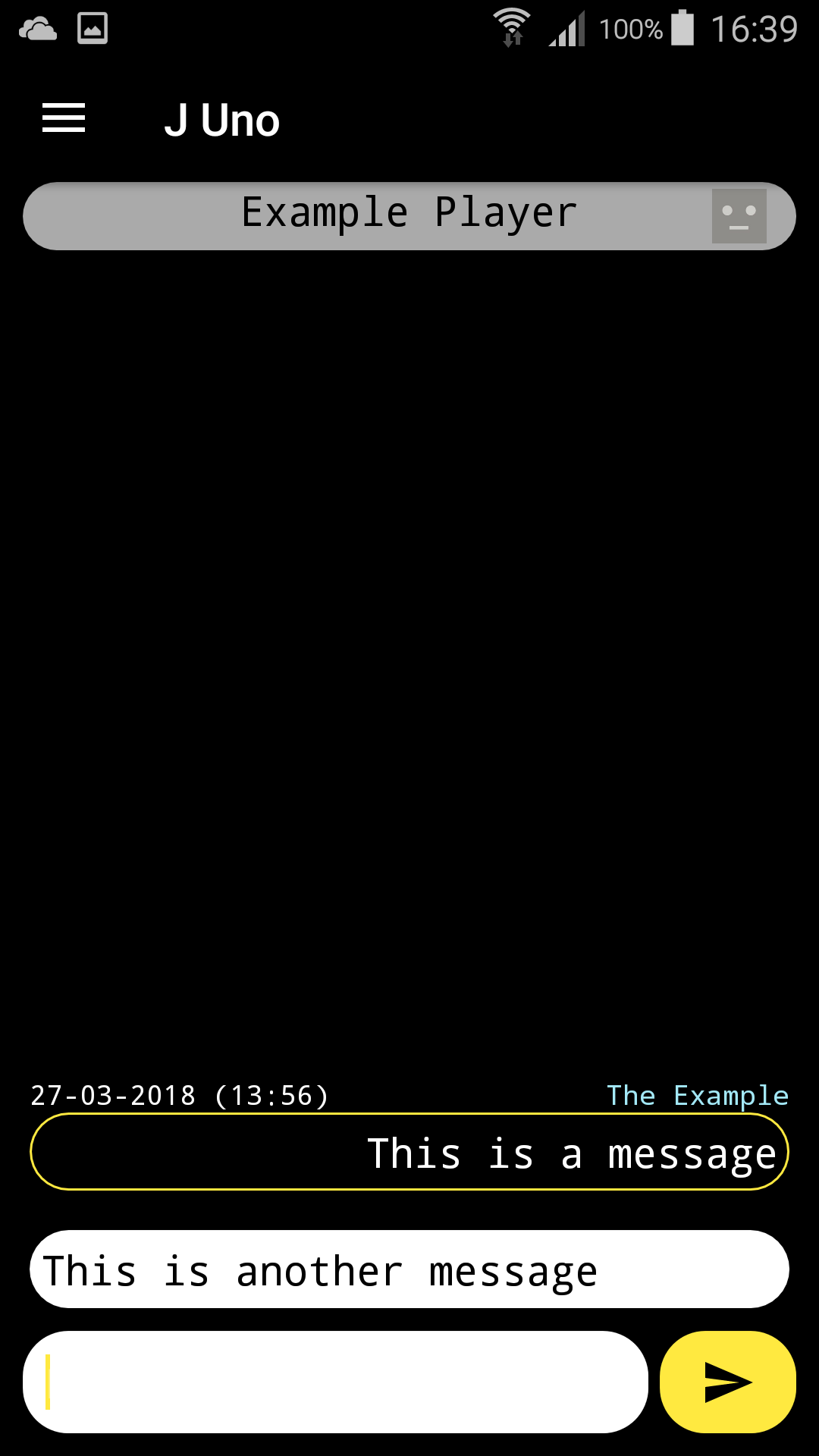
An ‘OnClickListener’ is also initialised for each item which reveals the sender and the time of the message selected.

Figure 5.69 – Message Sender and Time Example

**5.19 VIDEO UPLOAD**

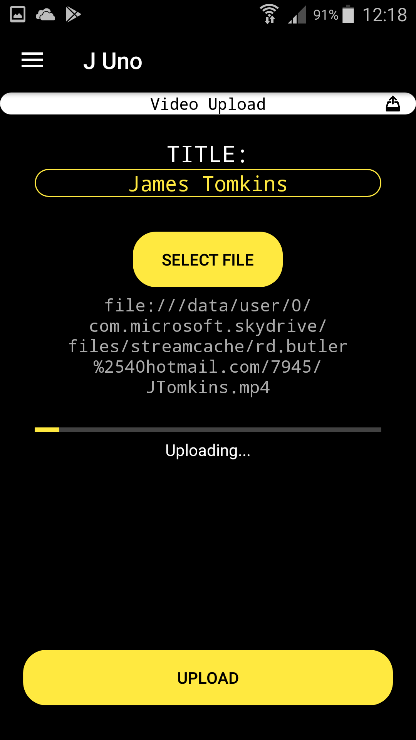
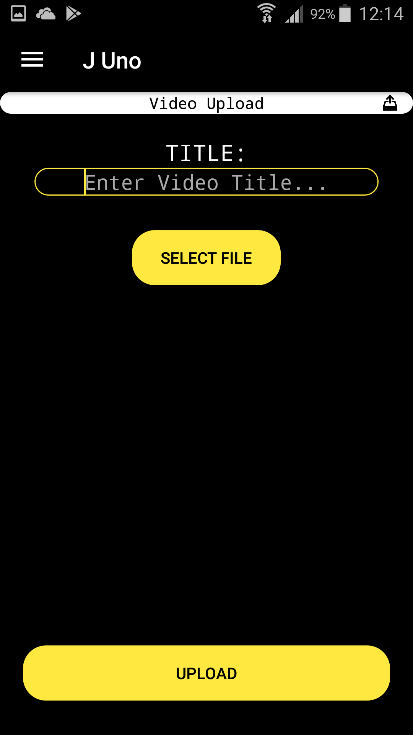


Figure 5.70 – Video Upload Layout

Requirement F17 states ‘Coaches should be able to upload videos to the system’, Figure 5.70 shows the layout developed to implement this function.

After implementing the upload of profile photos, this stage of implementation was completed in a shorter time than expected. This was due to the process being very similar, with some small changes, most to include UI elements to keep the user updated on the status of the upload.

The first stage of completing this activity was to add an ‘OnClickListener’ to the ‘Select File’ button which can be seen in Figure 5.70.

This ‘OnClickListener’ begins an Intent which facilitates the user selecting a video file from their device.

Figure 5.71 – Select Video File Code

**selectFile**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View view) {  
  
 Intent select = **new** Intent();  
 select.setType(**"video/\*"**);  
 select.setAction(Intent.***ACTION\_GET\_CONTENT***);  
 startActivityForResult(Intent.*createChooser*(select, **"Select Video:"**), **VIDEO**);  
 }  
});

Once a file is selected by the user it triggers a method names ‘onActivityResult’. This method gathers the URI for the file selected and displays this as the file path for the user.

When collecting this data an instance of a Media Player is created for the sole purpose of gathering the video duration before uploading. This code can be seen in Figure 5.72.

Figure 5.72 – onActivityResult Code Snippet

**if** (requestCode == **VIDEO**)  
 {  
 **videoURI** = data.getData();  
 **filePath**.setVisibility(View.***VISIBLE***);  
 **filePath**.setText(**videoURI**.toString());  
  
 **stringPath** = **videoURI**.getPath();  
  
 MediaPlayer mp = MediaPlayer.*create*(**this**, **videoURI**);  
 **int** duration = mp.getDuration();  
 mp.release();  
*/\*convert millis to appropriate time\*/* **videoDur** = String.*format*(**"%d min, %d sec"**,  
 TimeUnit.***MILLISECONDS***.toMinutes(duration),  
 TimeUnit.***MILLISECONDS***.toSeconds(duration) -  
 TimeUnit.***MINUTES***.toSeconds(TimeUnit.***MILLISECONDS***.toMinutes(duration))  
 );  
  
 **uriSet** = **true**;

Once all data is collected at this stage a Boolean named ‘uriSet’ is changed to ‘true’. This Boolean is used to check whether a file has been selected when the user clicks the ‘upload’ button.

When a user selects the upload button the title entered is gathered and converted into a String. A Firebase Storage Reference is initialised with the file path included.

An ‘OnSuccessListener’ is added to this reference to track when the upload has completed. Once it is completed a Database Reference is created to store the title, duration and download URI of the video.

Once this data is stored to Firebase the user is redirected to their Home screen with a Toast to confirm the upload was successful. This code can be seen in Figure 5.73.

To create the progress bar to allow the user to view a real-time update on the video upload process an ‘OnProgressListener’ was also added. This simply creates a Double variable which contains the percentage uploaded to Firebase. This Double is then assigned to the Progress bar.

An ‘OnFailureListener’ is also added to this process to identify and handle any errors with uploading the file. The ‘OnProgressListener’ and ‘OnFailureListener’ can be seen in Figure 5.74.

Figure 5.73 – OnSuccessListener for Video Upload

*//upload video*StorageReference firebasePath = **mStorageVideo**.child(**videoURI**.getLastPathSegment());  
firebasePath.putFile(**videoURI**).addOnSuccessListener(**new** OnSuccessListener<UploadTask.TaskSnapshot>() {  
 @Override  
 **public void** onSuccess(UploadTask.TaskSnapshot taskSnapshot) {  
 *//add to database when complete* String downloadUri = taskSnapshot.getDownloadUrl().toString();  
DatabaseReference dbReference = **dbReferenceVideos**.child(**user\_id**).child(videoKey6.toString());  
 dbReference.child(**"Title"**).setValue(title);  
 dbReference.child(**"DownloadURI"**).setValue(downloadUri);  
 dbReference.child(**"Duration"**).setValue(**videoDur**);  
 dbReference.child(**"CoachID"**).setValue(**user\_id**);  
 Toast.*makeText*(coach\_video\_upload.**this**, **"Successfully Uploaded "** + title, Toast.***LENGTH\_SHORT***).show();  
  
 *//navigate back to home screen* Intent c\_home = **new** Intent(coach\_video\_upload.**this**, coach\_Home.**class**);  
 c\_home.addFlags(Intent.***FLAG\_ACTIVITY\_CLEAR\_TOP***);  
 startActivity(c\_home);  
 }  
})

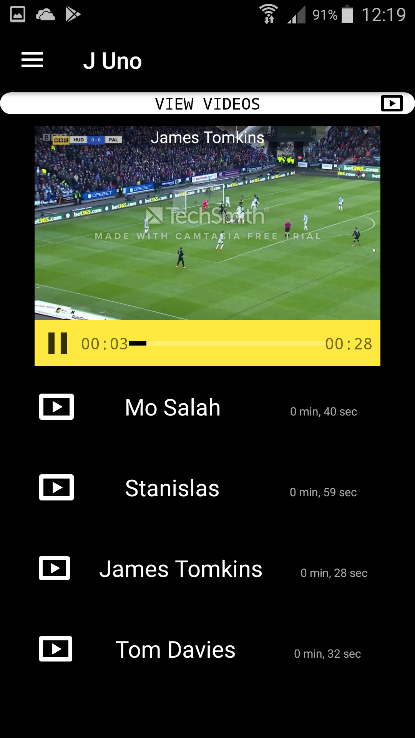
**5.20 VIDEO VIEW**  
  
 As stated in the Functional Requirements for this project, both user types should be able to view videos uploaded by the coach. To view videos the user selects a video file from the list. This video will play once selected.

Figure 5.74 – OnProgressListener and OnFailureListener Code

.addOnProgressListener(**new** OnProgressListener<UploadTask.TaskSnapshot>() {  
 @Override  
 **public void** onProgress(UploadTask.TaskSnapshot taskSnapshot) {  
 **double** progress = (100.0 \* taskSnapshot.getBytesTransferred() / taskSnapshot.getTotalByteCount());  
 **pBar**.setProgress((**int**)progress);  
 **uploading**.setVisibility(View.***VISIBLE***);  
 }  
});

.addOnFailureListener(**new** OnFailureListener() {  
 @Override  
 **public void** onFailure(@NonNull Exception e) {  
 Toast.*makeText*(coach\_video\_upload.**this**, **"There was an error while uploading video"**, Toast.***LENGTH\_SHORT***).show();  
 }  
})

A ‘VideoView’ element was utilised to contain the video. A basic video interface was created using various Buttons, TextViews and ProgressBars to create a conventional looking media player.

The list of video files is displayed below this and this is initialised with an ‘onStart’ method.

A new Java class named ‘VideoInformation’ was created to retrieve data on video files saved to the database. The ‘OnStart’ method uses ‘VideoInformarion’ to collect data from the database and fill the ‘ListView’. This code is included in Figure 5.76.

Figure 5.75 – Video View Layout

Figure 5.76 – Video List Creation Code

@Override  
**protected void** populateViewHolder(playersViewHolder viewHolder, VideoInformation model, **int** position) {  
 **final** String videoID = getRef(position).getKey();  
 **final** String videoTitle = model.getTitle();  
 **final** String videoURI = model.getDownloadURI();  
 **final** String videoDur = model.getDuration();  
  
 viewHolder.setTitle(model.getTitle());  
 viewHolder.setDuration(model.getDuration());  
  
 viewHolder.**itemView**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View view) {  
 **screenTitle**.setText(videoTitle);  
 **screenTitle**.setVisibility(View.***VISIBLE***);  
 Uri vidUri = Uri.*parse*(videoURI);  
 **mainVideo**.setVideoURI(vidUri);  
 **mainVideo**.requestFocus();  
 **mainVideo**.start();  
  
 **isPlaying** = **true**;  
 **playPause**.setImageDrawable(getResources().getDrawable(R.drawable.***ic\_pause***));  
  
 *//start ASYNC TASK* **new** videoProgress().execute();  
 }  
 });

Figure 5.76 shows the gathering of the video title and duration for the purpose of filling the ListView. It also includes the initialisation of the ‘OnClickListener’ for each item within the ListView.

Once an item is clicked the name of the video is transferred to the Title section of the VideoView. The URI is set and the video begins to play.

An Async Task is used to track and update the progress of the video to provide a valuable UI for the user. This Async task is began once an item is selected and can be seen being called within Figure 5.76.

The Async Task code can be seen in Figure 5.77:

**public class** videoProgress **extends** AsyncTask<Void, Integer, Void>{  
  
 @Override  
 **protected** Void doInBackground(Void... voids) {  
  
 **do** {  
  
 **if** (**isPlaying** == **true**) {  
 **current** = **mainVideo**.getCurrentPosition() / 1000;  
 publishProgress(**current**);  
 }  
  
 } **while**(**vidProgress**.getProgress() <= 100);  
  
 **return null**;  
 }  
  
 @Override  
 **protected void** onProgressUpdate(Integer... values) {  
 **super**.onProgressUpdate(values);  
  
 **try** {  
 **int** currentPercent = (100/**duration**) \* values[0];  
  
 **vidProgress**.setProgress(currentPercent);  
  
 String currentString = String.*format*(**"%02d:%02d"**, values[0]/60, values[0]%60);  
 **currentTime**.setText(currentString);  
  
 } **catch** (Exception e)  
 {  
  
 }  
 }  
}

Figure 5.77 – Async Task for Progress Update

This Async Tsk has 2 inner methods (doInBackground and onProgressUpdate). DoInBackground simply retrieves the current video’s position and publishes the progress.

This ‘publishProgress( )’ method triggers the onProgressUpdate method which updates the current time String and the Progress Bar.

There is also an ‘OnClickListener’ set to the ‘Play’ button which simply stops and starts the video and appropriately changes the icon to either a play icon or pause icon.

Testing is the last phases within the Software Development Life Cycle (SDLC) before the project is released to customers. For this project, testing would be completed before delivering the mobile application to the stakeholders for review.

CHAPTER 6 - TESTING

The aim of Testing is to identify defects within the system and to investigate whether the application behave as expected according to requirements documented in the Analysis stage.

Verification and Validation are important processes within Software Testing. These processes are used to check whether a system meets the specification and that it fulfils its intended purpose.

*VERIFICATION*

Verification is the process of evaluating products of a development phase to determine whether they meet the specified requirements for that phase, it’s main objective is to ensure that a product is being built according to the requirements and design specifications.

Verification took place regularly throughout the implementation of this project. The Gantt chart was reviewed frequently to ensure timescales were being met and after every element was created, the developer would consult the project requirements and designs to ensure they were being followed and met sufficiently.

*VALIDATION*

Validation is the process of evaluating software during and at the end of development to determine whether it satisfies specified business requirements. It’s main objective is to ensure that the product meets the needs of the intended user’s and that initial specifications were correct.

Validation was incorporated throughout the implementation stage of this project, through the form of unit testing. Systems testing was completed before releasing the project to the stakeholders.

At this stage many methods of testing were used to validate the system. Including navigation testing and device testing.

**6.1 UNIT TESTING**

Unit Testing is a method of software testing which individual units or components of a system are tested to validate that each performs as designed and specified.

For this project Unit Testing was performed throughout the Implementation stage. Each functional area of code was tested as it was being developed to ensure it performed as expected.

With this approach taken to Unit Testing, all errors discovered while developing the application were resolved immediately before moving on to the next feature of the system. This approach resulted in an efficient, consistent and reliable implementation of the application.

It assured the developer that each element of the application was functional once all features were complete when following the implementation plan. It also deterred minor issues developing into more serious problems, this was due to issues being easier to identify and resolve earlier in implementation.

**6.2 SYSTEM TESTING**

System’s testing is used to check the behaviour of a complete and fully integrated software product against the software requirements specification. Its main purpose is to evaluate Business, Functional and Non-Functional Requirements.

This type of testing is called Black Box type testing. It is when the external working of the software is evaluated with help from the requirements documents, results are based solely on the user’s point of view (i.e. structure of code, internal design etc. are not evaluated).

To document the process of System Testing the test’s themselves were broken into individual test cases which can be examined and given an outcome, this would be whether the test passed or failed.

Table 6.1 shows an example of a test case used to document this process:

Table 6.1 – Test Case Example

|  |  |  |  |
| --- | --- | --- | --- |
| **TEST ID** | **DESCRIPTION** | | |
| 1 | New coaches should be able to register a new account with the system | | |
| INPUT | First Name: The  Last Name: Example  Team Name: Team EG  Email: example@email.com  Password: Example01  Confirm Password: Example01 | | |
| EXP RESULT | - User will be able to navigate to the Register Screen successfully.  - User will be able to complete form with no issues  - Once input data is complete, it will be accepted and the user’s account will be created and they will be navigated to their Home screen | ACTUAL RESULT | As Expected. |
| PASS/FAIL | | | Pass |

Full System Testing results can be found in Appendix E. However, it is important to note here that although a test case may state it passed, there may be underlying issues that a user will encounter.

**6.2.1 ISSUES ENCOUNTERED**

**6.2.1.1 DELETED USER LOGIN**

When implementing the application, deleting a player was achieved by using a ‘removeValue( )’ method. This method does remove the player from the list, and from the Firebase database.

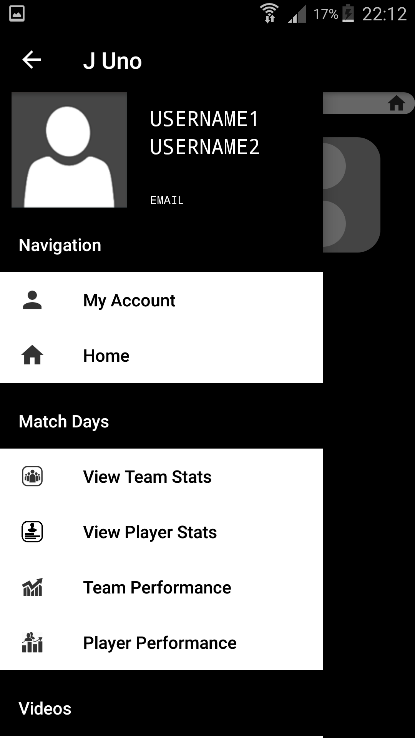
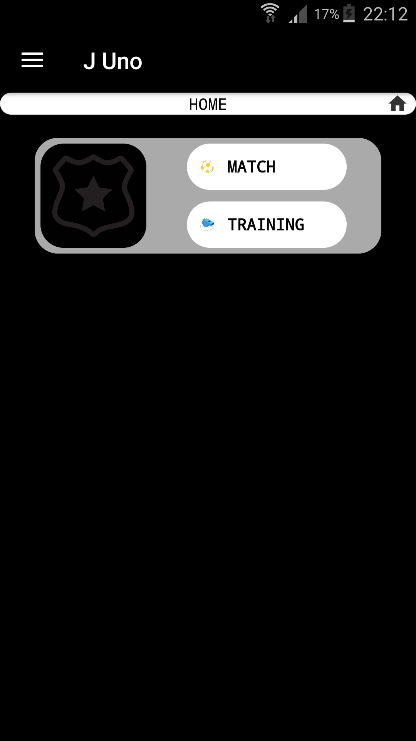
However, this does not delete the user from the Authentication server of Firebase. This means they are still authorised to log in to the system. This is due to a previously mention problem during development. The developer was unable to utilise the Firebase Admin SDK which would allow an admin user to delete another user from the system entirely.

Without this, the user’s data can only be removed from the database.

Upon testing this issue, it was revealed that even though a deleted user can log into the application, no data can be viewed as they are removed from the database. This database information is used to retrieve all other data for the application. This error causes the application to crash in some areas for deleted users.

Figure 6.1 displays what a deleted user can view when logged in.

Figure 6.1 – Deleted User Access



This is not an ideal situation, but sensitive data and information is still secure and unauthorised users cannot access it.

**6.2.1.2 VIDEO PROGRESS ERROR**

When testing the application during implementation it was noticed that when viewing videos on the application the progress bar used to track the current time of the video is incorrect.

All equations are correct and the developer is unaware as to why the error occurs. When a video is complete the progress figure is only ‘80’, where it should be ‘100’.

This error was not remedied as the causes was not identified. This was brought forward to all stakeholders and a solution is still being worked on.

**6.2.1.3 CHAT NOTIFICATIONS**

Although a chat facility was developed within the timescales of the project, the developer was unable to implement notifications for messages from this application.

The stakeholders have been made aware of this and are willing to accept the application, with the promise of an updated version with this issue resolved soon.

**6.2.1.4 SAMSUNG GALAXY S5 ERROR**

When testing the application on multiple devices an error was discovered to only affect one device, the Samsung Galaxy S5. This error occurs when selecting a formation for a match day.

From viewing the error on Android Studio, it seems to be an error in allocating memory for the images of formations. When is occurs, the application resets and on a second attempt runs without failing. This error has not occurred on any other device during testing.

The developer was unable to resolve this error but has made the stakeholders aware so they are able to advise potential users of the application that this error can occur on this device.

**6.3 USER ACCEPTANCE TESTING**

User Acceptance testing is the last phase of testing that was performed on this project. During User Acceptance Testing, actual users of the application test the software to assess whether it can handle required tasks in real-world scenarios.

User Acceptance Testing is a critical procedure in software testing which must occur before the new application is rolled out to the customer.

For this project, Jordan was able to test the application when at home for a month to aid the training of his previous club. Jordan was willing to take this opportunity to test the application on collecting real-time data on the players within this squad.

The developer gave Jordan a detailed walkthrough of the features and elements of the application. Once Jordan felt comfortable with the application, he used it for his temporary squad for one week, unsupervised. Jordan and the developer met after this week and discussed how the trial period went.

Jordan’s experience with the application was overall positive. He stated that using the application was slow at first, like it is with many applications, but once he learnt the layout it was easily navigated and collecting data was significantly easier than his current method.

Jordan was able to test logging match data within this week and was very happy with the speed data could be collected. Jordan was happy with the overall appearance and functionality of the application

During this testing period Jordan encountered no new errors that were not already identified by the developer.

**6.4 CONCLUSION**

Testing of the application has resulted in a robust, verified and validated application ready for releasing to its intended users.

Without this phase, errors would not have been identified which would inevitably lead to poor user experiences. The developer was also able to provide evidence that the application works as expected against requirements agreed with the stakeholders.

The application is now ready to be delivered to its intended users and the developer will continue with system maintenance and attempt to resolve any outstanding errors identified during Testing.

Documented in this chapter is the evaluation and results of the implemented mobile application. The planning of the project will be assessed along with end-user feedback to provide an insight into the success of the final product.

CHAPTER 7 – EVALUATION AND RESULTS

**7.1 REFLECTION ON TECHNOLOGIES USED**

**7.1.1 ANDROID STUDIO**

The developer had experience using Android Studio prior to beginning this project. This experience became vital in the implementation of the application as it was used to develop all elements of the system.

Android Studio was free to download and has a huge community of developers which was valuable to the developer when tackling new problems within the project. This community was called upon when errors occurred that the developer could not resolve alone.

Android Studio ran smoothly for the duration of the implementation stage and using their previous knowledge, the developer could implement the application in an efficient and timely manner. This saw the application being complete within timescales set in the Project Plan.

**7.1.2 FIREBASE**

Firebase provided all back-end functionality needed for this application. It facilitated data storage and included its own authentication methods to keep data secure.

The developer found the interface used in the Firebase console easy to understand and learn.

Firebase was used to store all data needed for the application, this includes user information, match information along with photos and videos.

Firebase is free to use, but the developer did run into one issue when implementing video facilities within the application. This issue was due to download limits for the free version of Firebase.

The download limit is set to 1GB per day. This caused issues when testing the video playback, as videos could only be played a certain amount of times before the limit was hit and access to Firebase Storage (that holds pictures and videos) is blocked for a 24-hour period. When this happens, the application will still function, but profile images and videos cannot be viewed, meaning development was delayed.

Firebase performed well even with this error and simplified the addition of an online database to mobile application, which was a new concept for the developer to overcome.

**7.2 REQUIREMENTS EVALUATION**

For the implemented system to be deemed a success, it would have to meet the functional, non-functional and business requirements defined within Chapter 3. There are a number of characteristics that make up a good requirement, and these are used to evaluate each requirement:

* Unambiguous
* Verifiable
* Clear
* Correct
* Understandable
* Feasible
* Necessary
* Consistent
* Complete

**7.2.1 FUNCTIONAL REQUIREMENTS**

The functional requirements for this project were made up of a combination of both coach and player requirements. All requirements mirrored tasks both users will need for the application to be complete.

All functional requirements were met through the implementation of the system, this is evidenced through all test cases resulting in a ‘Pass’.

Coaches can register, log in and log out of the application. Once logged in, they have access to editing their account, viewing and adding items to their team calendar, adding players to their squad, logging match days, viewing completed match day statistics, a chat facility alongside being able to upload and view videos.

Players can log in (once given their credential by their coach) and log out of the system. Once logged in players have access to editing their account, viewing their team’s calendar, viewing completed match day statistics, a chat facility and access to viewing videos uploaded by their coach.

These requirements are all understandable, feasible and required to create a system with all the functionality required by the end user. Table 7.1 displays the evaluation of each functional requirement against the characteristics listed above:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **No.** | **Unambiguous** | **Verifiable** | **Clear** | **Correct** | **Under-standable** | **Feasible** | **Necessary** | **Consistent** | **Complete** |
| F01 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| F02 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| F03 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| F04 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| F05 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| F06 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| F07 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| F08 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| F09 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| F10 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| F11 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| F12 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| F13 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| F14 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| F15 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| F16 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| F17 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| F18 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| F19 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| F20 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| F21 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| F22 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| F23 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| F24 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| F25 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| F26 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| F27 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| F28 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| F29 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

**7.2.2 NON-FUNCTIONAL REQUIREMENTS**

Table 7.1 – Functional Requirements Evaluation

The definition for a non-functional requirement is a requirement that specifies how a system should behave. In this project, non-functional requirements covered the compatibility and efficiency of a system.

All but one non-functional requirement was met through implementing the application. This one being that the application does not react to orientation changes.

As mentioned previously, it was decided by the developer that it would be better for the overall user experience if orientation was fixed when using the application. This decision was made due to information that needed to be displayed to the user, changing the orientation would cause the layout to be cluttered and unappealing.

Aside from this one non-functional requirement not being met, the application does have a consistent theme throughout that follows best practice and design principles. During testing, it was proved that it displayed correctly on differing screen sizes and no data can be accessed without being logged in.

The application is easily navigated, runs in the background and does not crash when being used. Passwords are also saved securely through Firebase’s Authentication facility.

With all other non-functional requirements successfully completed, Table 7.2 displays each requirement’s evaluation results. From these results, it was found that non-functional requirement NF09 - The application should conform to best practice and design principles for chosen devices – could be regarded as ambiguous.

To make this requirement less ambiguous, the developer could have specified conforming to Android’s ‘Material Design’ concepts. Material Design is a comprehensive guide, provided by android, for visual, motion and interaction design across platforms and devices.

Table 7.2 – Non-Functional Requirements Evaluation

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **No.** | **Unambiguous** | **Verifiable** | **Clear** | **Correct** | **Under-standable** | **Feasible** | **Necessary** | **Consistent** | **Complete** |
| NF01 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| NF02 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| NF03 | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ | ✗ | ✓ | ✓ |
| NF04 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| NF05 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| NF06 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| NF07 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| NF08 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| NF09 | ✗ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

**7.2.3 BUSINESS REQUIREMENTS**

There was one business requirement, which stated ‘The finished system should be completed by 22nd April 2018’. This requirement was met in full and was aided using the Gantt chart and work breakdown structure to manage the project through all phases of development.

**7.3 END USER EVALUATION**

The completed application was released to the end-users upon completion of Testing. After a period of time, both types of users (coaches and players) were asked their opinions on the functionality of the system and their own personal user-experience since the application was introduced.

Both Jordan Stephens (Coach) and his squad of 20 players had access to the application and completed evaluation questionnaires. A blank example questionnaire can be found in Appendix F.

The results from the questionnaires are analysed below.

**7.3.1 COACHES (JORDAN STEPHENS)**

Jordan Stephens was the only Coach targeted to use the application upon its release. He completed the evaluation questionnaires and his responses are detailed below:

*Question 1: I found the application easy to use and navigate.*

Strongly Agree.

*Question 2: I found match information clear and easily understood.*

Strongly Agree

*Question 3: Is the application an improvement from the current system?*

Strongly Agree

*Question 4: How is this an improvement?*

Able to access information 24/7,

Makes it easier to compare personal performances

Provides a more tangible method of providing feedback

Other – Increases efficiency of collecting statistics

Other – Makes it easier to compare team performances

*Question 5: How frequently would you use the new application?*

Daily

*Question 6: Did you experience any issues/errors whilst using the application?*

Yes – Progress bar on video did not work on occasion

*Question 7: I would recommend this application to other teams.*

Yes

*Question 8: How could this application be improved?*

Team dashboard – to allow updates to be posted by a coach,

Notifications would improve the chat facility,

Search Bar – used to search for players, used to get specific statistics on a player

Following receiving this feedback, the developer met with Jordan Stephens to discuss his experiences further. Overall, Jordan was very happy with the application that was developed in the timescales provided, he expressed his surprise at how well each of the requirements were met and was not discouraged by the error he encountered.

Jordan was able to suggest some interesting suggestions for improvement and plans to use the application going forward in his career as a youth coach. He also stated he was very keen to suggest the application to his peers.

**7.3.2 PLAYERS**

When released to Jordan’s squad, there were a potential user count of 20 players for the application. Each one was asked to complete an evaluation survey and the results are discussed below.

*Question 1: I found the application easy to use and navigate.*

Figure 7.1 – Evaluation Results – Question 1

From the 20 players in the team, 4 of them did not own an android device, and were therefore unable to use the application. The decision to create a native app, instead of a cross-platform application, did open this possibility to potential users. The developer is happy with this decision, however, as implementation of the application would have taken considerably longer if they were to try to use unknown tools and languages instead of drawing on previous experience in Android Studio.

Although undesirable, this still gave 16/20 players the opportunity to use the application. Of these 16, 13 stated that they would ‘strongly agree’ that they found the application easy to use and navigate. While 3 players selected ‘agree’ for the same statement.

This is a fantastic result as it proves that non-functional requirement NF04, ‘the application should be easily navigated’, has been met successfully.

*Question 2: I found match information clear and easily understood.*

From the 16 players that used the application, 15 of them strongly agreed that match information was clear and easily understood. 1 player stated they agreed with the same statement. This is a positive result from a design point of view, it means that all designs were meaningful and easy to use.  
  
This result was especially important to the developer as displaying so much information in an understandable format on a small device was a major challenge during design and implementation.

Figure 7.2 – Evaluation Results – Question 2

*Question 3: Is the application an improvement from the current system?*

Figure 7.3 – Evaluation Results – Question 3

A great result for the project was that all 16 players that could use the application strongly agreed that it was an improvement from their current system. This shows that all users believe this is a better method of conveying player and team performances than their current, manual method.

*Question 4: How is this an improvement?*

Figure 7.4 – Evaluation Results – Question 4

Question 4 asked the players their opinion on how this improved the current system. It was a multiple-choice question and the totals for each answer are shown in Figure 7.4.

The top answer amongst player was that it improved their knowledge on personal performance, it came out with a total of 14.

In second is making it easier to compare personal performances, with a total of 12 votes.

In third is increasing competition between team mates, with 10 votes.

These top three answers are a great result for the application. It shows that the application has helped the squad understand their performances whilst also having the added benefit of increasing competition within the team.

*Question 5: How frequently would you use the new application?*

From this question, it is clear to see that most players that used the application would use it after every game to check their performance. This is an acceptable result, as this is the application’s main purpose.

Figure 7.5 – Evaluation Results – Question 5

With further enhancements, addition of notifications and added functionality the developer hopes the application will grow to be used daily my members of sports teams.

*Question 6: Did you experience any issues/errors whilst using the application?*

4 players expressed that they had experienced errors whilst using the application. All specified that this was a video playback error, more specifically, an error with the progress bar when viewing videos.

The developer is happy with this result as no unidentified errors had effected users, proving that testing was thorough and complete.

All users have been advised that the application is being modified and an update is expected soon.

*Question 7: I would recommend this application to other teams*

All players that used the application answered that they would recommend the application to other teams.

Figure 7.6 – Evaluation Results – Question 7

This result is fantastic in terms of this project. The developer believes this result s due to the innovative nature of the application. This result give the developer confidence that continuing work on the project is worthwhile.

*Question 8: How could this application be improved?*

When asked how the application would be improved the answers given were grouped into similar groups and totalled to give the developer an idea of what actual users want from the application.

1. Notifications (10 votes)

Understandably, the most votes went towards notifications being added for the chat facility. The developer is aware of impact of the lack of notifications and is currently working on an update to add them to the application.

1. Leader Boards (8 votes)

In keeping with the results from Question 4, players enjoyed the competitive aspect of tracking personal performances. This lead to 8 votes being made for the addition of leader boards for key statistics such as Top Goal Scorers, Top Assists etc.

1. Group Chat (4 votes)

Four players suggested a group chat facility for messages to further enhance the functionality of the messaging service.

1. More Personalised Information (2 votes)

Two players suggested a more personalised approach to match data. All player data is available but in a group fashion (i.e. players can view their own statistics, but only within a list of all players). Creating a personalised screen for data would make it even easier for players to compare performances.

**7.4 REFLECTION ON PROJECT AIMS AND OBJECTIVES**

The aim of this project was ‘to create an application which allows the collection and sharing of football match statistics on each player in the team in the most efficient and practical manner possible’. This aim summarises the project and to meet this aim, the following main objectives were set:

1. **Research and evaluate current match statistic collection software available against the needs and requirements of the client** - This was completed as part of the Literature Review which identify similar applications with advantage and disadvantages of each documented.
2. **Create an online database to hold coach/player/match day information** – Firebase was utilised to create this online database. This was completed during the Implementation stage with the structure of each section within the database pre-defined during the Design phase.
3. **Create a mobile application to accommodate all users** – The design and implementation has been documented in Chapter 4 and Chapter 5 respectively. Android Studio was used to create the application, with Firebase providing the online database to complete the functioning system.

In order to achieve these main objectives, the following activities were planned to be performed:

1. **Review similar systems currently available to the client** – Completed as part of the Literature Review in Chapter 2. Current systems were evaluated and compared to gain insight into the current market
2. **Review mobile application development software** - Completed as part of the Literature Review in Chapter 2. Native and cross-platform development environments were reviewed to evaluate which is most appropriate for the project.
3. **Review online database tools for creating the database needed for the application** - Completed as part of the Literature Review in Chapter 2. All suitable options were evaluated along with a conclusion on which would be used in this project.
4. **Define system requirements through consultation with the client** – requirements gathering was completed using various method, all of which are documented in chapter 3. During Analysis, all requirements for the proposed application were defined.
5. **Review and evaluate various software development lifecycles appropriate to this project** - Completed as part of the Literature Review in Chapter 2. Software development Lifecycles were compared with advantages and disadvantages for each documented.
6. **Create initial designs for final product, with both coach and player functionalities considered** – all designs are documented within Chapter 4. This chapter details database structure, UI designs and application use case examples.
7. **Design Database format and relationships** – database designs and structures are contained within Chapter 4.
8. **Complete testing throughout the development stages to ensure a fully functional application is created** – Testing was completed using multiple methods to ensure the application was robust and ready to roll-out to the end users.
9. D**ocument all testing completed on the project** – The testing procedures used during this project are documented within Chapter 6.
10. **Create fully functional application to accommodate all users** – The implementation of the application has been documented in Chapter 5. This includes all aspects of the project to create the fully functioning application.
11. **Develop code to link online database to mobile application** – The beginning of implementation included creating the link between online database and mobile application.
12. **Create a project report** – this report has documented the project implementation and planning from start to finish.

This chapter ends the reports and includes a reflection on the success of the project along with developer plans for future enhancements for the application.

CHAPTER 8 – CONCLUSION

**8.1 POTENTIAL FUTURE ENHANCEMENTS**

Although the final application meets all the requirements initially set out in Chapter 3, the developer intends to continue working on the application and future enhancements to the application are listed below:

1. **Team Dashboard**

Allowing a facility for the coach to post updates to a notice board type interface. This would allow the coach to keep players updated on changes to training, match times, squad selection etc.

1. **Personalised Feedback for Players**

A more singular approach to feedback for matches would further improve players understanding of their performances. This interface would include highlighting differences in performances from their previous game.

1. **Squad Selection for Upcoming Matches**

To enhance a coach’s ability of selecting formations, it would be beneficial for them to be able to select a starting 11 for the selected match. This starting 11 could then be posted to the team dashboard, and each squad member would receive a notification.

1. **Leader Boards**

The addition of leader boards would further enhance in team competition and increase a coach’s knowledge of top performers within their team. This enhancement may also include the ability to select a ‘man of the match’ for each completed match day.

1. **Data Analytics**

The developer is keen to add analytics into the application. This would be a system of analysing current player data to enable to application to make suggestions to the coach when making their squad selections. This addition may also allow the application to highlight areas specific players need to work on.

1. **Cross-Platform Development**

The developer is keen to attempt to develop a similar style application which can be used on all mobile devices. This would eradicate the risk of some players not being able to utilise the system.

**8.2 PROJECT REFLECTION AND CONCLUSION**

The developer is very happy with the end result from this project. Throughout the development lifecycle there were uplifting achievements but also difficult challenges. However, despite the set-backs, the developer has received tremendous feedback from the end user which has inspired them to continue to improve the application.

When starting the project, the developer had very moderate experience in developing mobile applications, and had never attempted to implement an online database enabled system.

Even with this minimal experience, this large project has been very fulfilling and allowed the developer to gain significant knowledge in Android development alongside improving their skills in Java.

Due to the size of the project, the use of the Gantt chart and Work Breakdown Structure proved vital in keeping the developer on schedule and resulted in all elements being completed before strict deadlines.

The final application that was developed is a significant improvement to the stakeholder’s current system. The developer has been praised for the quality and complexity of the completed system. The coach, Jordan Stephens, expressed his delight in how easy the application is to use and record match data.

This feedback proves that the project was a success and has encouraged the developer to embark in similar projects.

References

Alshamrani, A. (2018). *A Comparison Between Three SDLC Models Waterfall Model, Spiral Model, and Incremental/Iterative Model*. [online] Academia.edu. Available at: http://www.academia.edu/10793943/A\_Comparison\_Between\_Three\_SDLC\_Models\_Waterfall\_Model\_Spiral\_Model\_and\_Incremental\_Iterative\_Model [Accessed 2 Feb. 2018].

Amazon Web Services, Inc. (2018). *AWS Mobile*. [online] Available at: https://aws.amazon.com/mobile/ [Accessed 2 Feb. 2018].

Andrews, S. (2018). *The evolution of the iPhone: Apple's journey to the iPhone 5*. [online] IT Pro Portal. Available at: https://www.itproportal.com/2012/09/07/evolution-iphone-apples-journey-iphone-5/ [Accessed 2 Feb. 2018].

Android Authority. (2018). *The history of Android OS: its name, origin and more*. [online] Available at: https://www.androidauthority.com/history-android-os-name-789433/ [Accessed 2 Feb. 2018].

Angular 5, 4 and 2 JavaScript ASP Net C# SQL Knockout Kendo Interview. (2018). *What Are the Advantages of Firebase?*. [online] Available at: https://www.code-sample.com/2017/09/what-are-advantages-of-firebase.html [Accessed 2 Feb. 2018].

Azure.microsoft.com. (2018). *What is Azure – Microsoft cloud service | Microsoft Azure*. [online] Available at: https://azure.microsoft.com/en-gb/overview/what-is-azure/ [Accessed 2 Feb. 2018].

Buzzle. (2018). *Modified Waterfall Model*. [online] Available at: https://www.buzzle.com/articles/modified-waterfall-model.html [Accessed 2 Feb. 2018].

Careerfoundry.com. (2018). *How To Design A Mobile App Using User Interface Design Principles*. [online] Available at: https://careerfoundry.com/en/blog/ui-design/how-to-design-a-mobile-app-using-user-interface-design-principles/ [Accessed 10 Feb. 2018].

Developer.android.com. (2018). *Android Studio Features | Android Studio*. [online] Available at: https://developer.android.com/studio/features.html [Accessed 2 Feb. 2018].

Developer.apple.com. (2018). *Xcode - Apple Developer*. [online] Available at: https://developer.apple.com/xcode/ [Accessed 2 Feb. 2018].

Firebase. (2018). *Firebase*. [online] Available at: https://firebase.google.com/ [Accessed 2 Feb. 2018].

Ibm.com. (2018). *Build Smarter Mobile Services and Apps | IBM Cloud*. [online] Available at: https://www.ibm.com/cloud/mobile [Accessed 2 Feb. 2018].

Ignitedsoftware.com. (2018). *Ignited Software - Great Coach Football*. [online] Available at: http://www.ignitedsoftware.com/products/greatcoach-football/ [Accessed 1 Feb. 2018].

Inf.ed.ac.uk. (2018). *Software Requirements*. [online] Available at: http://www.inf.ed.ac.uk/teaching/courses/cs2/LectureNotes/CS2Ah/SoftEng/se02.pdf [Accessed 3 Feb. 2018].

Istqbexamcertification.com. (2018). *What are the Software Development Life Cycle (SDLC) phases?*. [online] Available at: http://istqbexamcertification.com/what-are-the-software-development-life-cycle-sdlc-phases/ [Accessed 2 Feb. 2018].

Istqbexamcertification.com. (2018). *What is Iterative model- advantages, disadvantages and when to use it?*. [online] Available at: http://istqbexamcertification.com/what-is-iterative-model-advantages-disadvantages-and-when-to-use-it/ [Accessed 2 Feb. 2018].

Istqbexamcertification.com. (2018). *What is Spiral model- advantages, disadvantages and when to use it?*. [online] Available at: http://istqbexamcertification.com/what-is-spiral-model-advantages-disadvantages-and-when-to-use-it/ [Accessed 2 Feb. 2018].

Mobile Application Development: iOS vs Android vs Windows Phone. (2017). [online] Whalla Labs. Available at: http://whallalabs.com/mobile-application-development-ios-vs-android-vs-windows-phone/ [Accessed 2 Feb. 2018].

Mobile, W. (2017). *Android vs iOS vs Windows 10 Mobile: Which mobile operating system is best?*. [online] TrustedReviews. Available at: http://www.trustedreviews.com/opinions/which-mobile-operating-system-is-best [Accessed 2 Feb. 2018].

MSPoweruser. (2018). *A History of Windows Phone: The life and death of Microsoft's mobile platform - MSPoweruser*. [online] Available at: https://mspoweruser.com/a-history-of-windows-phone-the-road-to-threshold/ [Accessed 2 Feb. 2018].

Parker, M. (2018). *Android vs iOS vs Windows 10 Mobile: Which mobile operating system is best? | Trusted Reviews*. [online] Trusted Reviews. Available at: http://www.trustedreviews.com/opinion/which-mobile-operating-system-is-best-2928049 [Accessed 2 Feb. 2018].

Performasports.com. (2018). *Performa Sports | SPORTS PERFORMANCE ANALYSIS*. [online] Available at: https://performasports.com/ [Accessed 1 Feb. 2018].

Phone Arena. (2018). *What Firebase is and why it matters for Android apps*. [online] Available at: https://www.phonearena.com/news/What-Firebase-is-and-why-it-matters-for-Android-apps\_id81281 [Accessed 2 Feb. 2018].

PhoneGap, A. (2018). *Products*. [online] Phonegap.com. Available at: https://phonegap.com/products/ [Accessed 2 Feb. 2018].

Powell-Morse, A. (2018). *Waterfall Model: What Is It and When Should You Use It?*. [online] Airbrake Blog. Available at: https://airbrake.io/blog/sdlc/waterfall-model [Accessed 2 Feb. 2018].

Project Smart. (2018). *Requirements Gathering 101*. [online] Available at: https://www.projectsmart.co.uk/requirements-gathering.php [Accessed 3 Feb. 2018].

Samy, M. (2018). *Using SQLite Database with Android - CodeProject*. [online] Codeproject.com. Available at: https://www.codeproject.com/Articles/119293/Using-SQLite-Database-with-Android [Accessed 2 Feb. 2018].

SearchMobileComputing. (2018). *What is cross-platform mobile development? - Definition from WhatIs.com*. [online] Available at: http://searchmobilecomputing.techtarget.com/definition/cross-platform-mobile-development [Accessed 2 Feb. 2018].

Sehlhorst, S. (2018). *Ten Requirements Gathering Techniques*. [online] Tyner Blain. Available at: http://tynerblain.com/blog/2006/11/21/ten-requirements-gathering-techniques/ [Accessed 3 Feb. 2018].

Smartdraw.com. (2018). *Use Case Diagrams - Use Case Diagrams Online, Examples, and Tools*. [online] Available at: https://www.smartdraw.com/use-case-diagram/#whatisUseCase [Accessed 10 Feb. 2018].

Software Testing Fundamentals. (2018). *Verification vs Validation - Software Testing Fundamentals*. [online] Available at: http://softwaretestingfundamentals.com/verification-vs-validation/ [Accessed 3 Apr. 2018].

SportEasy.net. (2018). *SportEasy - Sports team management web & mobile app*. [online] Available at: https://www.sporteasy.net/en/home/ [Accessed 1 Feb. 2018].

Stafford, W. (2018). *The Importance of Requirements Gathering for Software Projects - KTL Solutions*. [online] KTL Solutions. Available at: https://www.ktlsolutions.com/project-management/the-importance-of-requirements-gathering-for-software-projects/ [Accessed 3 Feb. 2018].

Techopedia.com. (2018). *What is the Software Development Life Cycle (SDLC)? - Definition from Techopedia*. [online] Available at: https://www.techopedia.com/definition/22193/software-development-life-cycle-sdlc [Accessed 2 Feb. 2018].

Testingexcellence.com. (2018). *Cite a Website - Cite This For Me*. [online] Available at: https://www.testingexcellence.com/software-development-life-cycle-sdlc-phases/ [Accessed 2 Feb. 2018].

Ulf Eriksson. (2012). *Functional Requirements vs Non Functional Requirements.* Available: https://reqtest.com/requirements-blog/functional-vs-non-functional-requirements/. Last accessed 9th Feb 2018.

Webster, L. (2018). *Visual Studio IDE*. [online] Visual Studio. Available at: https://www.visualstudio.com/vs/ [Accessed 2 Feb. 2018].

Webster, L. (2018). *Xamarin App Development with Visual Studio | Visual Studio*. [online] Visual Studio. Available at: https://www.visualstudio.com/xamarin/ [Accessed 2 Feb. 2018].

Williams, R. (2018). *Apple iOS: a brief history*. [online] Telegraph.co.uk. Available at: http://www.telegraph.co.uk/technology/apple/11068420/Apple-iOS-a-brief-history.html [Accessed 2 Feb. 2018].

www.idc.com. (2018). *IDC: Smartphone OS Market Share*. [online] Available at: https://www.idc.com/promo/smartphone-market-share/os [Accessed 2 Feb. 2018].

www.rmcapability.com. (2011). *Probability-Impact Matrix (PIM).* Available: http://rmcapability.com/resources/Capability+Guidance+Sheet+-+Probability-Impact+Matrix.pdf. Last accessed 9th Feb 2018.

www.tutorialspoint.com. (2018). *SDLC Agile Model*. [online] Available at: https://www.tutorialspoint.com/sdlc/sdlc\_agile\_model.htm [Accessed 2 Feb. 2018].

www.tutorialspoint.com. (2018). *SDLC Iterative Model*. [online] Available at: https://www.tutorialspoint.com/sdlc/sdlc\_iterative\_model.htm [Accessed 2 Feb. 2018].

www.tutorialspoint.com. (2018). *Software Requirements*. [online] Available at: https://www.tutorialspoint.com/software\_engineering/software\_requirements.htm [Accessed 3 Feb. 2018].

Zielczynski, P. (2018). *1.4 Characteristics of a Good Requirement | Requirements Management Using IBM® Rational® RequisitePro® | InformIT*. [online] Informit.com. Available at: http://www.informit.com/articles/article.aspx?p=1152528&seqNum=4 [Accessed 5 Apr. 2018].

Appendices

|  |  |  |
| --- | --- | --- |
| **APPENDIX** | **TITLE** | **PAGE** |
| Appendix A | Blank Requirements Gathering Questionnaire | 106 |
| Appendix B | Requirements Gathering Questionnaire Results | 108 |
| Appendix C | Gantt Chart | 110 |
| Appendix D | Work Breakdown Structure | 111 |
| Appendix E | Test Cases with Results | 112 |
| Appendix F | Blank System Evaluation Survey | 128 |

**APPENDIX A**

**QUESTIONNAIRE**

My name is Ryan Butler and I am developing a mobile application for your team which will enable match statistics to be collected and displayed to coaches and players. I would like to get a better understanding of how this is done for your team at present.

Thank you for taking the time to complete this survey!

1. What mobile device to you own?

[ ] Android [ ] iPhone [ ] Windows [ ] Other

1. Do you currently have access to the statistics for your team?

[ ] Yes [ ] No [ ] Don’t Know

1. Do you currently have access to the statistics related to you?

[ ] Yes [ ] No [ ] Don’t Know

1. Is there any way to keep track of leader boards within your club? (i.e. Top Goal Scorer, Top Assists etc)

[ ] Yes [ ] No [ ] Don’t Know

1. Do you think it would be/is beneficial to have access to this information?

[ ] Yes [ ] No [ ] Don’t Know

1. How do you currently keep track of the fixture list?

[ ] Simply Remember [ ] Coach tells us week by week

[ ] Diary [ ] Online Fixture List

[ ] Other (*Please specify*): \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_

1. How do you currently keep track of your training schedule?

[ ] Simply Remember [ ] Coach tells us week by week

[ ] Diary [ ] It does not change

[ ] Other (*Please specify*): \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_

1. What statistics are important to you? (Select 3)

[ ] Dribbles [ ] Possession [ ] Shots

[ ] Tackles [ ] Distance Ran [ ] Assists

[ ] Headers [ ] Fouls [ ] Red/Yellow Card

[ ] Corners [ ] Substitutions [ ] GK

[ ] Other (*Please specify*): \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_

1. Would you like to be able to share statistics on social media?

[ ] Yes [ ] No [ ] Don’t Know

1. With this mobile application in mind, is there any features you would like to suggest?

|  |
| --- |
|  |

1. Any other comments?

|  |
| --- |
|  |

Signed:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**APPENDIX B**

**QUESTIONNAIRE**

My name is Ryan Butler and I am developing a mobile application for your team which will enable match statistics to be collected and displayed to coaches and players. I would like to get a better understanding of how this is done for your team at present.

Thank you for taking the time to complete this survey!

1. What mobile device to you own?

[**16**] Android [**4**] iPhone [**0**] Windows [**0**] Other

1. Do you currently have access to the statistics for your team?

[**0**] Yes [**18**] No [**2**] Don’t Know

1. Do you currently have access to the statistics related to you?

[**0**] Yes [**17**] No [**3**] Don’t Know

1. Is there any way to keep track of leader boards within your club? (i.e. Top Goal Scorer, Top Assists etc)

[**0**] Yes [**18**] No [**2**] Don’t Know

1. Do you think it would be/is beneficial to have access to this information?

[**18**] Yes [**1**] No [**1**] Don’t Know

1. How do you currently keep track of the fixture list?

[**2**] Simply Remember [**13**] Coach tells us week by week

[**1**] Diary [**4**] Online Fixture List

[**0**] Other (*Please specify*): \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_

1. How do you currently keep track of your training schedule?

[**3**] Simply Remember [**14**] Coach tells us week by week

[**1**] Diary [**2**] It does not change

[**0**] Other (*Please specify*): \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_

1. What statistics are important to you? (Select 3)

[**8**] Dribbles [**5**] Possession [**10**] Shots

[**7**] Tackles [**2**] Distance Ran [**7**] Assists

[**6**] Headers [**5**] Fouls [**5**] Red/Yellow Card

[**3**] Corners [**1**] Substitutions [**1**] GK

[**0**] Other (*Please specify*): \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_

1. Would you like to be able to share statistics on social media?

[**8**] Yes [**10**] No [**2**] Don’t Know

1. With this mobile application in mind, is there any features you would like to suggest?

|  |
| --- |
| * Heat map (x1) * Chat/Messaging (x5) * Player Ratings (x3) * Skill graphs “like FIFA” (x1) |

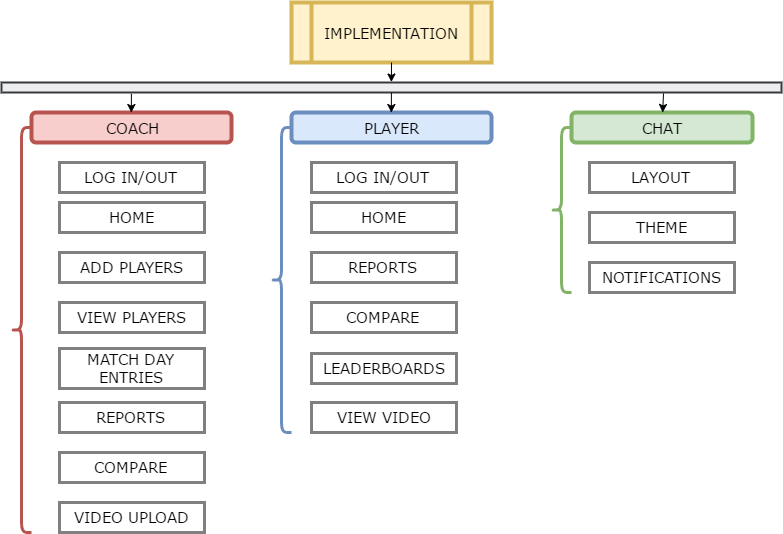
1. Any other comments?

|  |
| --- |
| N/A |

Signed:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**APPENDIX C**



**APPENDIX D**

**APPENDIX E**

|  |  |  |  |
| --- | --- | --- | --- |
| **TEST ID** | **DESCRIPTION** | | |
| 1 | New coaches should be able to register a new account with the system | | |
| INPUT | First Name: The  Last Name: Example  Team Name: Team EG  Email: example@email.com  Password: Example01  Confirm Password: Example01 | | |
| EXP RESULT | - User will be able to navigate to the Register Screen successfully.  - User will be able to complete form with no issues  - Once input data is complete, it will be accepted and the user’s account will be created and they will be navigated to their Home screen | ACTUAL RESULT | As Expected. |
| PASS/FAIL | | | Pass |

|  |  |  |  |
| --- | --- | --- | --- |
| **TEST ID** | **DESCRIPTION** | | |
| 2 | Registered Coaches should be able to log in to the system | | |
| INPUT | Email: example@email.com  Password: Example01 | | |
| EXP RESULT | - User should be able to enter details without issues  - User should be navigated to their home screen | ACTUAL RESULT | As Expected. |
| PASS/FAIL | | | Pass |

|  |  |  |  |
| --- | --- | --- | --- |
| **TEST ID** | **DESCRIPTION** | | |
| 3 | Coaches should be able to log out of the system | | |
| INPUT | User selecting ‘Log Out’ Option from Navigation Bar | | |
| EXP RESULT | - User should be logged out of the system and navigated back to ‘Log In’ screen | ACTUAL RESULT | As Expected. |
| PASS/FAIL | | | Pass |

|  |  |  |  |
| --- | --- | --- | --- |
| **TEST ID** | **DESCRIPTION** | | |
| 4 | Coaches should be able to upload a profile photo | | |
| INPUT | Image selection on Device | | |
| EXP RESULT | - User should be able to navigate to the ‘My Account’ Screen  - User should be able to change their profile image from this screen and select the ‘Submit’ button to confirm changes’ | ACTUAL RESULT | As Expected. |
| PASS/FAIL | | | Pass |

|  |  |  |  |
| --- | --- | --- | --- |
| **TEST ID** | **DESCRIPTION** | | |
| 5 | Coaches should be able to upload a club badge photo | | |
| INPUT | Image selection on Device | | |
| EXP RESULT | - User should be able to navigate to the ‘My Account’ Screen  - User should be able to change their club badge image from this screen and select the ‘Submit’ button to confirm changes’ | ACTUAL RESULT | As Expected. |
| PASS/FAIL | | | Pass |

|  |  |  |  |
| --- | --- | --- | --- |
| **TEST ID** | **DESCRIPTION** | | |
| 6 | Coaches should be able to add new Team members to the system | | |
| INPUT | First Name: Hello  Last Name: World  DOB: 01/02/2000  Email: world@email.com  Password: Example01  Position: DEF | | |
| EXP RESULT | - User should be able to navigate to ‘Add Player’ screen without issues  - User should be able to enter details without errors  - When submitted, details should be accepted and a player profile created  Coach will be navigated back to Home screen, with Toast confirmation of player added | ACTUAL RESULT | As Expected. |
| PASS/FAIL | | | Pass |

|  |  |  |  |
| --- | --- | --- | --- |
| **TEST ID** | **DESCRIPTION** | | |
| 8 | Coaches should be able to view their team calendar | | |
| INPUT | N/A | | |
| EXP RESULT | - User should be able to view their team calendar every time they navigate to their home screen | ACTUAL RESULT | As Expected |
| PASS/FAIL | | | Pass |

|  |  |  |  |
| --- | --- | --- | --- |
| **TEST ID** | **DESCRIPTION** | | |
| 7 | Coaches should be able to view players information | | |
| INPUT | N/A | | |
| EXP RESULT | - User should be able to navigate to ‘View Players screen without issues | ACTUAL RESULT | As Expected |
| PASS/FAIL | | | Pass |

|  |  |  |  |
| --- | --- | --- | --- |
| **TEST ID** | **DESCRIPTION** | | |
| 9 | Coaches should be able to add Match Days to their team calendar | | |
| INPUT | Date Picker Input by User – 05/03/2018 | | |
| EXP RESULT | - User should be able to select the ‘Add Match’ button to initiate process  - Date Picker Dialog should appear, allowing user to select date easily  - Once confirmed, the date should be added to calendar, highlighted by addition of appropriate icon on Team Calendar | ACTUAL RESULT | As Expected |
| PASS/FAIL | | | Pass |

|  |  |  |  |
| --- | --- | --- | --- |
| **TEST ID** | **DESCRIPTION** | | |
| 10 | Coaches should be able to add Training sessions to their team calendar | | |
| INPUT | Date Picker Input by User – 07/03/2018 | | |
| EXP RESULT | - User should be able to select the ‘Add Training Session button to initiate process  - Date Picker Dialog should appear, allowing user to select date easily  - Once confirmed, the date should be added to calendar, highlighted by addition of appropriate icon on Team Calendar | ACTUAL RESULT | As Expected |
| PASS/FAIL | | | Pass |

|  |  |  |  |
| --- | --- | --- | --- |
| **TEST ID** | **DESCRIPTION** | | |
| 11 | Coaches should be able to log statistics to Match Day entries | | |
| INPUT | * Submitted Date for Match – 05/03/2018 * User selecting Player – Hello World * User selecting Event – Pass Complete | | |
| EXP RESULT | - User should be able to navigate and select their intended date with no issues  - User should be able to enter match information  User should be able to log input data by selecting player from list, followed by event  - Data should be stored in database, with real-time feedback given to user | ACTUAL RESULT | As Expected |
| PASS/FAIL | | | Pass |

|  |  |  |  |
| --- | --- | --- | --- |
| **TEST ID** | **DESCRIPTION** | | |
| 12 | Coaches should be able to view overall team statistics for completed match day entries | | |
| INPUT | * User navigation to ‘View Team Stats’ option using Navigation Bar. * User selection of completed Match Day Date from List | | |
| EXP RESULT | - User should be able to navigate and select their intended date with no issues  - User should be able to use a bottom navigation bar to view overall team statistics for selected match day | ACTUAL RESULT | As Expected |
| PASS/FAIL | | | Pass |

|  |  |  |  |
| --- | --- | --- | --- |
| **TEST ID** | **DESCRIPTION** | | |
| 13 | Coaches should be able to view individual player statistics for completed match day entries | | |
| INPUT | * User navigation to ‘View Player Stats’ option using Navigation Bar. * User selection of completed Match Day Date from List | | |
| EXP RESULT | - User should be able to navigate and select their intended date with no issues  - User should be able to use a button list to view individual player statistics for selected match day | ACTUAL RESULT | As Expected |
| PASS/FAIL | | | Pass |

|  |  |  |  |
| --- | --- | --- | --- |
| **TEST ID** | **DESCRIPTION** | | |
| 14 | Coaches should be able to review team performances of completed match day entries | | |
| INPUT | * User navigation to ‘Team Performance’ option using Navigation Bar. * User selection of completed Match Days Date from List | | |
| EXP RESULT | - User should be able to navigate and select their intended dates with no issues  - User should be able to use a bottom navigation bar to compare overall team statistics for selected match days  - Graphs should be displayed with animation | ACTUAL RESULT | As Expected |
| PASS/FAIL | | | Pass |

|  |  |  |  |
| --- | --- | --- | --- |
| **TEST ID** | **DESCRIPTION** | | |
| 15 | Coaches should be able to review individual player performances of completed match day entries | | |
| INPUT | * User navigation to ‘Player Performance’ option using Navigation Bar. * User selection of completed Match Day Date from List | | |
| EXP RESULT | - User should be able to navigate and select their intended date with no issues  - User should be able to use a button list to view individual player statistics for selected match day | ACTUAL RESULT | As Expected |
| PASS/FAIL | | | Pass |

|  |  |  |  |
| --- | --- | --- | --- |
| **TEST ID** | **DESCRIPTION** | | |
| 16 | Coaches should be able to send/receive messages to their players | | |
| INPUT | * User navigation to ‘Messages’ option using Navigation Bar. * User selection of player from a list * Sending message:   “Test Message” | | |
| EXP RESULT | - User should be able to navigate and select the user they want to message  - User should be able to enter message and use ‘Send’ button.  - When sent it should appear to the user, as well as the recipient | ACTUAL RESULT | As Expected |
| PASS/FAIL | | | Pass |

|  |  |  |  |
| --- | --- | --- | --- |
| **TEST ID** | **DESCRIPTION** | | |
| 17 | Coaches should be able to upload videos to the system | | |
| INPUT | * User navigation to ‘Video Upload’ option using Navigation Bar. * User selection of video file | | |
| EXP RESULT | - User should be able to navigate to Video Upload screen with no issues  - User should be able to select a video file to upload  - When a user selects the ‘Upload’ button they should be able to view the progress of the upload  -When upload is complete, user is navigated to ‘Home’ screen with Toast indicating upload success | ACTUAL RESULT | As Expected |
| PASS/FAIL | | | Pass |

|  |  |  |  |
| --- | --- | --- | --- |
| **TEST ID** | **DESCRIPTION** | | |
| 18 | Coaches should be able to view uploaded videos | | |
| INPUT | * User navigation to ‘View Videos’ option using Navigation Bar. * User selection of video file from List | | |
| EXP RESULT | - User should be able to navigate to View Videos screen with no issues  - User should be able to select previously uploaded video from list  - Once selected the video should start playing | ACTUAL RESULT | As Expected |
| PASS/FAIL | | | Pass |

|  |  |  |  |
| --- | --- | --- | --- |
| **TEST ID** | **DESCRIPTION** | | |
| 19 | Players should be able to log in to the system once created by their coach | | |
| INPUT | Email: world@email.com  Password: Example01 | | |
| EXP RESULT | - User should be able to enter details without issues  - User should be navigated to their home screen | ACTUAL RESULT | As Expected |
| PASS/FAIL | | | Pass |

|  |  |  |  |
| --- | --- | --- | --- |
| **TEST ID** | **DESCRIPTION** | | |
| 20 | Players should be able to log out of the system | | |
| INPUT | User selecting ‘Log Out’ Option from Navigation Bar | | |
| EXP RESULT | - User should be logged out of the system and navigated back to ‘Log In’ screen | EXP RESULT | - User should be logged out of the system and navigated back to ‘Log In’ screen |
| PASS/FAIL | | | Pass |

|  |  |  |  |
| --- | --- | --- | --- |
| **TEST ID** | **DESCRIPTION** | | |
| 21 | Players should be able to upload a profile photo | | |
| INPUT | Image selection on Device | | |
| EXP RESULT | - User should be able to navigate to the ‘My Account’ Screen  - User should be able to change their profile image from this screen and select the ‘Submit’ button to confirm changes’ | ACTUAL RESULT | As Expected. |
| PASS/FAIL | | | Pass |

|  |  |  |  |
| --- | --- | --- | --- |
| **TEST ID** | **DESCRIPTION** | | |
| 22 | Players should be able to view their team calendar | | |
| INPUT | N/A | | |
| EXP RESULT | - User should be able to view their team calendar every time they navigate to their home screen | ACTUAL RESULT | As Expected |
| PASS/FAIL | | | Pass |

|  |  |  |  |
| --- | --- | --- | --- |
| **TEST ID** | **DESCRIPTION** | | |
| 23 | Players should be able to view overall team statistics for completed match day entries | | |
| INPUT | * User navigation to ‘View Team Stats’ option using Navigation Bar. * User selection of completed Match Day Date from List | | |
| EXP RESULT | - User should be able to navigate and select their intended date with no issues  - User should be able to use a bottom navigation bar to view overall team statistics for selected match day | ACTUAL RESULT | As Expected |
| PASS/FAIL | | | Pass |

|  |  |  |  |
| --- | --- | --- | --- |
| **TEST ID** | **DESCRIPTION** | | |
| 24 | Players should be able to view individual statistics for completed match day entries | | |
| INPUT | * User navigation to ‘View Player Stats’ option using Navigation Bar. * User selection of completed Match Day Date from List | | |
| EXP RESULT | - User should be able to navigate and select their intended date with no issues  - User should be able to use a button list to view individual player statistics for selected match day | ACTUAL RESULT | As Expected |
| PASS/FAIL | | | Pass |

|  |  |  |  |
| --- | --- | --- | --- |
| **TEST ID** | **DESCRIPTION** | | |
| 25 | Players should be able to review individual player performances of completed match day entries | | |
| INPUT | * User navigation to ‘Player Performance’ option using Navigation Bar. * User selection of completed Match Day Date from List | | |
| EXP RESULT | - User should be able to navigate and select their intended date with no issues  - User should be able to use a button list to view individual player statistics for selected match day | ACTUAL RESULT | As Expected |
| PASS/FAIL | | | Pass |

|  |  |  |  |
| --- | --- | --- | --- |
| **TEST ID** | **DESCRIPTION** | | |
| 26 | Players should be able to send/receive messages to their players | | |
| INPUT | * User navigation to ‘Chat to Coach’ option using Navigation Bar. * Sending message:   “Test Message” | | |
| EXP RESULT | - User should be able to navigate to Chat facility with no issues  - User should be able to enter message and use ‘Send’ button.  - When sent it should appear to the user, as well as the recipient | ACTUAL RESULT | As Expected |
| PASS/FAIL | | | Pass |

|  |  |  |  |
| --- | --- | --- | --- |
| **TEST ID** | **DESCRIPTION** | | |
| 27 | Players should be able to view uploaded videos | | |
| INPUT | * User navigation to ‘View Videos’ option using Navigation Bar. * User selection of video file from List | | |
| EXP RESULT | - User should be able to navigate to View Videos screen with no issues  - User should be able to select video from list of videos uploaded by their coach  - Once selected the video should start playing | ACTUAL RESULT | As Expected |
| PASS/FAIL | | | Pass |

|  |  |  |  |
| --- | --- | --- | --- |
| **TEST ID** | **DESCRIPTION** | | |
| 28 | All users should be able to reset a forgotten password | | |
| INPUT | Email: world@email.com | | |
| EXP RESULT | - User should be able to access facility through Login screen  - User should be able to enter email address to receive password reset email  - Once email is received, user can reset password using link provided | ACTUAL RESULT | As Expected |
| PASS/FAIL | | | Pass |

|  |  |  |  |
| --- | --- | --- | --- |
| **TEST ID** | **DESCRIPTION** | | |
| 29 | All users should be able to change their password | | |
| INPUT | Current Password: Example01  New Password: TestPassword  Confirm New: TestPassword | | |
| EXP RESULT | - User should be able to access facility through My Account screen  - Once user confirms current password and enters a new password, user is redirected to Home Screen | ACTUAL RESULT | As Expected |
| PASS/FAIL | | | Pass |

|  |  |  |  |
| --- | --- | --- | --- |
| **TEST ID** | **DESCRIPTION** | | |
| 30 | Application should have a consistent theme throughout | | |
| INPUT | N/A | | |
| EXP RESULT | No changes in design or layout throughout.  Use of the same colours, text face, animations throughout | ACTUAL RESULT | As Expected.  All elements of application use the colour scheme decided in chapter 4 of project report. |
| PASS/FAIL | | | Pass |

|  |  |  |  |
| --- | --- | --- | --- |
| **TEST ID** | **DESCRIPTION** | | |
| 31 | Application should display correctly on all varying screen sizes | | |
| INPUT | N/A | | |
| EXP RESULT | No errors in display on different sized devices | ACTUAL RESULT | As Expected.  Tested on 3 different devices with no noticeable errors in display |
| PASS/FAIL | | | Pass |

|  |  |  |  |
| --- | --- | --- | --- |
| **TEST ID** | **DESCRIPTION** | | |
| 32 | Application should react appropriately to orientation changes | | |
| INPUT | N/A | | |
| EXP RESULT | Application displays correctly in both portrait and landscape orientation | ACTUAL RESULT | It was decided to fix screen orientation within the application due to the nature of data that needed to be displayed.  It was attempted to accommodate orientation changes but layouts were untidy and cramped |
| PASS/FAIL | | | Pass |

|  |  |  |  |
| --- | --- | --- | --- |
| **TEST ID** | **DESCRIPTION** | | |
| 33 | The application should be easily navigated | | |
| INPUT | N/A | | |
| EXP RESULT | Application should have a logical and convenient layout and navigation method for users. | ACTUAL RESULT | As Expected.  Users can access any element through the Navigation Bar on any screen |
| PASS/FAIL | | | Pass |

|  |  |  |  |
| --- | --- | --- | --- |
| **TEST ID** | **DESCRIPTION** | | |
| 34 | The application should not crash | | |
| INPUT | N/A | | |
| EXP RESULT | The application should not crash unexpectedly while in use | ACTUAL RESULT | As Expected.  Application does not fail during use. |
| PASS/FAIL | | | Pass |

|  |  |  |  |
| --- | --- | --- | --- |
| **TEST ID** | **DESCRIPTION** | | |
| 35 | The application should be capable of running in the background | | |
| INPUT | N/A | | |
| EXP RESULT | The application should not end, or loose progress when running in the background | ACTUAL RESULT | As Expected.  Application saves changes and progress when running in the background |
| PASS/FAIL | | | Pass |

|  |  |  |  |
| --- | --- | --- | --- |
| **TEST ID** | **DESCRIPTION** | | |
| 36 | Users should not be able to access data without logging in | | |
| INPUT | N/A | | |
| EXP RESULT | Users not registered to the system should not be able to access data | ACTUAL RESULT | As Expected.  Users without valid log in credentials cannot access the application’s data |
| PASS/FAIL | | | Pass |

|  |  |  |  |
| --- | --- | --- | --- |
| **TEST ID** | **DESCRIPTION** | | |
| 37 | Passwords of all users should be stored securely | | |
| INPUT | N/A | | |
| EXP RESULT | Once registered on the application, user’s passwords should be securely encrypted. | ACTUAL RESULT | As Expected.  User’s passwords are encrypted automatically through the Firebase Authentication Plugin. |
| PASS/FAIL | | | Pass |

|  |  |  |  |
| --- | --- | --- | --- |
| **TEST ID** | **DESCRIPTION** | | |
| 38 | The application should conform to best practice and design principles for chosen devices | | |
| INPUT | N/A | | |
| EXP RESULT | Application should conform to Material design, utilising Android’s best practices to create a satisfactory application | ACTUAL RESULT | As Expected.  All elements within the application are defined by Android for use in their devices applications |
| PASS/FAIL | | | Pass |

|  |  |  |  |
| --- | --- | --- | --- |
| **TEST ID** | **DESCRIPTION** | | |
| 39 | The finished system should be completed by 22nd April 2018 | | |
| INPUT | N/A | | |
| EXP RESULT | The system should be fully functional by 22nd April 2018 | ACTUAL RESULT | As Expected. |
| PASS/FAIL | | | Pass |

**APPENDIX F**

***UNIVERSITY OF ULSTER***

**EVALUATION OF J UNO**

1. **I found the application easy to use and navigate.**

**(Please tick (✓) one answer)**

[ ] Strongly Agree [ ] Agree [ ] Neutral

[ ] Strongly Disagree [ ] Disagree [ ] Do not own Android Device

1. **I found match information clear and easily understood.**

**(Please tick (✓) one answer)**

[ ] Strongly Agree [ ] Agree [ ] Neutral

[ ] Strongly Disagree [ ] Disagree [ ] Do not own Android Device

1. **Is the application an improvement from the current system?**

**(Please tick (✓) one answer)**

[ ] Strongly Agree [ ] Agree [ ] Neutral

[ ] Strongly Disagree [ ] Disagree [ ] Do not own Android Device

1. **How is this an improvement?**

**(Please tick (✓) all appropriate answers)**

[ ] Not an improvement

[ ] Improves Knowledge on personal performance

[ ] Increases competition between team mates

[ ] Adds a social aspect to team performance reviews

[ ] Able to access information 24/7

[ ] Makes it easier to compare personal performances

[ ] Provides a more tangible method of providing feedback

[ ] Other: Please specify: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **How frequently would you use the new application?**

**(Please tick (✓) one answer)**

[ ] Daily [ ] Weekly [ ] Fortnightly

[ ] Monthly [ ] Never [ ] Checked after every game

1. **Did you experience any issues/errors whilst using the application?**

**(Please tick (✓) one answer)**

[ ] Yes Please specify: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[ ] No

1. **I would recommend this application to other teams.**

**(Please tick (✓) one answer)**

[ ] Yes [ ] No [ ] N/A

1. **How could this application be improved?**

**Signed: ……………………………………………………… Date: ……………………………….**