Shimmer3 IMU Application:

Data Gathering and Injury Mitigation in Rugby

Final Year Project
B.Sc.(Hons) in Software Development

by
Justin Servis
David Gallagher

May 21, 2021

Advised by Damien Costello
Department of Computer Science and Applied Physics Galway-Mayo Institute of Technology (GMIT)

Contents

List of Figures

Chapter 1

Introduction

This project aims to provide a means to prevent injury in sports, with a specific target audience of Rugby athletes. The developers have used an inertial measurement unit (IMU) to determine the orientation of an athletes body, with particular attention to the orientation of the spine, in an attempt to gather meaningful data for analysis during and post training. The objective is to determine if the athlete is performing (rugby) tackles in a safe and controlled manner such as not to incur any personal physical injury. Our goal is to provide a technical solution which incorporates the IMU and a cross-platform software application which enables a coach or an athlete to determine if a safe tackle has been performed. The application will target the wearer of the IMU and stream the live data which can then be recorded and replayed for analysis.

This is an important area of research in Rugby specifically as there is a high incidence of collisions in the sport in general. This project attempts to form part of a larger area of research into the athletes posture during a tackle. It should be viewed as a tool to enhance the ability of the researchers in gathering meaningful data on tackling in Rugby. Specifically this version of the project will target training scenarios, where known variables such as a correct tackling posture can be recorded and measured against training data. The training data can be reviewed by a coach and athlete alike, where they will be able to distinguish between a safe tackle which conforms to accepted standards, and a risky or unknown safety level, tackle which should be avoided. We are hopeful that the coach and athlete will gain valuable insight into tackle techniques and athlete behaviour, and that this application might drive safety in the sport forwards.

During the final stages of this project, we conducted interviews with sports and rugby athletes who were enthusiastic about the application and its potential to impact the sport and hopefully be a catalyst for increasing safety.
The developers hope to show that this project meets the goals set out in this introductory section. Additionally, it will be a priority to explain the rationale behind decision making from both a technical and a design standpoint, adhering to best practices in both schools of thought. Further to the design and implementation, any profound learnings will be documented along with any shortcomings. It is our intention to juxtapose these two paradigms so that any future development of this or other similar projects can draw for this experience.

1.1 Project Background

The current project builds upon a prototype research and development project from 3rd year. The developers were offered some hardware (the Shimmer3 IMU), a Software Development Kit (SDK), and a high concept pertaining to the prevention of injury in rugby. The terms were broad and allowed for much iteration over the course of the project in developing a possible solution. The initial goal was to connect to any application and read the data from the device. Much of this initial project was exploratory in nature, working with the applications provided in the Shimmer SDK and streaming data from the IMU. After some weeks of interrogation and investigation, the developers were able to extract a trimmed down class library with the means to extract data from the IMU and integrate this class library into another application as a reference.

Two example projects were undertaken to test this approach:

- C# WPF Application.
- Unity3D Application.
- Microsoft Hololens compatible project.

This approach lead to further developments. The application in WPF was reasonable, however drawing frames wasn't ideal and proved to be a bit slow in comparison to the Unity3D application, which is designed to draw frames more efficiently and with its integrated physics engine was able to handle the angular calculations quickly and more efficiently than our manual conversions. The Hololens project, developed in Unity3D was initially promising. However due to time constraints and complexities with the Hololens requiring WiFi in order to deploy applications it was deemed that it would be infeasible for the end goals that the project demanded.

The developers provided a proof of concept application by the end of the project. It was a lightweight application which connected the IMU directly to a 3D model in the Unity game engine. The IMU data was received from the IMU and used to manipulate a rudimentary 3D model in real time, converting the axial output from the IMU to Euler angles and applying those values to the 3D model. The success of this research and exploration lead to this current project.

Through the course of the project, numerous challenges and potential development opportunities were identified, many of which would be outside the scope of the research and development project. These were outlined for further development and submitted at the end of the project in the review, documentation and presentation of the findings from the research. Subsequently, the developers were approached by the project supervisor, Damien, and lecturer Ed Daly from the Sports Science department who were interested in pursuing this further. They inquired if we would consider developing the concept in more detail and documenting our research and findings as a final year project.

1.2 Project Objectives

Defining the Objectives As previously mentioned in the project Background, scope for further development was identified throughout the initial research project. Of core interest to the developers and the supervisors involved would be the development of a mobile application which could leverage the Shimmer units potential as a sensor based recording system for training purposes. Central to this development would be the availability of the application. One of the major drawbacks of developing for the Hololens is the price tag. This project aims to provide an effective training tool and make it available through a mobile application, ideally cross platform so that any mobile OS is supported, i.e. iOS and Android. This would allow widespread use of the application, at least for any one with the shimmer unit.

The developers would pair up with final year sports students and collaborate their efforts to develop a solution that met the following goals / requirements:

- Deploy a mobile application.
- Connect the application to the Shimmer3 IMU via Bluetooth.
- Focus on gathering real-time athlete movement data.
- Specific attention to the angle of incline and rotation of the athletes spine.
- Application should include the following features:
  - Enable / Disable Bluetooth.
  - Recording live Data from the IMU.
  - Store data for future analysis.

  - Playback live or pre-recorded data from the IMU.
  - Provide feedback to the user about the angles streaming from the IMU.
- Further application with a means to add or remove athlete profiles to a central database.
- Analysis of the data, pre and post impact to discover meaningful indicators of potential injuries.
- Collaborate with students from the Department of Sports and Exercise Science in a cross-department project.

Meeting the Objectives The goals set out herein are realistic and achievable. There are two developers for the project. Both developers have relevant experience having produced the original prototype application which serves as a proof of concept and a platform from which to launch the more in-depth development process. The developers are determined to provide a robust solution which incorporates key learnings from the 3 years of prior study. Best practices will be followed through each step of development in order to produce a solution that meets the objectives as outlined above.

Moreover, the students collaborating from the sports department are keen to be involved with the project and have experience in working with similar software to the proposed application and as such, will be able to provide expert guidance with regards to the real world application of this software. This includes plans for standardized testing with athletes to gather data and details a methodology from which to draw conclusions about the results of the tests. The results are important to determine if the application delivers reliable and accurate data. Comparing the results against known benchmarks from other software will be a crucial part of the collaboration.

While the original proof of concept consisted of a basic application built in the Unity3D engine as described earlier, meeting the new objectives would require reworking the original concept from the ground up. The new objectives incorporate the future development plans of the original prototype, and supplement that with concepts in line with professional application development.

Measuring Success and Shortcomings From a functional perspective, measuring the success of the application will involve setting out a list of expected behaviours and testing if the application performs as expected. It will also mean looking at how the functionality is presented to the user. Questions such as 'Is it intuitive and clearly explained?' will help determine if the application delivers the intended functionality in an understandable fashion. The ability of the application to deliver the expected measuring capability will inevitably be a collaborative effort from both developers and sports researchers.

The developers will strive to deliver a means to record the data accurately and store is securely. While the sports research will determine feasibility of the recorded data through comparative testing against known benchmarks and standards. Further measurements of success will be in the form of delivering upon the objectives in a literal sense, as well as identifying possible avenues which were discovered through the iterative development process. Similarly, the above testing will enable us to find and describe shortcomings with the project. Did scope creep occur or were some objectives too ambiguous? Or where, due to time restrictions or other complications, some promising avenues were left unfinished or undiscovered.

1.3 Document Layout/Overview

Overview of this document

- Methodology - A retrospective look at the development process.
  - Here we outline our approach to the research of the Windows and Android applications.
  - We will elaborate on gathering the requirements for the project.

  - Collaborating with the students from the sports department.
  - General internal collaboration between the developers.
  - Expand on how we intend to deliver the project.

- Technology Review - Research and Review of the technology used to develop the project.
  - Overview and background research on the development process.
  - Description of the technologies in detail.
  - Comparison between similar technologies.
  - Rationale for choosing one technology over another.
  - Review and research into other technologies with regards to wearable sensors and their role in mitigating injuries in athletes.

- System Design - Describes the major architecture of the solution.
  - Architectural diagrams.
  - UML class diagrams.
  - Code snippets explaining implementation rationale.

- System Evaluation - An evaluation of the completed project.
  - Testing Plans for the project.
  - A retrospective analysis of the goals set out in the Introduction and the Methodology compared against the completed application.
  - Identifies limitations within the project.
  - General insight into the project.

- Conclusion - What we learned from the development.
  - Brief summary of learnings.
  - Outlines discoveries and unforeseen issues..?
  - Future development opportunities.

Chapter 2

Methodology

2.1 Overview

Our overall approach was iterative, as we met with our supervisor *Damien Costello* weekly, collaborated with two final year sports science students, *Emma* and *Orna* for the initial two months of the project, and maintained a relationship with *Ed Daly*, lecturer and supervisor in *Sports and Exercise Science* from the *Department of Natural Sciences* who was involved with our initial third year research project. Ed, Emma and Orna would assist us in focusing the application towards the end user. Additionally, Ed would also put us in contact with a group of second year sports students for the purposes of testing our prototype application and gaining valuable feedback on functionality, usability and UI design. For the purposes of our methodology, we will refer to Ed, Emma and Orna as our clients.

Our development plan was:

- Meet with supervisor and pitch our ideas for developing the application.
- Meet with our clients to establish User requirements.
- Research technology with which to develop the project.
- Meet with supervisor and clients weekly or bi-monthly to discuss progress.

2.2 Establishing Client Requirements

We took a qualitative approach to gathering the project requirements. This was the best approach we could have taken in our opinion, as we felt this would enable us to assemble realistic project requirements. Empowering the clients with some decision making at an early stage in the project was important for us to realise the expectations for the end product. This process informed our approach as we would research the best way to provide the client's desired features and determine feasibility based on the different technologies we had considered. We would then set expectations based on our time availability and what we thought would be achievable.

We arranged an initial meeting with our supervisor and clients in order to establish the scope of the project. During this meeting we agreed on some basic functionality expected from the application. We discussed other products which the clients were familiar with and the features they provided. We knew from this meeting that there were other products which provided similar features, however these features were not self contained within a single application. Our clients were interested in an application which provided features they were familiar with from a set of different applications, in one single practical solution.

The solution we agreed on was as follows:

- Deploy as a mobile application.
  - Android.

  - iOS.

- Store athlete data.
  - Athlete age.
  - Athlete training age.
  - Athlete gender.
  - Athlete Height.
  - Athlete Weight.
- Record data from an athlete in motion.
  - Store training records for each athlete.
  - Each Athlete would have a training record composed of recorded training sessions.
- Store that data to a database.
  - Database would need to be online and accessible at all times.
  - Accessibility would need to extend at a minimum to a coach or training supervisor.
- Reload an athletes data to the application.
- Replay the data in the application.
- Implement an Admin database where a coach or trainer can add and remove athletes.
- Develop a separate application to add athlete data to the database, ideally this would be a cross-platform desktop and mobile application.

2.3 Establishing Developer Requirements

Once we had our initial meeting, we had established some basics and set some expectations. We began our research to decide on which technology would best suit our needs and enable us to meet the requirements. We investigated a number of potential solutions. Ultimately we settled on a small selection of technology which we felt suited the needs of the application and would benefit any future development needs of the application.

- Developing the Mobile Application.
  - Xamarin Desktop Application.
    * Add athlete profiles to Database.
    * Update or Remove profiles.
    * View profiles and training records for athletes.
    * View individual training sessions in an athletes training record.
    * Add training data to athletes.
  - Unity3d.
    * Select athlete profiles.
    * Record training session for that user.
    * Upload session to athlete profile.
    * Load previously recorded session for that athlete.
    * Playback session for that athlete, from memory or loaded from database.
  - Android Studio.
    * Investigate shimmer for android application developed by Shimmer Sensing.
    * Extract a bluetooth plugin for use in the Unity3d application.
- Data Storage Solution.
  - Azure.

- REST API.
  - Dot Net Core Web API.
- Website database access.
  - Login/Logout.
  - Upload training data.
- GitHub for source control and collaboration

2.4 Developing the Project

As we laid out briefly in steps in the Overview, our development approach involved meeting with our supervisor and clients regularly, showcasing progress and discussing the next stages of development. This allowed us to quickly assess the direction of the project and make changes as we developed the application. Given that the project was developed over a period of 8 months, there were many changes as we identified short comings with certain technologies, and development paths, hence the iterative nature of the development. Much of the initial development was separated into different GitHub repositories. This involved lightweight investigative builds with the various technologies we had identified as candidates for the project. We purposefully split the project into core development, which Justin initially took the lead on and investigative development, which David initially took the lead on.

Managing the Workload Units of work are divided between the developers to report back on a weekly basis to supervisor meetings. In some cases where larger volumes of work are being undertaken, the meetings may be delayed and updates reported via email.

The list in Figure 2.1, while not exhaustive, contains a broad view of how we divided up work on the project. Justin assumed the role of lead architect as the more competent programmer, and took on the more challenging development with regards to the Plugins, and designing the core of the Unity application, supported by David with regards to how the Shimmer units would be calibrated and how best the data should be presented. David spent time researching the identified technologies and prototyping early ideas, supported by Justin with regards to best practices for design and implementation.

Justin Roles	**David Roles**
Bluetooth Plugin for Android	Firebase / AWS
Bluetooth Plugin for Windows	Device Calibration
REST API	Xamarin / React / Vue / Unity Form
Blazor - Entity and Identity frameworks	Blender modeling
SQL Server backend	Unity UI
Unity Core	UI Design and iteration

Figure 2.1: Development Roles

Example week of Development

- David builds a minimal prototype application in Unity 3D which allows inputting of athlete data through a form, builds the project for windows and android.
- While Justin interrogates the Shimmer for Android SDK to begin building the Bluetooth plugin that would eventually be used in Unity.
- Both developers attend a meeting with Damien to review the progress and discuss the next phase of development.
- After weighing pros and cons of the Unity application for data input, it is discontinued in favour of a more robust framework such as Xamarin, React, or other alternatives.

- Challenges or breakthroughs in the Bluetooth plugin development are discussed. For example, separate plugins will be required for Windows and Android.

- Development goals are established for the following week and a meeting time arranged.

This format continues for the initial two months of the project. Once the project starts taking shape and we conclude that certain technologies fall short of expectations, we begin to define tasks that need to happen in a linear fashion. In cases where we encounter unexpected challenges or crossroads, where a chosen technology comes into question, we would have developer meetings which would involve brainstorming pros and cons of the given technology or a possible solution. One example of this was our decision to migrate away from Firebase which is elaborated upon further in the Technology Review Section. Development meetings often involved paired coding sessions and technical conversations about how best to implement a modular solution.

Supervisor Meetings We engaged in weekly meetings with Damien Costello to keep him up to date on our progress and development plans. These meetings proved critical to the early development of the project as we developed the requirements of the overall project. Furthermore, they were an avenue to showcase early application ideas and discuss the direction of development before time consuming work was undertaken without proper planning. These meetings gave us the opportunity to clarify our implementation from a development and implementation perspective, and to get feedback on our proposed direction. An example of this forward planning was agreeing to limit development to Windows and Android early in development. Neither of the developers had access to iOS or Mac devices for development or testing. Although we would be developing the applications in cross-platform environments for the purpose of future proofing the software, and keeping it open to further development if the project scope allowed it.

Damien provided consistent feedback on our implementation and questioned our reasoning and decision making so as to ensure we had thought through our choices. This had the net effect of opening up interesting discussions. It was encouraging that we could test and discover good candidate technologies, and review the results before delving too far into development. We would have strong reasoning for choosing one particular technology over another when we came to making final decisions and pushing forward with development. These meetings also provided us with a guideline on what stage of development we should be expected to be at for the overall project in terms of time availability and project completion.

As the project developed and we began work on more time consuming aspects of the implementation, some meetings were postponed in lieu of completing certain deadlines we had set ourselves. In the case of any postponed meetings, we emailed reports of progress, reasons for postponing the meeting and arranged follow up meetings with involved parties to showcase newly developed features of the application. Intermittent to the weekly meetings with Damien, we meet with our clients.

Client Meetings The client-developer meetings are very important for the project as we establish boundaries in what is achievable and what falls out of scope. We met with our clients on a bi-monthly basis. Initial client meetings gave us opportunities to discuss our project, showcase a prototype we had in early development at the time, and discuss collaboration on the overall project with Emma and Orna. We set out expectations with regards to development time and what we intended to deliver through the course of the project.

Each meeting offered very productive discussions on the goals of the project. The benefit of the clients perspective and expectations informed the developers goals. After the initial requirements were set the meetings took the form of clarifying points on what form the clients would like the data to be in. Emma and Orna provided us with valuable insight into the application's intended usage, and the training environment in which they intended to test it. This enabled us to direct our development towards certain deadlines early on, such as building an android prototype for testing, and data gathering functionality to store the test data for analysis.

The clients brought to light that we would need to include the option to upload a CSV file of an athlete profile, as many coaches and team managers use Microsoft Excel to keep their records. Other key features of this nature were discovered during these meetings where the developers would learn how the end user intended to use the application, and in what environment, operating system and across which levels of network access such as mobile data, WiFi the application would be used. Overall refining the requirements of the end product.

This collaboration with Emma and Orna unfortunately came to a close as they found placements in line with their preferred research areas, as such their involvement in devising test plans for testing our project implementation also came to a close. This was a loss to the overall project, however Ed frequently made himself available to us for consulting on our progress. He was interested in where the project was at, and what form it was taking. Ed continued to give us valuable feedback on the appearance of the android app and it's functionality. He would later assist us in testing the application in his training environment in Lough George Training Centre.

Extracurricular Meetings Further to our supervisor and client meetings, we also met with professor *Chris Duke* from NUIG who is involved in a project which also uses sensors to gather data, in a similar fashion to our project. It was very insightful for us to hear about a different approach to achieving some similar results and also solving some similar issues we faced with our project. We signed an NDA during this meeting and are unable to go into detail about the contents we discussed.

GitHub Collaboration We used GitHub throughout the development process, assigning issues to each other where necessary. The issue tracked feature was used extensively during the UI development as we sent ideas back and forth, identified issues with behaviour and iterated over the UI appearance and user experience. GitHub allowed us to explore different technologies separate from the core application as well, which was crucial in terms of maintenance. We used GitHub to separate our concerns early in development and only include those aspects we were happy with. For example, while Justin would focus on building out some core features of the application such as the Bluetooth plugin for Unity, David would explore Firebase as a cloud based database option for athlete profile storage.
Inevitably some of these applications did not make the final cut and in some cases were tested and discarded without adding to GitHub as there would be no collaboration, and the prototype was not fit for purpose.

2.5 Testing the Implementation

Throughout the development process, visible errors will be fixed as they arise. This applied to each category as we developed it. Development was broken loosely into categories such as core application, data storage, UI development, API development. We used the GitHub issue tracker and instant messaging in lieu of being unable to meet each other. Any small bugs were noted and given time or recorded while testing for overall functionality. Behavioral errors were tweaked until they met our expectations. Larger unforeseen errors for which we had no immediate solutions were documented for future solving. In some cases alternative technologies were used as workarounds until better solutions became apparent.

Further to the former testing, both developers would use the application and attempt to do things that should not be possible. The expected allowable actions are outlined in the System Evaluation section, this was performed on an adhoc basis as the project took shape and we had functionality available to be tested such as uploading data to athlete profiles. Overall, the application(s) would be deemed good enough to deliver after it is noted that the functional and behavioral aspects appear consistently appropriate. This applied also to the UI (Panels, text boxes, buttons etc.) were performing the expected actions. At no point were there any major application breaking bugs. Much of the testing was iterative, as sections of development were completed, manual testing would take place to confirm expected behaviour. This overall process was repeated throughout

the development cycle.

A test plan was developed for the UI of the application to assist with the development. This served to guide the developers in working towards the same goals for the UI behaviour. This was developed in line with behaviour driven development guidelines, something which the developers were unfamiliar with and had no formal education in at the time of development. Further documentation and reasoning on this testing is available in the section on **System Evaluation**.
Additionally, we has the opportunity to showcase the application to a group of second year sports students in the closing months of development. Acquiring feedback from the likely end users of the application proved to be a good indicator of success. While the application was not yet complete, there was an overall agreement that it was fit for purpose from the students.

Chapter 3

Technology Review

Throughout the development of the project continuous research was conducted as we, the developers, realised the potential to expand or further develop certain areas of the application, its design and implementation, in order to meet the end goal laid out in the Introduction.
This section covers the research element wherein we considered alternative methods to gathering data from an athlete, followed by the various technologies used during development and implementation. Some of the technologies we initially considered did not make the final cut and will be elaborated upon where relevant, in this Technical Review.

3.1 A Comparison of Wearables and Video Analysis

This section will highlight the fact that there are many options for wearable sensors, as well as motion capture alternatives, such as video analysis, both of which are broadly used to good effect in sports, including rugby. Video analysis is the gold standard in athlete analysis [1] and acts as the benchmark to which wearables are compared in order to determine accuracy and effectiveness. Furthermore, we will attempt to identify limitations of both technologies and where they are most effective.

Video Analysis The article 'Exploring the role of wearable technology in sport kinematics and kinetics: A systematic review' [2], investigates a range of the available motion tracking and wearable sensor options today, identifying that with regard to video analysis, "Motion capture systems have the ability to analyse the biomechanics of many functional and sporting tasks" [2]. In comparison to video analysis and its limitations, Yewande Adesida et al also suggest that "Wearable technology, however, is an alternative approach that has the potential to overcome these limitations." [2].

A clear advantage of using video analysis in sports is having the ability to infinitely pause, replay and focus in on points where athletes can improve their performance. When an athlete can see where they need to improve, they are more likely to correct their technique. This applies to practice and training sessions alike, giving the athlete an improved understanding of their performance in training and during professional engagements. However, Yewande Adesida et al also identify that there are limitations to how effective the motion capture systems can be, due to the need for multiple camera set up. Also, placing markers on athletes or the play area, and in some cases, manually placing markers on the athletes during video playback of recorded footage, which can be time consuming. This sentiment is also reflected upon by Daniel Kelly et al, "to evaluate measurements specific to player tackles, a time-consuming manual analysis of player sensor data and video footage is required" [3]. Video analysis shows it's strengths in post performance analysis, giving a visual reference to the athlete in identifying areas for improvement. However, due to the aforementioned limitations, video analysis is not ideal for real time tracking.

Wearable Technology When it comes to real time data gathering, wearable sensors are leading the way forwards. This is achieved by gathering markers of training intensity such as heart rate,

breathing rate, acceleration, core temperature and GPS position. These metrics lend themselves to generating objective data on athlete performance and stress on the body, enabling real time monitoring of the athlete's workload during training or professional performance. The data can be used to build a profile focused on an individual or expanded to include a team, enabling players or athletes to analyse their performance and adjust their training accordingly to accommodate areas for improvement. Using this profile, coaches and trainers alike can optimize the training regime for individuals or teams, track responses to changes in training patterns and intensity, and get a big picture of athlete and team progression over time.

As with video analysis there are drawbacks to using wearable technology. There is a general dependence on WiFi or Bluetooth to stream the data to the recording and analytical software, which presents challenges: "using a wireless method to transfer data has the potential for loss of signal during recording time or interference from mobile phones or other devices that may be on the same transmission frequency" [2]. Additionally, the safety of the athlete must be considered with regard to impacts received during a performance. To elaborate, if the wearable is impacted will the athlete themselves, or another athlete be endangered by the device should it break? Tyler Ray et al find and investigate the alternatives to hard bodied wearables in their article: 'Soft, skin-interfaced wearable systems for sports science and analytics' [4], wherein they acknowledge that hard-bodied inorganic wearables "lack the requisite soft, stretchable physical properties for establishing an intimate, non-irritating interface with biological tissues, including the skin, necessary for long-duration monitoring of athletes." [4]. Furthermore, this study explores the integration of skin-interfaced wearable systems, stretchable sensors, suggesting gains in body coverage and athlete comfort during training.

3.2 Injury Prevention: Recorded data and Athlete profiles

Firstly, it must be agreed that injury can be mitigated or prevented with the use of wearable technology. Chalmers outlines and defines injury in the context of sports in their 2002 article [5], which concludes that "Injury is just a part of the game". There has been a shift in this attitude in more recent times with the integration of video analysis and wearable sensors to gather data as previously discussed. The competitive nature of sports has lead to athlete's and their trainers developing appropriate training plans to build resilience and endurance, and mitigate the occurrence of injury. This strategy allows athletes to train longer and more effectively without suffering down-time due to injury, while gaining a competitive edge in the process.

"Tackling has been shown to be the most common cause of injury in rugby union" [3]. Daniel Kelly et al in 2012 provide a comprehensive study on collision detection using sensor based technology, identifying the need for accurate detection. Their literature describes the possibility of high accuracy collision detection using accelerometers placed directly on the athlete. By using GPS, heart rate monitoring and accelerometer data they can generate granular data as it pertains to stress on the athlete's cardiovascular system and physical loads experienced during the collision. [3].

Using video analysis in injury prevention gives coaches and athletes the opportunity to see and correct injury prone behaviours. The ability to clearly illustrate events where an athlete's technique needs attention is very useful in preventing injury, identifying a strategy and techniques to fix bad habits, and improve overall fitness, endurance and performance. However, an issue with post performance analysis to determine or assist in preventing injury is that it may be too late for the prevention of some injuries.

Wearable technology advances injury prevention with its ability to collate and calculate real time data relevant to the individual athlete. With regard to data recording, it is possible to monitor the athletes heart rate, stress load on the body, rotation and movement using accelerometers and gyroscopes, all in real time. Additionally, Tyler Ray et al describe the use of wearable biosensors, "such devices offer significant improvements in both measurement accuracy and multi-functional analysis compared with existing commercial platforms" [4], and suggest that these improvements have no negative impact on athlete performance or endurance during training. In fact, biosensor technology has been designed to minimize invasiveness to the athlete, thereby reducing discomfort.

However, there are still challenges in the application of biosensor technology, challenges which the wearable sensor devices also face, the challenge of on-board power management. The devices require a means to maintain a connection to WiFi/Bluetooth during use, batteries mean added bulk to the device and in turn negatively impacts on the seamless integration of the device and the athlete's body. The soft bodied wearables lead the charge in this regard.

The ability to gather this data during a real world training session enhance the possibilities of discovering early indicators of injury, muscular imbalance or dysfunctional movement, athlete dehydration, calories burned, all of which can be addressed proactively with training, rest or therapy. More specifically for rugby, real time data regarding collisions between athletes provides metrics to inform critical decisions for player safety, such as removing a player from a match after a collision which had the potential to cause concussion. The wearable technology plays an important role in detecting these events, allowing informed decisions to be made and preventing further serious injury, which is a core concern for the sport. Head injuries and concussion can potentially lead to more serious brain injury if left undetected. As Gardener et al point out in their article from 2015, "Future research focused on studying the acute consequences and best management strategies in current players, and the potential longer term outcomes of concussion in retired players, is needed." [6]. The data gathered in these instances is used to build a better understanding of how the brain and body react and recover after impact. Coaches and medical staff in turn can devise recovery strategies to ensure the safe recovery of these athletes. Furthermore, Coaches and training staff can devise post injury recovery strategies tailored to specific athletes, informed by that athletes profile. In a team training scenario, comparisons can be generated between athletes of similar ages, weights, positions played during matches, reactions to changes training intensity or diet as a means to gain a deeper understanding of player to team cohesion.

3.3 Shimmer Sensing - Shimmer3 IMU.

The Shimmer3 Wireless Sensor Unit used throughout the development of the project contains two accelerometers, one gyroscope and one magnetometer. The two accelerometers provide the option for wide range or low noise measurements, which allowed the developers to tailor the configuration to the needs of the application. Given the inclusion of the three sensors, the developers also had the ability to measure data with 9 degrees of freedom, henceforth referred to as 9DOF, if necessary. The developers consulted a number of technical manuals while working with the IMU devices, for the purposes of calibration [7], implementing solutions for the device with Windows [8] and Android [9], and learning about the range of software applications which were included with the Shimmer3 SDK [10] for general usage, maintenance and software development.

Figure 3.1: Shimmer3 Wireless Sensor Unit

3.3.1 Hardware Overview

Low Noise Accelerometer The output of the low noise accelerometer device on the Shimmer3is analog. A KXRB5-2042 device from Kionix is used [11]. The following approximate values apply to this device:

- Zero-output: 1.5 V.
- Full scale range: ±2.0 g.
- Sensitivity: 600 mV/g.

Wide Range Accelerometer The output of the wide range accelerometer device on the Shimmer3 is digital. An LSM303DLHC device from STMicro is used [11]. The following approximate values apply to this device:

- Full scale range:
  - ±2.0 g;
  - ±4.0 g;
  - ±8.0 g;
  - ±16.0 g.
- Sensitivity (LSB/g):
  - 1000 (±2.0 g);
  - 500 (±4.0 g);
  - 250 (±8.0 g);
  - 83.3 (±16.0 g).
- Output: 16 bit output3.

Gyroscope The output of the gyroscope device on the Shimmer3 is digital. The gyroscope on the MPU-9150 chip from Invensense is used [11]. The following approximate values apply to these devices:

- Full scale range (deg/sec):
  - ±250;
  - ±500;
  - ±1000;
  - ±2000.
- Sensitivity (LSB/(deg/sec)):
  - 131 (±250);
  - 65.5 (±500);
  - 32.8 (±1000);
  - 16.4 (±2000).
- Output: 16 bits.

Magnetometer The output of the magnetometer device on the Shimmer3 is digital. An LSM303DLHC device from STMicroelectronics is used [11]. The following approximate values apply to this device:

- Full scale range (Ga):
  - ±1.3;
  - ±1.9;
  - ±2.5;
  - ±4.0;
  - ±4.7;
  - ±5.6;
  - ±8.1.
- Sensitivity (X,Y/Z) (LSB/Ga):
  - 1100/980 (±1.3);
  - 855/760 (±1.9);
  - 670/600(±2.5);
  - 450/400 (±4.0);

- 400/355(±4.7);
- 330/295 (±5.6);
- 230/205(±8.1).

Bluetooth

- Range: 10 metres Class 2 Bluetooth Radio.
- Rovingg Networks RN - 42.
- Soft-power control.

Shimmer LED Indicators Two software-controlled LED indicators (visible in Figure 3-1) are available, the lower indicator is intended to display operational status and is tri-coloured(green, yellow, and red). The upper indicator is bi-coloured(blue/green) and is intended to display the data communication mode or status [12].

- **Solid Green** - Shimmer is powered on.
- **Solid Blue** - Bluetooth connected.
- **Flashing Blue** - Shimmer is streaming only.
- **Flashing Green/Blue** - Shimmer is streaming and logging.

3.4 Blender

Blender was chosen as the modeling environment to build a simple representation of the Shimmer3 IMU for use within the Unity 3D android and windows applications. Blender was used in the initial research and investigation project, to model a person model. The software has a vast toolset enabling fast prototyping of 3D models which was desirable to the developers as 3D modeling was not at the core of the project, and having a 3D replica of the Shimmer unit should not consume a lot of development time.

Alternatives to Blender do exist. We looked at 3DSMax and Maya as potential modeling tools. However we would have a learning curve with both alternatives, and we had a small familiarity with Blender having used it to create a human rig for the initial investigative and research project from the previous year. It was an easy decision to make. Moreover, while we may not produce a commercially viable product, we would be protected if we did having used Blender instead of either 3DS Max or Maya, as both of these alternatives require payment for any use in a commercial product and are only freely available as a limited version of the full tool or with a student approved license.

Advantages of Blender:

- Free Software without limitations.
- Open Source Software - Built by the community with a broad set of freely available features and libraries that can easily be downloaded and used.
- Runs on any OS, we could easily share models and files without worrying about conflicts or corruption.
- Abundant documentation and guide material available online.
- Abundant tutorials available online from a diverse range of skill levels. We would be able to find resources to assist in building a model effectively and quickly.
- Experience with Blender from previous prototype project.

Disadvantages of Blender:

- The main disadvantage of Blender is generally seen as its steep learning curve and reliance on a large amount of hotkey combinations. Given that we were already somewhat proficient in using Blender, this did not apply to us.

3.5 Unity 3D

3.5.1 3D Development Environment and Engine

Unity3D was the chosen development environment for the android application for a range of reasons. Unity has an integrated physics engine which the developers would leverage when manipulating the 9DOF provided by the sensor unit. Converting from Euler Angles to Quaternions was central to manipulating the data returned from the IMU, in order to display accurate rotations in the application, and avoid gimbal lock. The unity engine physics engine proved to be of significant value during development.

Unity uses a left-handed coordinate system (fig 3.2), which would be a valuable piece of knowledge for us to have during development. Clearly, when we are retrieving data from the shimmer unit and calculating velocities and angular rotation to apply to the model, we will need to be certain that the axis's are aligned with each other from our model to the IMU. More on this in the section on calibrating the Shimmer3 IMU.

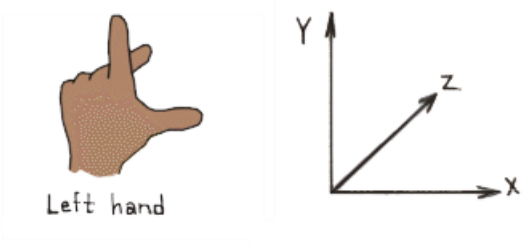


Figure 3.2: Unity3D Co-ordinate System

Unity provides an environment to develop cross platform applications seamlessly, for a wide range of platforms. The cross platform aspect of the project was one which was identified during the initial research project mentioned in the Introduction. While other platforms such as Microsoft's Hololens, and iOS were alluded to in the future development opportunities from the initial research project, the developers made the decision to develop for Windows and Android due to limitations in the availability of iOS and Mac hardware. Similarly, the Hololens concept was dropped due to how restrictive the end product would likely be for future development, and for potential end users for testing and wide scale deployment.

We considered some alternatives to Unity3D, such as developing the project as a mixture of a UWP application and Xamarin forms. Those options had some advantages in terms or providing nice libraries for some of the back end, plugins for managing JSON with the Newtonsoft package manager, CSharp which we are familiar with. We had also some experience working in these environments from course work and modules through 2nd and 3rd year.

There are other games engines we also could have used such as Godot or Unreal. The Unity Game engine was familiar to us from using it during a course module in our second year, as well as having used it for the initial research and investigation prototype project in 3rd year. There is a learning curve to any new technology and part of this project for us was delivering some usable application. We had set some goals during the initial meetings with the clients and supervisor, as such we would be playing to our strengths with our choice technology.

UI design, animating and manipulating our shimmer model, and indeed using a model built in Blender as opposed to drawing it in UWP for example, were important factors to us in choosing Unity3D.

Some of the key driving factors to our decision to work with Unity3D:

- Unity 3D is Free for developing non commercial products, games, applications.
- Cross-platform development.

- Supportive community with tonnes of documentation.
- Abundance of free tutorials available online to support the documentation.
- Asset store with useful functional packages to cut down on development time.
- Optional use of Javascript for our REST integration.
- Integrated Physics Engine.
- Optional 2D development which we used for our UI.
- Unity Remote 5 Android application for run-time testing of application.
- Familiarity with the Engine from previous project work.
- Build experience using Unity3D for the prototype project in 3rd year.

3.5.2 UI

UI development was of key interest to the developers, who understand that the application not only depends upon well implemented code but on having an intuitive and responsive User Interface in order to provide a user-friendly experience. The human-computer interaction is based on the user providing some input to a system and the system providing some representative feedback to the user so that they understand the action has been acknowledged, and is being interpreted. As Andrew Dillon alludes to in his paper 'User Interface Design', "users engage in an ongoing cycle of information exchange involving exploration of a changing information environment" [13]

Unity provides extensive UI development tools and animations which we intended to leverage to provide tangible user feedback. During the initial investigation of technology we identified that Unity has a canvas. The Unity canvas has many features which are ideal for developing menu systems. The developers can use specific 'UI' elements to arrange neatly designed menus, adding functionality as necessary, or simply using some UI element to slide in and out of view with a notification of some activity. We used the canvas for all of the menu and notification activities within the application.

3.5.3 Wireless Device Connection

Although Unity does not support connection of external devices such as the Shimmer IMU units, since we had already conducted an exploratory project into the use of these devices within Unity, we had both experience and a small code base to begin with. The decision to also build the project for the mobile Android platform, however, increased the complexity of the overall system by the necessity of building a Java based plugin to allow the handling of Bluetooth connections on that platform. Various packages already existed within the Unity Asset store, but these were paid assets so we decided against using of them at an early stage in development.

3.5.4 Data and Storage

Based on our requirements, it would be necessary to stream data in almost realtime from the IMU devices. In addition to this streaming functionality, we would also need the data to be persisted somehow for reuse - either by this application or by some other external application, e.g. for potential analysis in Excel at some point.
This data handling and storage is two fold in the application:

- Data streaming from the IMU to the application should be managed in real time, so that we see the 3D model representing the actual shimmer unit rotating in real time in the application.
- Data is collected during application run-time and stored in memory. Once a session has been streamed, the user should have the option of saving this data either locally or to some central location. For example, a record of a training session is created and saved to an athletes profile in a database. From our research, we realised that Unity has builtin libraries to handle many types of data transfer and storage.

3.6 Shimmer 9DOF Calibration v2.10

The shimmer unit comes with a calibration stand, [10] as we can see from the user manual which shows the unit place in this stand. The stand serves to keep the unit in place during calibration. The image in fig 3.3 shows the stock calibration on the unit.

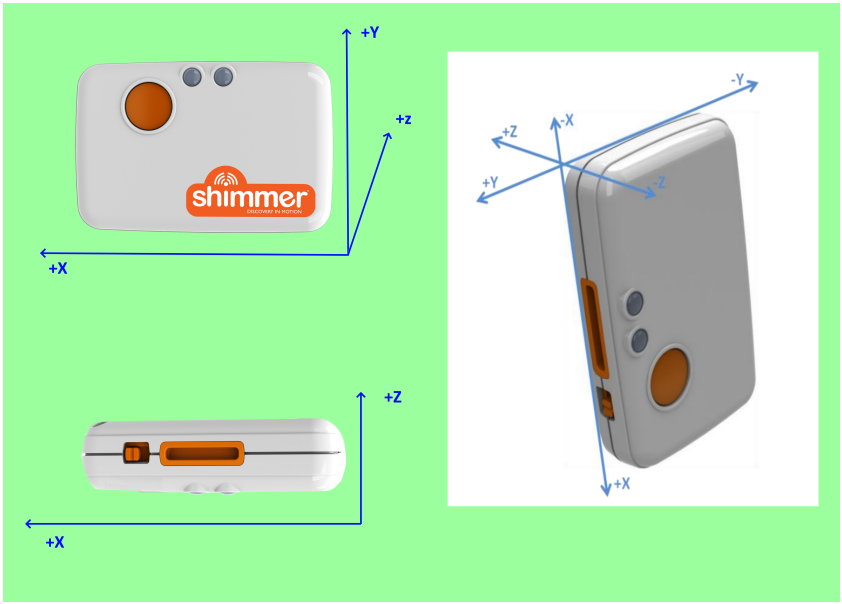


Figure 3.3: Shimmer3 Stock Calibration

We would be placing the unit on an athlete's back, between the shoulder blades, with the unit rotated such that the long face of the unit is horizontal from the ground plane while the athlete is standing upright. The authors of 'Evaluating squat performance with a single inertial measurement unit', placed the shimmer IMU at the base of the target's spine, the 5th lumbar vertebra to be precise. During their comparison where they examined eight different squat conditions, they showed that the single IMU could distinguish between varying levels of squat performance, noting that "This serves as preliminary evidence that a single IMU may be an effective method of monitoring multi-joint exercise performance in both rehabilitation and S&C (Strength and Conditioning) contexts" [14]. This research informed our decision to place the unit at the athletes neck, along with advice from Ed Daly, and current practices in rugby and Gaa.

Calibrating the units required some research into coordinate systems in both Unity which as we mentioned earlier, is a left-handed coordinate system, and on the IMU's themselves. The IMU needed to be calibrated to a left handed coordinate system through the calibration software from the SDK.

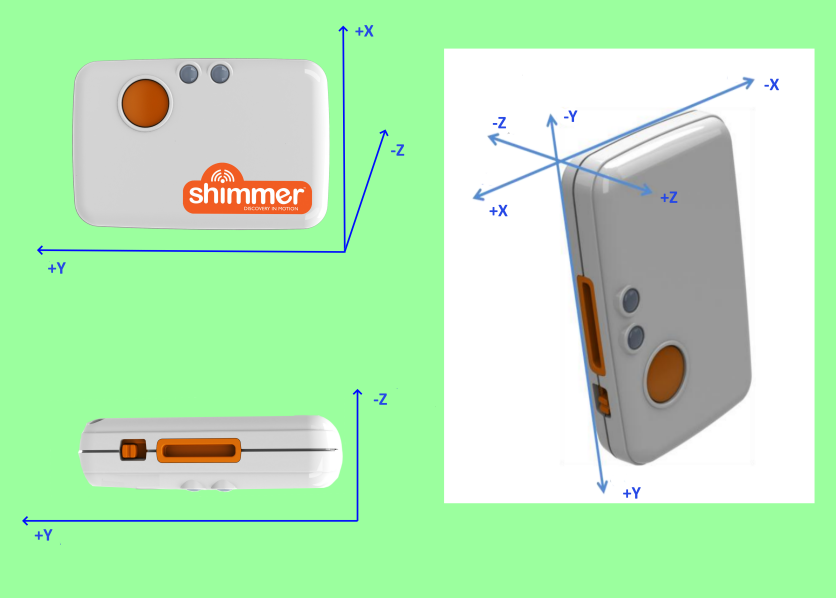


Figure 3.4: Shimmer3 New Calibration

The image in fig 3.4 shows the proposed calibration for our use in the Unity application.
The placement of the unit on the athlete would align the x-axis with the ground plane. By which we mean that with the x-axis returning a value of 0 degrees, the athlete is standing upright, perpendicular to the ground plane. With the x-axis returning a value of 180 degrees the athlete is parallel to the ground plane. During some test sessions with Ed Daly and a group of his students, we trialled wearing the sensor and determined that the calibration was fit for purpose as a means to record the angle of incline of the athletes spine.

The y axis would act as the measure of spinal rotation. A value of 0 degrees on the y-axis would infer that the athletes spine is in it's natural position. Any value of y which returns positive would mean the spine is rotating in the athletes counter-clockwise direction, while a negative y value would mean the athlete is rotating in a clockwise direction.

Wearing the unit with the elastic straps provided with the shimmer kit was infeasible as there was a high probability of the unit coming loose during a tackle. Another issue was the elastic straps would allow the unit to shake or move considerably more than would be acceptable. This would have the adverse affect of introducing noisy data to the recordings and would likely reduce the overall accuracy of the data. We were lucky however as there were alternative vests available in the training centre which were normally used with other GPS units. These vests had a secure and tightly woven slot on the back where the GPS would slot into place. The shimmer unit was a tight fit, which allowed us to vastly reduce any noisy data from being recorded.

3.7 Xamarin Application.

It was clear from the initial meetings that we would be providing the client a means to upload existing athlete profiles to some central location. We put together a prototype application with unity to test this feature and functionality. At this time we were examining our options, and considering keeping the project tightly coupled together as a single Unity application, however, we came to the conclusion that while Unity's UI elements provide a great experience for many input types, they somewhat lacked when it came to the type of data input which we required for maintaining player profiles. While Unity provides much of the functionality we would need for the shimmer unit, we felt that it did not offer enough to allow for a good experience with inputting numerous text fields, so, We compiled a list of alternatives that would provide the functionality we needed and researched each before making a decision.

For this portion of the project, we considered the following frameworks and technologies:

- React.
  - Build applications on native platforms.
  - Based on JavaScript and React.
  - Focus of React Native is on developer efficiency across all platforms.
  - Facebook uses React Native in multiple production apps and will continue investing - Future Proof.
- VueJS.
  - Vue Native - mobile framework to build truly native mobile app using Vue.js.
  - Designed to connect React Native and Vue.js.
  - Vue Native is a wrapper around React Native APIs.
  - Rich mobile User Interface.
- Xamarin.
  - Build applications on native platforms.
  - Mono-based - use existing .NET code, libraries, tools, CSharp.
  - Create mobile applications.
  - Target Android-based smartphones and tablets, iPhone, iPad and iPod Touch.

We drew up a simple comparison table to help us decide which would best fir our development needs, listing the features that we felt would benefit the build and future proof our solution.

	React	**VueJS**	**Xamarin**
			
Cross Platform	YES	YES	YES
DotNet	NO	NO	YES
Supports Windows	YES	Partial	YES
Native Performance	YES	YES	YES
Free	YES	YES	YES
Developer Familiarity	NO	NO	YES
Future Proof	YES	-	YES
Fast Build Time	YES	-	YES
CRUD Functionality	YES	YES	YES
REST	YES	YES	YES
3rd Party Dependencies	YES	YES	NO
MVVM Pattern	YES	YES	YES

Table 3.1: Comparison of Technologies.

Xamarin was the clear choice for multiple reasons. It enables the developers to leverage existing knowledge of Dot Net frameworks and libraries within those frameworks. Running C# natively on mobile devices as well as providing a robust solution for windows based environments which we would be targeting as a secondary concern, for office based users to upload bulk data to the database. Choosing Xamarin would also allow us to share some common code with our Unity application, increasing code reusability and modular design.
To that end, we agreed we would design a simple interface to allow easy inputting and updating of the player profile data, either from a CSV file or adding athletes and related teams on a case by

case basis. The CSV file uploading was requested during early meetings with the clients, as they explained many existing athlete profiles would exist, and would already have the relevant data associated to them that we had discussed, such as age, weight, height etc.

3.8 Android Development.

The end goal of this portion of the project was to create a JAR file which could be loaded by Unity and provide native Bluetooth functionality to the application when running on the Android platform. Android Studio was instrumental in developing this 'Android Bluetooth Plugin'. Central to this development was the availability of the 'ShimmerAndroidAPI' application suite. This is a collection of example Java implementations from the Shimmer SDK which contained functionality for using the Shimmer3 IMU with Android OS. Interrogating this code base was important for the developers to gain insight into the workings of the application, and subsequently extract the Bluetooth functionality away from the bulk of the program. This application contained a very large amount of complex source code and both developers contributed to interrogating this in order to identify the key components needed for the standalone Bluetooth plugin to be developed.

The development required research into working with Gradle builds, something which was mostly new territory for both developers. The plugin had to be developed in Android Studio which was also a new IDE for the developers. While it was a daunting task to learn a new framework in a short space of time, a persistent collaborative effort was undertaken to make progress on this front, as this would potentially make or break the entire project. As outlined in earlier sections regarding the methodology and the overall approach, this was one of the earlier tasks we had tackled as it was identified and labeled as critical to the project.

3.9 Data Storage

We looked at the standard concerns for choosing a cloud storage service. From the outset, our focus was on storing player profile data along with data received from the IMU. Storing the IMU data would mean that the entire data set would have the potential to grow exponentially large given the volume of data a single recording might contain. Considering this application is intended for use by groups of athletes, the storage would need to be expansive. At this point we drew up a number of broad points and questions which we deemed to be important regarding the selection of our data storage strategy:

- Types of Data - what type of data will be actually be storing? Is this data of a consistent form or is it liable to change over time? Is a relational database to restrictive for this?
- Scalability - will the storage option be able to scale both with the amount of data being stored and the amount of unique clients needing to access or modify this data?
- Security - how is access managed to the data and can this access be managed at a very granular level?
- Integrity - would we have the option to backup and restore data, either automatically or manually? If so, how much human intervention would be necessary and how frequently would these tasks need to be performed?
- Ease of development - how difficult is it to get set up with option and what exactly will need to be developed if this option is chosen?
- Cost - is this something that we could implement at a low cost or for free? If so, what are the drawbacks of choosing this option over more costly options?

All of the above points were factored in when considering our choice of storage strategy. The first point was at the fore since the type of data to be stored has a huge bearing on which type of database would be used. We looked at a selection of the most popular databases and compared under a number of headings, as can be seen in Table 3.2.

	Firebase	**SQL Server**	**MongoDB**	**neo4j**
	Firebase	SQL Server	mongoDB	neo4j
Type	Doc	RDBMS	Doc	Graph
Server	N	Y	Y/N	Y
Schema	N	Y	N	Y
Concurrency	Y	Y	Y	Y
Transactions	Y	Y	Y	Y
Authentication	Y	Y	Y	Y

Table 3.2: Brief Comparison of Data Storage Options.

After some deliberation, we settled on trying out Firebase. Core to our choice was the fact that Firebase does not require a server so costs are low and as a result, overall complexity is reduced. When we also considered the fact that Firebase storage was implemented in both the Unity and Xamarin applications for player profile management with great results. However, when we reached the stage where we began implementing the storage of the IMU data in Firebase, issues with this strategy became apparent and we began to rethink our decision.

We came to the realisation that using a document-oriented storage solution was not very suited to the type of IMU data which we were recording in bulk. Since Firebase stores all of its data in documents, this resulted in very long query times - when only a subset of the data was required, all of the data is queried and then filtered. At this point, we had considered taking a hybrid approach - storing the profile and related data in Firebase, while opting for a Relational Database Management System for storing the IMU data since the form of that data is, in the scope of this project, never liable to change.

At this point, We came to the conclusion that it may be better in the long run to create a more robust backend than Firebase to store all of the application data and we shifted towards a REST API server based application backed by a relational database. This would also mean that any application could then use HTTP requests to access the data rather than using specific libraries for, e.g. Firebase or Neo4J, etc. libraries for each framework: Xamarin, Unity and whatever other application may eventually require the data. Again, we looked at many of the options options available for such a solution: ExpressJS, Python, Java Spring, DotNet. What stood out for us was a DotNet based option - since our current choice of language for each application to date was based on C#, it made a lot of sense to us to once again develop with the C# language and Visual Studio IDE.

After doing some further research into the potential for developing a REST API in C#, we discovered that the next generation Blazor framework from Microsoft was recently released. Blazor is based on Dot Net Core 3.1 and has the ability to leverage Web Assembly technology to run C# code directly in the browser. Although we would not actually need a web-based frontend for this project, we decided to undertake and learn this new framework. It should be noted that the REST API portion of this framework is firmly rooted in the tried and trusted ASP NET Web API framework and as such is already proven to be a robust and resilient technology. We laid out a new project plan for this portion of the project which detailed the following components:

Server Application A Blazor client/server application (using .NET core 3.1) to act as a frontend and allow REST API access to a SQL Server backend instance. We would use Entity and Identity frameworks in conjunction with SQL Server to facilitate fast prototyping and development of both the database and REST API. SwaggerHub would be used to design the REST API endpoint scheme and their functionality. Using the Identity framework, JSON Web Token technology would

be enabled and configured meaning that our security concerns would be addressed by having authorization enabled on any or all API endpoints which we deemed necessary. Since JWT is ubiquitous at this point, it is also well supported by both Unity and Xamarin.

- As outlined above, a frontend web client is contained within the server application and uses C# in the browser via Web Assembly along with HTTP requests to the API. For the scope of this project, we would disable this web client when deploying the application to cloud hosting. However, this client would be very useful for testing the API functionality and, in future it could allow for a rich web experience perhaps for reviewing and charting uploading IMU data.

Database A SQL Server instance whose access is restricted to the REST API application, in addition to developer access when needed. Initially we began by using a SQL Server in a Docker image and this worked very well for our exploratory and prototyping development work. Once the application and database were in a usable and secure state, we would then migrate both the database and the server application to Azure Web Services where the REST API endpoints would then be accessible across the internet to both of our client applications. While a frontend is included as a part of the server application, we would use this only for debugging and development purposes, e.g. testing the REST API endpoints, and this would then be disabled once the application was deployed to cloud hosting.

Docker While developing both the database and server applications together, Docker would be used to host the SQL Server instance. The server application would also be packaged as a docker image and deployments would be tested to both the Azure, AWS and Google Cloud Platform web services. Since Docker images are designed to be portable and platform agnostic, this approach would leave our options open down the line should we run into any issues using either of the aforementioned cloud platforms. All of these platforms allow for simple deployment of Docker images to their respective 'Compute' services but we felt the actual chosen platform open to be decided later would be the most prudent approach.

3.10 Docker.

Docker sounded like an ideal candidate for the development workflow. Docker would allow us to:

- Create reproducible environments for our application.
- Automate the setup of the environment.
- Eliminate some manual set-up on our different systems.
- Reduce the overall risks of introducing independent and unknown variables to the workflow.
- Increase the reliability of the deployment process.

There are also challenges to using a new and unfamiliar technology, and possible environments where using Docker can be difficult. This section discusses several advantages and drawbacks as we encountered them, with the use of Docker.
We developed a workflow using Docker to containerise our web app.
Pros:

- Highly portable.
- Widely used in industry.
  - Good experience to familiarise ourselves with Docker.
  - Working with images and creating containers.
  - Deploying containers and images to cloud platforms.

- Full control over the execution environment of the application.
- Reduced Risk.
- Testing across multiple environments

Cons:

- Steep learning curve, e.g. creating images and containers, deploying to GCP.
- Slow development cycle and deployment.
- Increased complexity in the project.
- Windows and Docker not the friendliest of friends.
  - Initial running issues with Docker on Windows 8.1.
  - Follow up issues running Docker on Windows 10 home with VirtualBox, clunky and sluggish.
  - Cross platform containers can run on any operating system running Docker.

As we have outlined in our review of the technologies we intend to use, and their limitations, we have outlined some of the alternatives to gathering data with wearable IMU's such as the Shimmer3, and their limitations.
Establishing some points of reference was an important step in the development of this project as the developers would gain valuable insight into alternative approaches to solving some similar technical problems. This research and review section serves as a means of elaboration on how we learned through examining various other technologies, which ones would best suit the needs and requirements of our application as we identified them in the methodology.

With this in mind it's our intention to use the former technologies and Shimmer unit to attempt to gather accurate data on an athletes tackling performance and attempt to generate some usable data to distinguish between what would be deemed as a safe tackle and a risky tackle.
It is our intention to provide researchers in this arena of sports development and research, with a tool which enables the gathering of meaningful data. That this data can be used to form and backup reasonable arguments and assertions with accurate data which is both robust and reliable.

Additionally to any research opportunities, this application may also be used in a training scenario, enabling coaches and athletes alike to view in real time, the rotation and incline angle of an athletes spine during a tackle. Providing an added layer analysis, to reinforce the visible aspect of watching the tackle allows for further insight into identifying areas of improvement for athletes and has the potential to change coaching style.

Over time the athlete would build up a profile and have the data available to review. Viewing tackling data in this way serves to keep the athlete and coach informed and aware of improvements as the athlete trains, and also potentially avoid incurring injury by correcting risky tackling posture in a safe and controlled training environment thereby mitigating incidents of injury during a live activity such as competitive matches.

Chapter 4

System Design

4.1 Overview

The system is composed of three core applications and a database server: a REST API server application backed by the database instance, a cross-platform client application created in Unity3D and a cross-platform application created with the Xamarin framework. The server application has been deployed to Azure Web Services and the client applications may be deployed on any Android (>= API level 20 aka Lollipop[15]) or Windows 10 device. An overview of the system can be seen in Figure 4.1. We have discussed the reasoning behind many of the choices behind the selection of various technologies in the Technology Review Section 3 of this document.

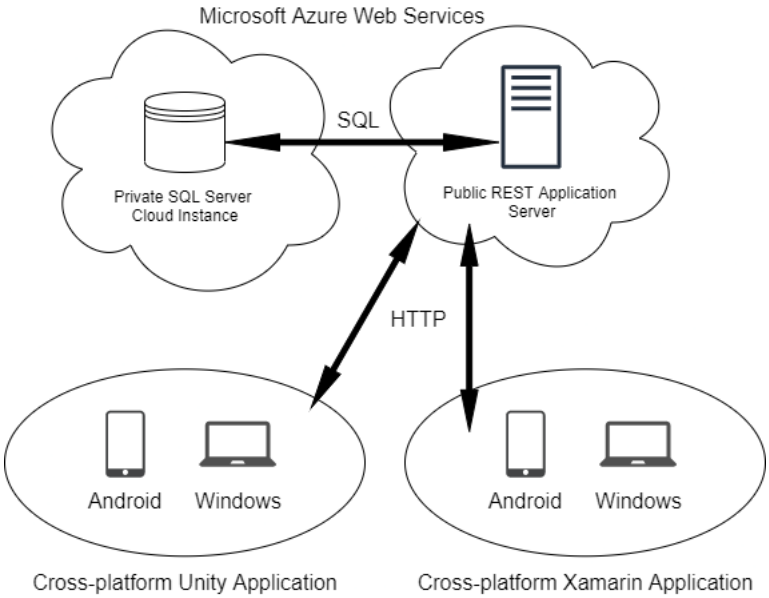


Figure 4.1: System Design Overview

4.1.1 Shared Code

In the interest of code reuse and DRY principles, and since each of the applications are written in C#, the core data models used within all of these applications are provided through a shared Dynamic Link Library (DLL). In this way, each application uses the same Plain Old CLR Object (POCO) classes to describe the data entities within. Taking this approach meant greatly reducing code repetition in each application by creating a common data model layer.

This shared code library contains three classes, which are all declared with the C# Serializable annotation. The Serializable annotation denotes this class as being a POCO and thus is easily serialised and deserialised to and from JavaScript Object Notation (JSON), respectively. Since all data transfer to and from the REST API is in JSON format, this allows for very easy management

and centralisation of our core data models. Any change made to one of these models must be reflected in the handling of models in each of the applications, or, in other words, if the data specification of the data used by the applications changes, each application should be updated accordingly to match the data models.

These data models will be referred to frequently throughout the rest of this chapter.

4.1.2 Data Models

BaseDbModel and BaseTrackedDbModel

Each of the aforementioned data models inherit from a common, abstract, base class - BaseDbModel seen in Listing 4.1. This class defines a sole attribute, Id, which will be the Primary Key on each model in the database and so should be represented in code. The data type of this attribute is declared as a nullable integer which is necessary when creating new models which do not yet have a primary key value assigned to them.

```
\begin{lstlisting}
[Serializable]
public abstract class BaseDbModel
{
    [Key]
    public int? Id { get; set; }
}
```

Listing 4.1: Abstract BaseDbModel POCO class

Additionally, both the Player and TrainingRecord models inherit from a further base class, BaseTrackedDbModel, which defines a number of attributes for tracking changes made to the underlying database models. This class can be seen in Listing 4.2.

```
public abstract class BaseTrackedDbModel : BaseDbModel
{
    public DateTime CreatedAt { get; set; }
    public DateTime ModifiedAt { get; set; }
    public string CreatedBy { get; set; }
    public string ModifiedBy { get; set; }
}
```

Listing 4.2: Abstract BaseTrackedDbModel POCO class

PlayerModel

Representation of a Player in code which includes relevant all attributes, some of which can be seen in the code snippet in Listing 4.3. The one-to-may model relationship with TrainingRecord is also defined here. Note that since the base class has already been annotated as Serializable, there is no need to add this annotation to inheriting classes:

```
public class PlayerModel : BaseTrackedDbModel
{
    public int Id { get; set; }
    public string Email { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    //... some attributes removed for conciseness

    // Relations
    public virtual ICollection<TrainingRecord> TrainingRecords { get; set; }
}
```

Listing 4.3: PlayerModel POCO class

TrainingRecord

Representation of a single, recorded training session in code as seen in Listing 4.4. Each Player will have one or more training records related to them and each training record has a one-to-many relationship with ShimmerDataModel, as defined by the two virtual properties:

```
public class TrainingRecord : BaseTrackedDbModel
{
    public virtual PlayerModel Player { get; set; }
    public virtual ICollection<ShimmerDataModel> ShimmerData { get; set; }
}
```

Listing 4.4: TrainingRecord POCO class

ShimmerDataModel

Representation of an IMU data record. Each record contains a number of attributes which store the numeric values output from the IMU sensor. Since project work was performed solely with one type of IMU device at hand, the Shimmer3 IMU, we have named this class ShimmerDataModel, as seen in Listing 4.5, but due to the modular design of the Unity application, data from any IMU could be stored as such. Note that the ShimmerDataModel class does not inherit from the BaseTrackedDbModel class but from the BaseDbModel class in an effort to keep its serialised form to a minimum size by reducing the number of fields therein.

```
[Serializable]
public class : BaseDbModel
{
    public virtual TrainingRecord TrainingRecord { get; set; }
    // Timestamp
    public float T { get; set; }

    // Accelerometer
    public float LN_X { get; set; }
    public float LN_Y{ get; set; }
    public float LN_Z{ get; set; }

    //...some attributes removed for conciseness

    // Quaternions
    public float Q_0 { get; set; }
    public float Q_1 { get; set; }
    public float Q_2 { get; set; }
    public float Q_3 { get; set; }
}
```

Listing 4.5: ShimmerDataModel POCO class

4.2 REST API Server Application

4.2.1 Overview

This application, running on Microsoft Azure, acts as a central data storage and retrieval server. The REST API application is part of a Blazor web application which is backed by a SQL Server database instance. The frontend of this web application has been used for developmental and testing purposes but, for security purposes, it has been configured to be disabled when deployed to Azure. The application was initially created using the Blazor App template for Visual Studio. Our criteria and reasoning for selection of this particular technology has been discussed in detail in Section 3.9.

4.2.2 REST Design

SwaggerHub has been used to design the API routes, following closely along with REST principles in doing so. Figure 4.2 shows an example of the API design, specifically for the Player resource.

Figure 4.2: Players REST API design from SwaggerHub project

4.2.3 Application Design

Entity Framework

Since the REST API is not the main focus of this system, the Entity Framework[16] Object Relational Mapper (ORM) has been used to manage the database through the application of database migrations and the handling of the majority of the Data Access Layer.

Code First and Database Migrations[17]

A code first approach has been taken, meaning that the classes representing the database entities are written first and then database migrations are created which build the relations for these models in the database. Using these migrations, it is then possible to easily step through each iteration of the database design at any given point in the development cycle.

Entity Framework greatly simplifies the data access layer of the application through its advanced ORM. This is configured by registering each model which is to be stored in the database within the ApplicationDbContext class.

The following code snippet registers the PlayerModel (POCO class) with the Entity Framework ORM:

```
1     public DbSet<PlayerModel> Players { get; set; }
```

The DbSet class denotes a collection of database entities, in this case, a table of PlayerModels. After registering the model like so, a database migration can now be automatically created based on changes made to any POCO registered in the ApplicationDbContext. Using the Package Manager Console in Visual Studio, the following command must be run:

```
1    Add-Migration
```

The migration has now been added and contains automatically generated code which may be used to change the underlying database structure, relations and relationships. To apply this migration to the database, the following command must be run, again from the Package Manager Console:

```
Update-Database
```

MVC

Adhering to the the Model-View-Controller (MVC[18]) pattern, the API is implemented using Controllers and Models. The View component of the MVC pattern is used in the frontend which we have only used for developmental and testing purposes.

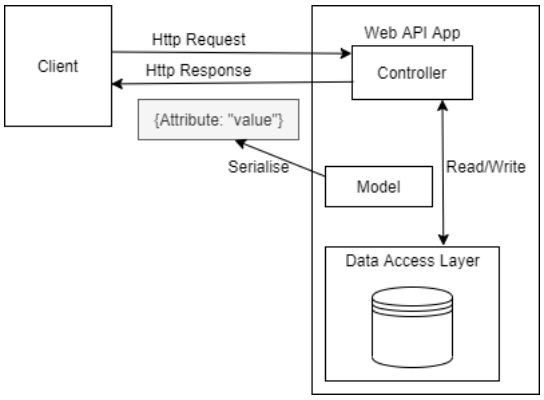


Figure 4.3: MVC Pattern

Controllers

A controller is generally only used to handle requests to one set of endpoints, although as the complexity of an application increases so too do the responsibilities of controllers. Controllers within the DotNet framework do not need to be configured or registered within the application but must adhere to a number of rules:

1. Each controller should extend the ControllerBase[19] class.
2. The controller class must be annotated with the [ApiController] annotation.
3. The controller class should also be annotated with the [Route("api/[controller]")] annotation. The [controller] part of the annotation refers to 'the name of this controller', i.e. for PlayersController the route will be 'api/players". This value can be overridden using a plain string, e.g. [Route("api/myPlayerRoute")] will result in the route "api/myPlayerRoute" being used.
4. The constructor of a controller should accept a single argument - an instance of ApplicationDbContext. In this way, the data context is passed in to the controller using Dependency Injection.
5. Each method within a controller should be annotated with the HTTP method expected by that endpoint, e.g. [HttpGet], [HttpPost], etc. If a route parameters is also to be passedm this should be included along with the HTTP method, e.g. [HttpGet("id")] would be used on players/id route where id is the primary key of the PlayerModel to fetch from the database.

Example Request Handling in MVC with Entity Framework

Using the above Player resource example, this section will run through an example GET request sent to the application's /players endpoint:

1. The incoming HTTP request is received by the application. The underlying framework inspects the requets's destination endpoint and compares this to all subtypes of ControllerBase within the application's controller folder.
2. If a controller matching the route is not found, an error is thrown and the appropriate response is sent to the requester. Otherwise, the request is sent to the appropriate controller, in our example, the PlayersController.

3. The controller receives the request. It also inspects the request and looks at the endpoint and HTTP method to determine whether it has a matching method. In this case, it finds the GetPlayers() method which can be seen in Figure 4.6.

```
1     // GET: api/Players
2     [HttpGet]
3     public async Task<ActionResult<IEnumerable<PlayerModel>>>
4     GetPlayers()
5     {
6         return await _context.Players.ToListAsync();
7     }
8
```

Listing 4.6: GetPlayers method in PlayersController.cs

4. The code within the GetPlayers() method is executed. As mentioned in the previous section, the context variable here is assigned through dependency injection into this class' constructor. The return statement finds all Player models (Players) within the database and returns them as a list of PlayerModel POCO objects.

4.2.4 Database

The REST API application is backed by a remote SQL Server instance which is also hosted on Azure Web Services. The database has been configured with security in mind and as such, access to the database is limited to [REST API] application. This isolation is achieved through the Azure security settings console which allows for instances to be connected privately to one another, as if they exists on the same local network. One exception to this security rule is that access can be temporarily granted to each of the developers when direct access is needed to the database. This can be achieved in SQL Server Management Studio by logging in to the database server via an approved Azure account.

Entity Relationship Diagram

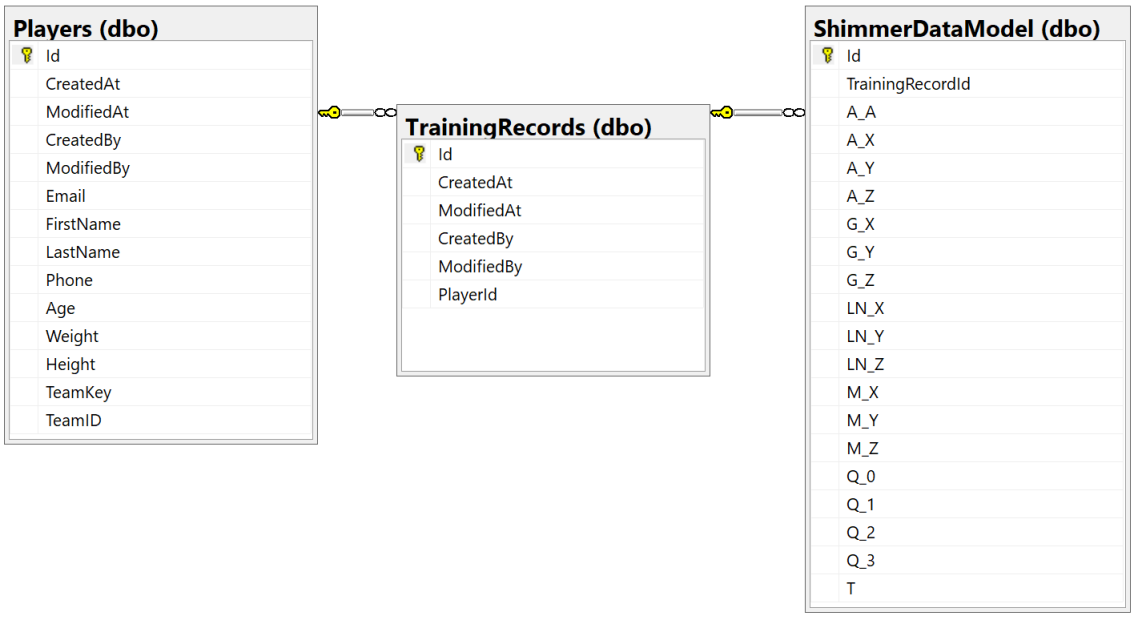


Figure 4.4: Database Design

As mentioned in the Shared Code section, the applications share a common code library containing the data models used in each application. Each of these models are persisted in the database

through the use of Entity Framework and are only accessible to the applications through the REST API interface.

The three core entities which are persisted within the database, as shown in Figure 4.4 are:

1. Player - the represenatation of a PlayerModel within the database. This entity has a one-to-many relationship with the TrainingRecordModel, meaning that each player can have zero or more training records associated with them.
2. Training Record - the represenatation of a TrainingRecordModel within the database. This entity has a one to many relationship with ShimmerDataModel.
3. ShimmerDataModel - the representation of an IMU data point within the database, this entity contains fields to record data such as rotations, linear accelerations, angular velocity, etc from the IMU device. These data points are recorded as a group and belong to a Training Record entity.

4.3 Unity 3D

4.3.1 Overview

The main focus of the project, a cross-platform application developed in Unity3D which has been designed to allow for easy connection to a Shimmer IMU device through Bluetooth and use the data received from the IMU to manipulate a 3D object and present the user with useful feedback. The intent is to allow for an almost real time view and output of the orientation of the IMU. Additionally, impacts from the device can be sensed and displayed on-screen. The application is intended to be as user-friendly as possible and the user interface (UI) design has been developed based on feedback from potential end-users, users that are currently using associated data sensing, collection and reviewing or reporting software.

4.3.2 Bluetooth Plugins

As the application is intended for cross-platform use, each platform required its own implementation of the Bluetooth device management system. Unfortunately, this functionality is not included in any standard Unity libraries, although paid assets do exist in the Unity Asset Store. It was necessary for us to two create two custom plugins for Unity3D - one for Android and one for Windows.

The Windows plugin is provided as a compiled Dynamic Link Library (DLL) and the Java plugin is a compiled JAR file. While both plugins achieve the same goal for each platform, there is a key difference between the plugins and this is as a result of how the Java Native Interface[20] (JNI) is used to communicate with the Java plugin's code.

Since the windows plugin is written in C#, it can pass POCOs directly to Unity code, however, the JNI is only capable of passing string data through its interface and so the data from the Java plugin is serialised to JSON and then deserialised into objects when received by Unity. One may be forgiven for assuming that this is a slow process but our testing has proven that due to the rate at which data packets are generated and transmitted by the IMU, the latency introduced by the serialisation and deserialisation process has no effect on either the frequency of objects recieved in Unity or the overall responsiveness of the application to changes in the IMU orientation or acceleration.

Windows Bluetooth Plugin

Utilising the Shimmer C# SDK[21], this plugin has been developed over time as a separate Visual Studio Class Library project which has then been compiled to a DLL and included in the Unity project. Unity allows external DLLs to be loaded and used within its projects, once a certain folder structure convention is adhered to, specifically that any plugin code should be placed under the Assets/Plugins/ folder of the project. We have placed our Windows plugin in Assets/Plugins/Windows/ShimmerRT.dll along with its core dependency ShimmerClosedLibraryRev_04.dll from the Shimmer SDK.

The plugin greatly eases connection to and data retrieval from the Shimmer IMU. All of the necessary external functionality required has been abstracted in to a single public class, ShimmerControllerRT. This class can be instantiated within Unity and provides a clean interface to the Shimmer functionality. Data retrieval is achieved by supplying the ShimmerControllerRT's constructor with an implementation of the IShimmerHandler interface, an interface that declares a number of callback methods which are called when various types of data are received from the device.

The implementing class in the Unity application is the ShimmerDataListener class. An instance is passed to the controller's constructor and the methods are then delegated to the scripts which it references, specifically the UIController, ShimmerOrientationScript and the RecordingController.

This structure and its relations can be seen in Figure 4.5 (ShimmerControllerRT.png)

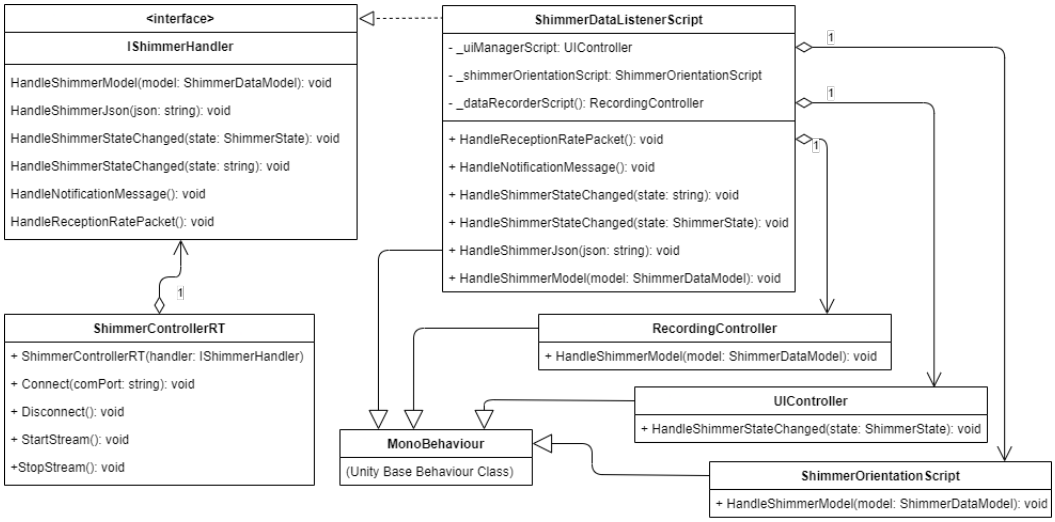


Figure 4.5: Windows Plugin UML

Android Plugin

An Android plugin project created in Android Studio which allows for access to native Android functionality in Unity through the use of Java code and the JNI. This plugin was developed to co-exist alongside the Windows Plugin and its design is very similar in that it utilises callback methods on a provided interface. The code has been compiled to a JAR file and included in the assets folder of the Unity project.

Unlike all other aspects of the overall project, this plugin requires its own Plain Old Java Object (POJO) class to hold the data recieved from the IMU. This class is a Java replica of the ShimmerDataModel class and it is serialised and deserialised just like its counterpart. As outlined above, this data is then serialised and sent to Unity where, upon reception, it is deserialised as a POCO.

Java Code Design The main Java class in use here is the AndroidUnityPlugin.java class, outlined in Figure 4.6. This class is a subtype of UnityAndroidActivity which itself is a subtype of the base Android Activity class. By extending UnityAndroidActivity, it is possible to attain a reference to this class from Unity C# code and invoke methods on it. The AndroidUnityPlugin.java class, therefore, is the single point of communication between the C# and Java code, utilising the JNI.

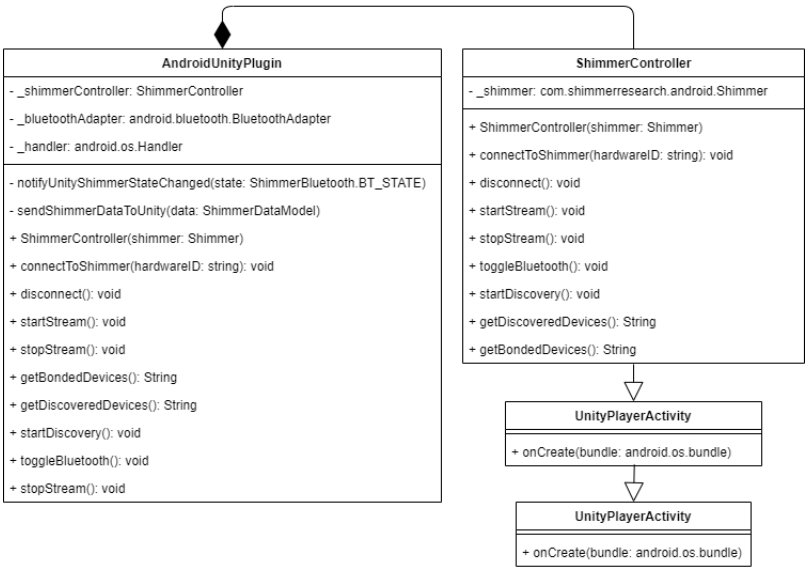


Figure 4.6: Java Android Plugin UML

C# Code Design Within Unity, a singleton class, AndroidUnityPluginWrapper handles all communication through the JNI, specifically to the AndroidUnityPlugin.java class. A reference is obtained to the Java class using classes and methods provided by Unity which can be seen in Listing 4.7.

```
// First get the activity class, this will always be UnityPlayer.
_activityClass = new AndroidJavaClass("com.unity3d.player.UnityPlayer");

// Get the context - this will be an instance of AndroidPlugin.java in the jar file
// This is defined in the AndroidManifest.xml as:
//    <activity android:name="com.example.unity_plugin.AndroidUnityPlugin"..../>
_activityContext = _activityClass.GetStatic<AndroidJavaObject>("currentActivity");
```

Listing 4.7: Getting a reference to the Android Activity class from Unity C# code

Once the reference has been obtained, the public methods within the AndroidUnityPlugin.java class can now be called from the C# code, shown in Listing 4.8.

```
public void ConnectToShimmer(string hardwareID)
{
    _activityContext.Call("connectToShimmer", hardwareID);
}
```

Listing 4.8: Calling a Java method from C# code

Due to the fact that the Java method names must be passed as strings, it made a lot of sense to use this wrapper class to obtain the references and call the methods. Obtaining a reference each time it is needed and then calling methods by string names can lead to mistakes and bugs which are hard to track down. Creating the wrapper class over this functionality limits the likelihood of code errors and means that the reference is obtained once and we simply call the methods on the wrapper whenever and wherever they are needed. In this case, the AndroidPluginScript also delegates most of its methods to the wrapper class, although some data transformations do happen, e.g. in the case of GetBondedDevices(), the wrapper returns string values as JSON, while the plugin script deserialises these values to a list of objects. An overview of the relationship between these two classes can be seen in Figure 4.7

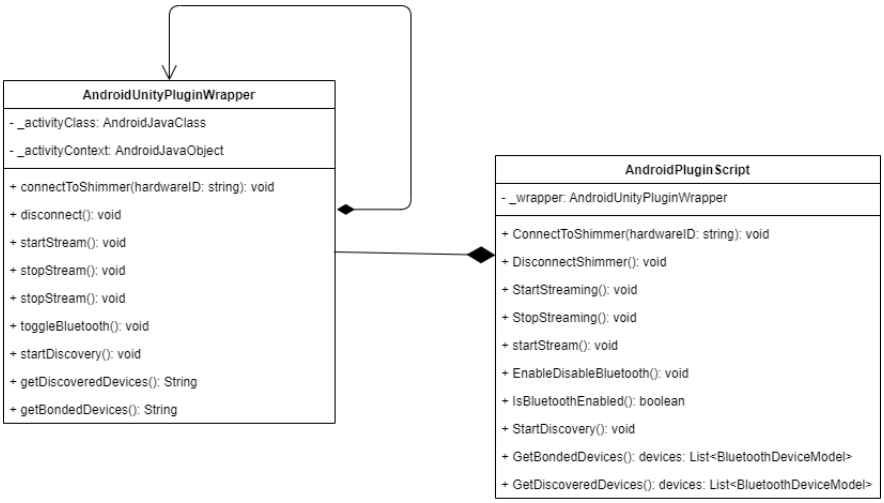


Figure 4.7: C# Android Plugin Wrapper and Script

Returning Data from the device through the JNI It is possible to send data through the JNI from any Java code to any Unity code, once a couple of requirements have been satisfied, i.e. we know:

1. The name of the class containing the method we wish to invoke - this class should only be instantiated once within Unity so a singleton pattern is useful.

2. The name of a public method within this class and its parameters.

Once the above has been satisfied, it is straightforward to call the method and supply an argument using the static UnityPlayer.UnitySendMessage method, as shown in Listing 4.9. The first argument here is the class name, the second is the method name and the third is any argument the method accepts. In this case, the HandleShimmerJson method only expects one string argument.

```
1 private void sendShimmerDataToUnity(ShimmerDataModel data)
2 {
3     // Convert the object to JSON.
4     String json = gson.toJson(data);
5     // Send the JSON to Unity - (class, method name, JSON)
6     UnityPlayer.UnitySendMessage("ShimmerDataListener", "HandleShimmerJson", json);
7 }
```

Listing 4.9: Calling a C# method from Java code

The above code will find the first instance of the ShimmerDataListener class in Unity and attempt to call the HandleShimmerJson method on it, supplying the argument 'json' as a string. The ShimmerDataListener class here receives the serialised IMU data, deserialises it and decides where to send this data, e.g. straight to the 3D model.

Plugin Selection at Runtime Depending on which platform (i.e. Android or Windows) the application is running on, one of the above plugins will be loaded at runtime. The plugin selection is based on a conditional provided by Unity which allows easy detection of the host operating system along with whether or not the application is running in the development environment, i.e. UNITY_EDITOR. This can be seen in Listing 4.10

```
#if UNITY_EDITOR
    LoadWindowsPlugin();
#elif UNITY_STANDALONE_WIN
    LoadWindowsPlugin();
#elif UNITY_ANDROID
    LoadAndroidPlugin();
#endif
```

Listing 4.10: Determining host OS in Unity

The plugin selection process is simplified by the implementation of a common interface (IShimmerPlugin) by both plugins, which means that the rest of the application code concerned with these plugins is unaware of the plugin implementation, but rather the specification provided by the interface. This also makes the application more modular in the sense that further support could be provided by another implementation of this interface. An overview of this class and interface structure can be seen in Figure 4.8

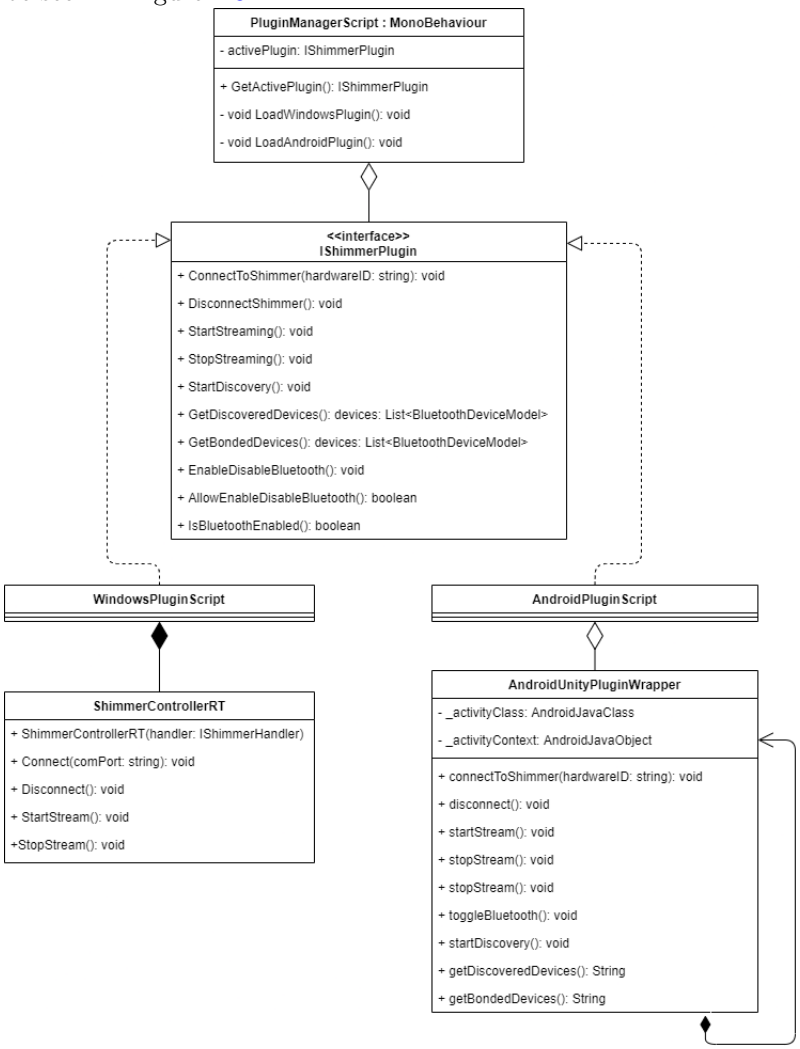


Figure 4.8: Plugin selection mechanism

4.3.3 Events and Actions

For certain functionality, we have made good use of Unity's event system in conjunction with C# Actions and Events[22]. A good example of this is the notification system which allows the UI classes to display a popup notification on the screen. Elsewhere, we have also used events to allow the UI to respond accordingly when an IMU device changes its state. These are just two examples of our use of actions events which are used extensively throughout the application. In the following

sections, we will show the use of these events in both of the aforementioned systems.

Notification System

The notification system has been designed using a queue to which new messages are added. The Update() method of the notification system checks the queue and if a message is present, the notification panel is enabled and brought into view with the message text displayed. Each message is displayed for a specific amount of time after which the queue is checked again for another message to display.

While the notification system itself is handled by the PanelNotificationScript class, it is the UIController which receives the events from the other UI classes and forwards the messages to the notification system. This design has allowed us to use the UI Controller as a central class which manages all UI events such as this. The declaration of the event can be seen in Listing 4.11 and we can see the event being subscribed to in Listing 4.12.

```
public static Action<string> OnNotificationReceivedAction;
public static void SendNotification(string message)
{
    OnNotificationReceivedAction(message);
}
```

Listing 4.11: UIControllerScript declares a static Action

```
// When UIControllerScript.OnNotificationReceivedAction action is fired,
// call this classes OnNotificationReceieved method.
UIControllerScript.OnNotificationReceivedAction += OnNotificationReceieved;

// PanelNotificationScript's implementation of OnNotificationReceived:
protected void OnNotificationReceievd(string message)
{
    // Add the new notification to the queue
    AddNotification(message);
}
```

Listing 4.12: PanelNotificationScript subscribes to the static Action in its constructor

Any UI class may now call the UIControllerScript.SendNotification method to send a message to the notification system as seen in Listing 4.13, where the PanelPlaybackScript sends the message "Playback Data Loaded" to the notification system.

```
private void OnSuccess(string path)
{
    /// code omitted..
    // Data Loaded Successfully, send notification...
    UIControllerScript.SendNotification("Playback Data Loaded");
    /// code omitted..
}
```

Listing 4.13: PanelPlaybackScript submits a message to the notification system

IMU State Change

Actions and events are also used to handle incoming state changes from an IMU device. These states are declared as an enumeration in C# and form part of the ShimmerRT DLL meaning that they are also accessible from the Unity code. This enum can be seen in Listing 4.14 and it can be seen that the IMU device has four main states which it can be in: Disconnected, Connecting, Connected and Streaming.

```
/// <summary>
/// Enum representing the various states a Shimmer can be in.
/// </summary>
public enum ShimmerState
{
    [Description("Disconnected")]
    NONE,
    [Description("Connecting")]
    CONNECTING,
    [Description("Connected")]
    CONNECTED,
    [Description("Streaming")]
    STREAMING
}
```

Listing 4.14: ShimmerState enum type

The change of state of the IMU is detected by the ShimmerDataListener's implementation of the IShimmerHandler HandleShimmerStateChanged method, seen in Listing 4.15. In this implementation, the ShimmerStateChangedAction event is triggered. This event is subscribed to by the PanelConnectionStatusScript which reacts to the new state and sets some of its panel attributes accordingly. The switch statement handling the state change on the panel can be seen in Figure 4.16.

```
public void HandleShimmerStateChanged(ShimmerState? state)
{
    if (state != null)
    {
        // Trigger the action so any listeners can react.
        ShimmerStateChangedAction(state.Value);
        // Send a notification to any listeners, e.g. Notification system.
        SendNotification($"Shimmer State: {state.Value.EnumValue()}");
    }
}
```

Listing 4.15: Implementation of IShimmerHandler HandleShimmerStateChanged

```
switch (_shimmerState)
{
    case ShimmerState.NONE:
        _imgConnectionStatus.sprite = SpriteDisconnected;
        _imgConnectionStatus.color = Color.red;
        break;
    case ShimmerState.CONNECTING:
        _imgConnectionStatus.sprite = SpriteDisconnected;
        _imgConnectionStatus.color = Color.yellow;
        break;
    case ShimmerState.CONNECTED:
        _imgConnectionStatus.sprite = SpriteConnected;
        _imgConnectionStatus.color = Color.green;
        break;
    case ShimmerState.STREAMING:
        _imgConnectionStatus.sprite = SpriteConnected;
        _imgConnectionStatus.color = Color.black;
        break;
    default:
        break;
}
```

Listing 4.16: Handling a state change in PanelConnectionStatusScript

4.3.4 Unity UI

User Interface (UI) development was of key interest to us, as developers who understand that the application not only depends upon well implemented code but on having an intuitive and responsive UI in order to provide a user-friendly experience. The application was developed to be modular and this is reflected in the design of the UI panels. For the user, this should provide a smooth experience from opening the app to connecting to a Shimmer unit, recording data or selecting an athlete profile. The initial app screen aims to present a simple view, uncluttered and free from notifications. We felt it would be best to use simple animations for pulsing or rotating items to reflect an action taken.

The icons used were specifically chosen to represent known actions within the app. As part of the research for the UI design we looked at many different applications on mobile and PC platforms in an effort to identify icons which were commonly used without description, i.e. icons that are intuitively understood to have a specific purpose and require no explanation. We then identified where in our application they would be put to use. The ubiquitous 'burger' icon represents the menu in many applications and websites, and needs no explanation to the user so this icon was used to toggle the collapsible main menu which can be hidden from view when unused to free up screen real-estate. A comparison of the main menu in both open and closed positions can be seen in Figure 4.9

Menu System

Designing the look of the application was an iterative process. Our intention was to utilize as much of the screen as possible with the shimmer model so that the panels which contained any functionality were designed to slide in and slide out from view. This effect, along with the collapsible menu system and the pulsing icons was achieved using the free DOTween[23] Unity asset which provides intuitive methods for manipulating Unity's UI elements - something which is not provided out of the box by Unity itself. The Unity Standalone File Browser[24] asset has also been used to simplify file saving in a cross-platform environment and these are the only two external Unity assets that we have used in this project which have not been developed ourselves.

The menu functionality was broken down into a number of distinct sections:

Main Menu The parent of all other menus. When the application starts, the entire menu is hidden from view save for the 'burger' button which toggles the menu from a collapsed state to a visible state. The menu burger icon is a ubiquitous and intuitive icon used widely to denote a menu. Throughout the UI design we provide the user with intuitive icons which perform actions that any user would be familiar with.

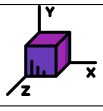
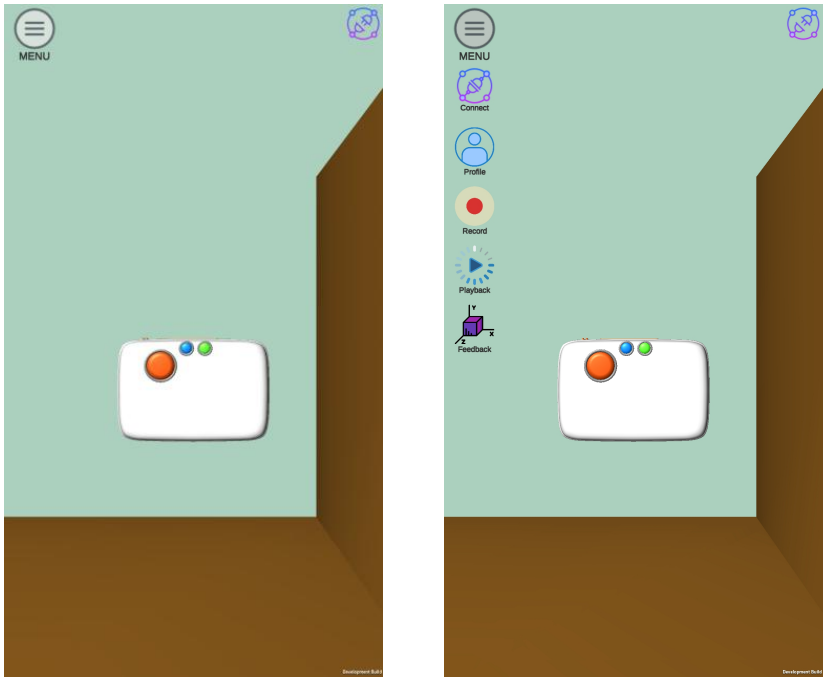
Icon	Action	Icon	Action
	Main Menu Dropdown		Open Connect Menu
	Open Profile Menu		Open Recording Menu
	Open Playback Menu		Open Feedback Menu

Table 4.1: Main Menu Icons.



(a) Menu Closed (b) Menu Open

Figure 4.9: The Main Menu in closed and open states, respectively

Connect Menu Allows the user to search for and display any Bluetooth devices within range. Discovered devices are added to a grid and the user may select a device and attempt to connect to it. Once connected, the device will automatically start streaming data to the application. Feedback is given to the user either through Toast notifications on Android, or though our own popup notification system on Windows.

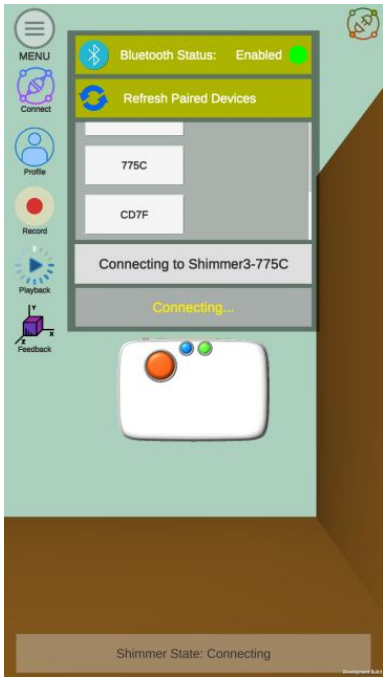


Figure 4.10: Connect Menu

Icon	Action
	Enable/Disable Bluetooth
	Refresh Devices List

Figure 4.11: Connect Controls

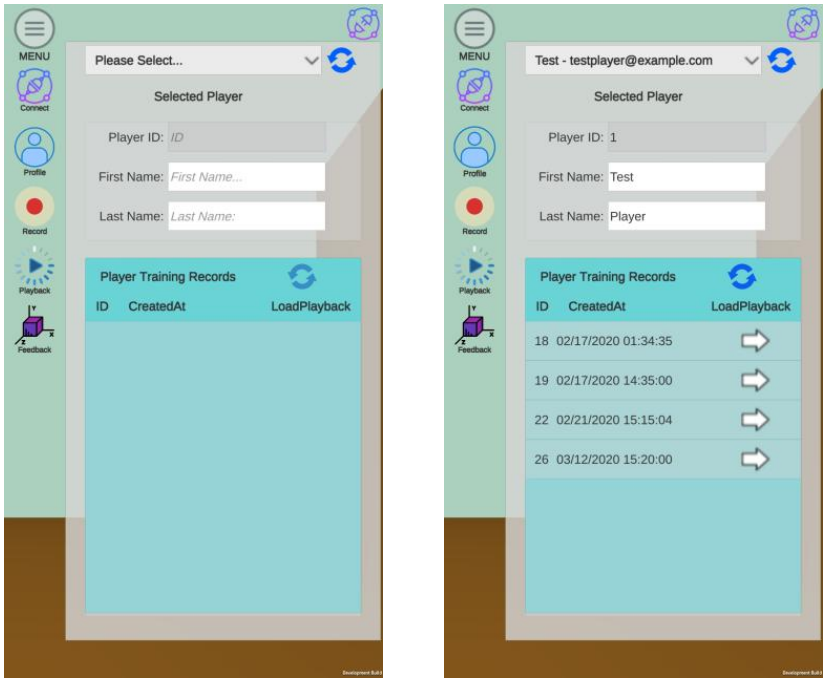
The connection menu displays the current bluetooth connection status as a green/red LED along

with 'enabled'/'disabled' to the user. The bluetooth icon is a button which allows enabling/disabling bluetooth. As mentioned, the refresh button will discover paired devices within bluetooth range and display them in a scroll box for selection. The refresh icon rotates clockwise on click. feedback to the user that the button is performing an action.
We have also added a connection status icon to the top right corner which has four distinct display phases. The connection status is directly pulling it's state from the IMU state through a class library. For each of the four states it displays a different colour.

STATE	COLOUR
ShimmerState.NONE	RED
ShimmerState.CONNECTING	YELLOW
ShimmerState.CONNECTED	GREEN
ShimmerState.STREAMING	BLACK

Table 4.2: Connection Notification States.

Profile Menu Retrieves data from the RESP API and populates the dropdown list with any Player data which has been received from the server. Selecting a Player here allows any data recorded by the application to be appended to a training record for that player and uploaded to the REST API. Additionally, it is possible to select a previously recorded training record for a player and playback that data in the application. Fig 4.12 displays the profile menu system. We can see the recorded data for the selected Player. Selecting the arrow will load this training record into memory, allowing it to be played back from the playback menu.



(a) Profile Empty (b) Profile Populated

Figure 4.12: The Profile Menu before and after selecting an athlete.

Record Menu This menu panel allows the user to record live data streamed from the IMU. This data is stored in memory and can be saved locally to a CSV file or, provided a Player profile has been selected from the Profile menu, uploaded to the REST API as a Training Record for that Player. The file saving implementation also differs between Android and Windows and so we have used the FileBrowserHelper asset to assist in implementing this functionality.

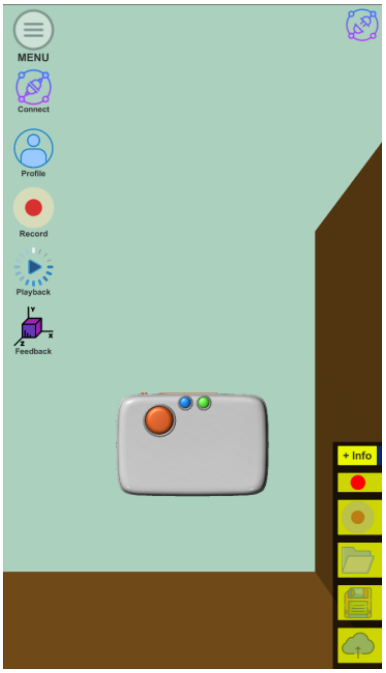


Figure 4.13: Playback Menu

Icon	Action
	Start Recording
	Stop Recording
	Open Directory Browser
	Save File
	upload to Cloud

Figure 4.14: Playback Controls

A notification system will notify the user when the file has been saved, using the events and actions as described above. Similarly, the user will be notified if attempting to upload data without having a profile selected.

Playback Menu This panel will only be useful if live data has been recorded by the application or if a Player profile and Training Record have been selected from the Profile menu. Once data is present, this panel allows for playback of the in-memory data and provides a number of standard playback controls seen in table in figure 4.16:

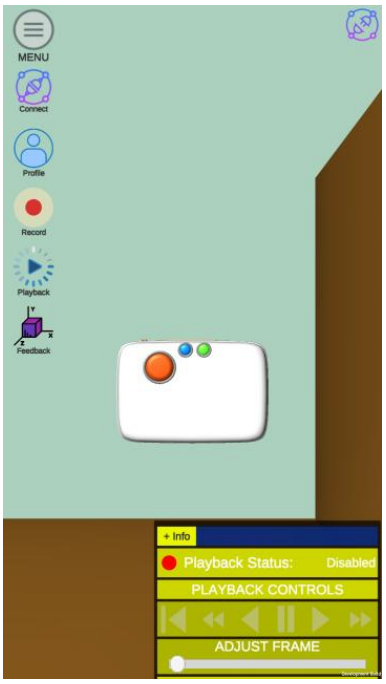


Figure 4.15: Playback Menu

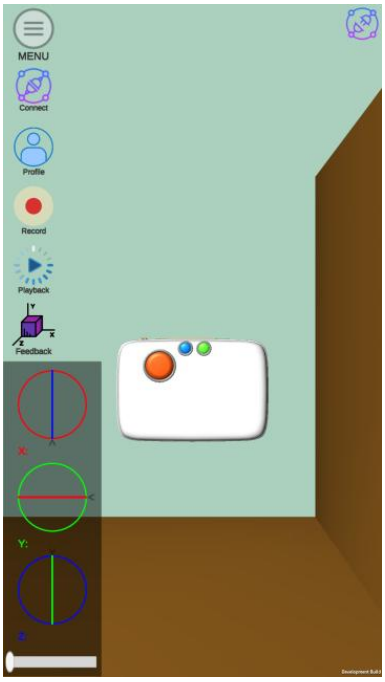
Icon	Action
	1 Skip to start
	2 Fast rewind to start
	3 Play backwards to start
	4 Pause
	5 Play forwards to end
	6 Fast forward to end
	7 Skip to end

Figure 4.16: Playback Controls

In addition to the standard playback controls, we have also added a slider control which allows the point of playback to be scrolled forwards or backwards frame by frame. The design of the playback

panel was a result of numerous iterations and meetings with the clients and having potential end users test run the application for us, offering feedback and suggestions until we arrived at the current design which was approved by all involved in the testing.

Feedback Menu The feedback menu is designed to provide angular feedback to the user. For each axis there is a dial, with each dial having a line through it. The dials represent the axes (X,Y.Z respectively), while the line through the axis in each case, is representative of the viewing angle.
For example, while viewing the X-axis dial, the blue line shows that the user is viewing along the Z-axis.



(a) Feedback Empty

Figure 4.17: The Feedback Menu.

The Feedback menu is available to assist the user in interpreting live data streaming from the shimmer, or as a guide when viewing playback data from an athlete profile. The intention is to offer a more tangible readout of the shimmer units orientation than a floating point number for the degrees of each axis, which was an earlier representation used in development. The menu also features a sliding toolbar which can be used to rotate the shimmer model within the application. Both the radial readouts and the sliding bar are a result of live testing with end users, gathering feedback from their user experiences and iterating over the design to develop some intuitive feedback as opposed to ambiguous degrees.

4.3.5 REST API Requests

HTTP requests to the REST API were accomplished using the static methods in Unity's UnityWebRequest class. This class provides a platform agnostic method of sending requests from Unity and is the recommended method of adding such behaviour to a Unity application. Due to the way that the methods in this class handle errors, we built up a small, two class structure of static methods to make error handling more efficient. One class contains generic methods and the other has concrete methods for each type of object that is to be sent or received via HTTP.
The reasoning for taking this approach was that rather than reproducing the same boiler plate code everywhere that a request was needed, and for each data type, we could simply call a static method and provide two callback methods - OnSuccess and OnError. Also, full CRUD functionality was not needed for any particular model, so we have kept the RestService contained within one class.

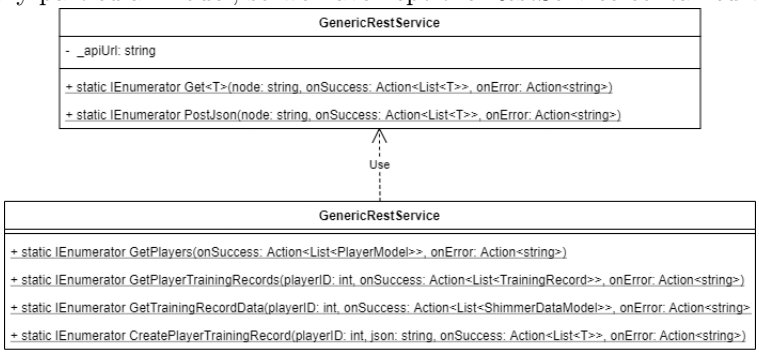


Figure 4.18: REST service in Unity application

An example usage of the service in Figure 4.18 would look like the code snippet in Listing 4.17.

```
private void GetPlayers()
{
    // Send a request to retrieve all players from REST API.
    // Note that the two callback methods are also passed.
    StartCoroutine(RestService.GetPlayers(GetPlayersRequestCompleted,
    GetPlayersRequestError));
}

private void GetPlayersRequestCompleted(List<UnityPlayerModel> players)
{
    // Request successful, do something with the data received
    _players = players;
    UIController.SendNotification("Player data received successfully!");
}

private void GetPlayersRequestError(string msg)
{
    UIController.SendNotification("Error fetching Player data!");
}
```

Listing 4.17: Example REST Service usage in Unity

4.3.6 IMU data

Receiving Data and Notifications from IMU As explained in the [Bluetooth Plugins Section], data from the IMU is handled by different means depending on the host operating system. What is key to note, however, is that on both systems, the data is always received by the same script within the Unity application. This script is the main point within the application which handles any incoming data from the IMU, whether it is sensor data or message notifications informing us that the connection has been lost or similar.

This class, ShimmerDataListenerScript.cs, is only ever instantiated once within the application and each plugin interacts with this script by calling various methods on it. The class is an implementation of IShimmerHandler, shown in Figure 4.19. Through its implementation of these interface methods, the script handles incoming data from the device and sends the data to the

Unity GameObjects concerned with that type of data.

For example, incoming status messages from the IMU are routed to the UIController where they are submitted to the notification system. Incoming sensor data is sent directly to the script concerned with manipulating the on screen 3D object and the data feedback system. If recording has been enabled, data models are also sent to the recording system where they are stored in memory until the user decides to either save them to local storage or upload them as a Training Record related to a Player profile.

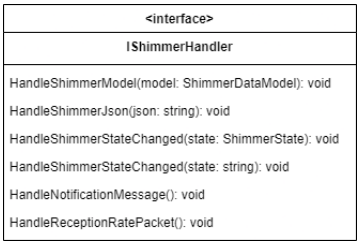


Figure 4.19: IShimmerHandler interface

Rotation Transformation We know from the Shimmer3 IMU documentation that the default coordinate system used by the device is right-handed coordinates. Unity, on the other hand uses a left-handed coordinate system. To get the IMU data into a form which can easily be used within Unity we have used a static method, ConvertRightHandedToLeftHandedQuaternion() to achieve this. To be consistent across all uses and storage of the IMU data, this data transformation is only applied to the IMU data as it is being used within the scene. Wherever we are storing this IMU data, e.g. to file or uploading to the REST API, the data is left as it was received, i.e. in right-hand coordinate system. Should different types of IMU ever be introduced to be used with this application, this strategy should be adhered to to maintain a certain level of data consistency. Listing 4.18 contains the code we have used to achieve this coordinate system conversion. As per common guidelines regarding working with Quaternion values, rather than trying to manipulate the Quaternion, we simply create a new one passing in the correct arguments.

```
1 public static Quaternion ConvertRightHandedToLeftHandedQuaternion(Quaternion
      rightHandedQuaternion)
2 {
3     return new Quaternion(-rightHandedQuaternion.x,
4                           -rightHandedQuaternion.z,
5                           -rightHandedQuaternion.y,
6                             rightHandedQuaternion.w);
7 }
```

Listing 4.18: Converting a right hand Quaternion rotation to Unity's left hand system

4.4 Xamarin Client Application

4.4.1 Overview

This supplementary application was designed to be used as a data entry application for users of the Unity application. Due to the lack of richness of Unity's UI controls for extensive data input across multiple fields and data types, we opted to use a separate application to handle this data entry and to then POST the data to the REST API server. Xamarin Forms[25] offers a set of rich cross-platform controls and so was a good option for this small application.

The application retrieves data from the REST API server through HTTP requests and can also POST data to the server. Full CRUD functionality is offered for the PlayerModel data model, allowing the user to create new players, update existing players' data and also remove any player from the database.

4.4.2 Application Design

The Model-View-ViewModel pattern has been used to develop this application as it is the recommended pattern to be used with Xamarin Forms applications. Additionally, the Repository Pattern has been used in the data access layer which creates an abstraction in the data access layer, hiding the exact implementation of how the data is fetched from the rest of the application through the specification of a number of interfaces. In the following sections, we will take a more in-depth look at these two patterns and how they promote abstraction, code reuse and portability.

The Model-View-ViewModel (MVVM) Pattern

MVVM[26] is a popular architectural pattern used in many modern frameworks. It helps to separate the business and presentation logic of an application from its UI. MVVM is a core aspect of the Xamarin framework as it allows separate UI implementations for each supported platform if required. An overview of this pattern can be seen in Figure 4.20.

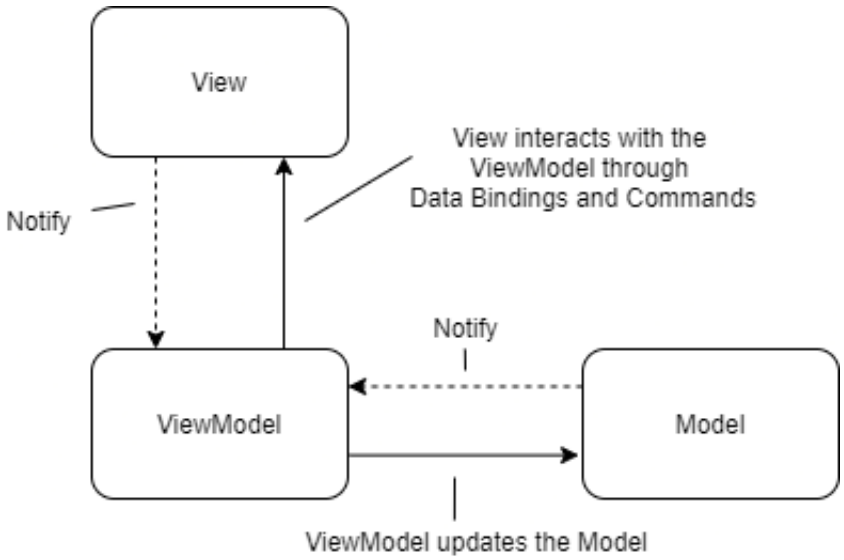


Figure 4.20: Overview of the MVVM architectural pattern

The Model A layer of classes which encapsulate the application's data. We are using the classes described in the Shared Code section section as our application models. Within the MVVM pattern, it is generally expected that POCO classes are used, however, Data Transfer Objects may also be used should the need arise. In our case, we have used POCOs and our data access is handled by the Repository pattern.

The View Visual classes responsible for the layout and appearance of what the user sees on the screen. In Xamarin forms, the view layer is a collection of XAML classes and associated partial C# code classes. The view layer should only ever interact with the ViewModel layer and never directly with the model layer. Views generally contain an instance of one or more view models and can bind to the properties of the view model through data binding syntax, along with being able to send Commands to the view model without directly calling any methods on it.

The ViewModel Sitting between the Models and the Views, the ViewModels can be thought of as an adapter for the model which is to be presented to the view. Viewmodels define the functionality which is to be offered to the view, but the view defines how that functionality is to be displayed and interacted with. Viewmodels generally are generally bound to one or more models and provide a mechanism for presenting an abstraction of the model to the view, allowing a means for the view to update the model through this abstraction and also for the view to be updated by changes made in the model. The core mechanism for achieving this in Xamarin Forms (and other DotNet frameowrks) is through the implemenatation of the INotifyPropertyChanged[27] interface.

When used correctly, an implementation of the INotifyPropertyChanged allows changes to flow from model to view and vice versa, while neither ever interact directly with each other. Collections maintained in Viewmodels should also use the type ObservableCollection, a collection which provides functionality similar to INotifyPropertyChanged, meaning that any items added to or removed from this collection will notify the view accordingly.

Example Implementation Figure 4.21 represents the relationship between the PlayerModel, PlayerViewModel and the PlayerDetailView - a view used to display data related to a single Player in the UI.

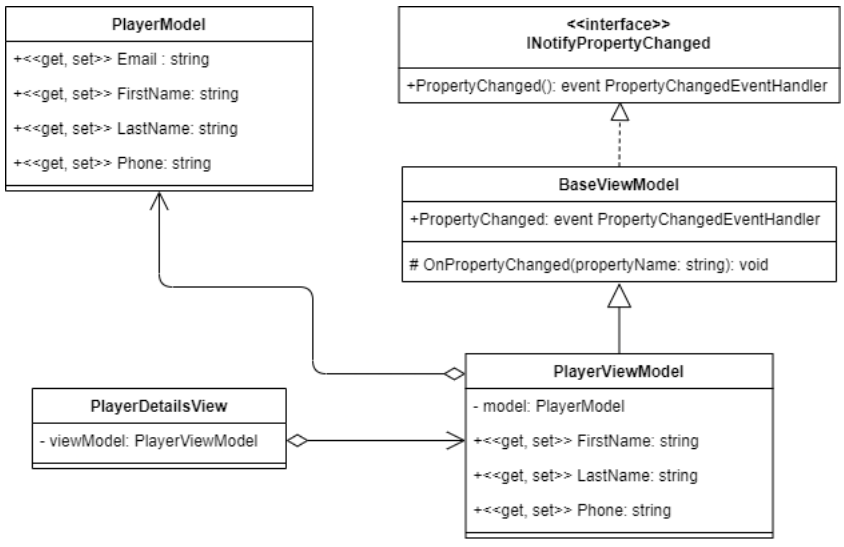


Figure 4.21: Example of MVVM pattern applied to PlayerModel and one of its related views, PlayerDetailView

It can be seen that the viewmodel inherits from a base class, BaseViewModel, which implements the INotifyPropertyChanged interface. This approach cuts down on code repetition as it means that this interface only needs to be implemented once within the application. An instance of PlayerModel is then bound to the viewmodel, which exposes the underlying model's properties as properties of its own, e.g. the FirstName property of PlayerModel is not exposed directly. Each property exposed is done in a certain fashion, ensuring that the INotifyPropertyChanged. This method of exposing properties is shown in Listing 4.19.

```
public class PlayerViewModel : BaseViewModel
{
    // reference to underlying model
    public new PlayerModel _model { get; private set; }

    // Sample ViewModel property implementation
    public string FirstName
    {
        get => _model.FirstName;
        set
        {
            if (_model.FirstName != value)
            {
                _model.FirstName = value;
                OnPropertyChanged();
            }
        }
    }
    // remaining code omitted for brevity
}
```

Listing 4.19: Sample ViewModel Property

It can be seen in Listing 4.19 that each time the FirstName property is set, the OnPropertyChanged() method in the base class is invoked. This is the mechanism through which binding is achieved, as the View is 'notified' of this change by the view model.

Repository Pattern

Overview The repository pattern is an architectural pattern commonly used to avoid duplication of data acesss logic throughout an application. The purpose is to hide the data access implementation by adding a layer of separation between the data and domain layers of the applicaton. The repository pattern allows for easy querying of data, without having to provide such things as a connection string or REST API url each time a query is to be executed. The design of the repository pattern abstracts all of this underlying implementation away from the calling classes and allows the application data to be accessed as easily as an in-memory collection of data. This abstraction also allows for data access methods to be swapped in and out easily at one location in the code, while not affecting any of the calling classes. This last point became invaluable as we eventually migrated from using Firebase realtime database to a SQL Server instance exposed thorugh a REST API server.

Example Implementation Figure 4.22 provides an overview of the pattern as implemented within our client application and focusing on the PlayerModel data model. We begin by creating a generic repository interface, in this case IRepository<T> - this interface describes the standard behaviour expected by any application data model, i.e. CRUD - Create, Read, Update and Destroy. Note that create and update have been combined into a single method, Upsert() to further cut down on code duplication. Extending the generic interface is the typed interface IPlayerRepository which describes all of the CRUD behaviour expected by a PlayerModel type while also describing two further methods specific to this type.

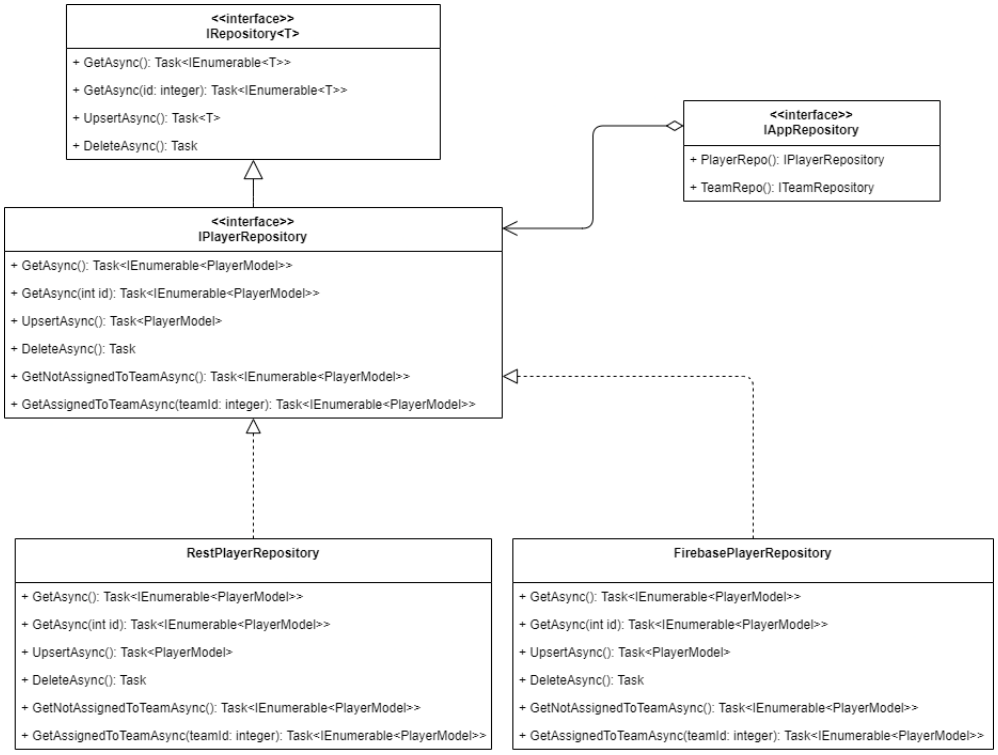


Figure 4.22: Example of the Repository Pattern applied to data access for the PlayerModel data type

It can be seen that there are two implementations of the IPlayerRepository interface - RestPlayerRepository and FirebasePlayerRepository. The Firebase repository was the original class which we had used to access Player data, when this data was stored in the Google Firebase realtime database. As our project matured, we came to the realisation that when it came to storing a large amount IMU data, Firebase was not as well suited to this type of data storage as SQL Server. At this point, we began migrating our data to SQL Server, accessed through a REST API. From the persepective of this application, all that needed to be done was to create another (REST) implementation of the IPlayerRepository and use this for data access instead.

The IAppRepository interface describes the application-level data access method. It provides two C# properties declared as interface types - PlayerRepo and TeamRepo. Since these types are declared as interfaces, it was possible to easily switch over from the Firebase repository implementation to the REST implementation without needing to alter code anywhere else within the application. This is one of the strong points of the repository pattern and made our developement of this application much easier as a result.

REST API Calls The code snippet in Listing 4.20 is the implementation of the IRepository<T> GetAsync(int id) method taken from the RestPlayerRepository class. The code contains an example of how the HTTP request is sent asynchronously and how the response data is handled and deserialised from JSON to a PlayerModel.

```
1
2  // Example HTTP client initialisation.
3  _client = new HttpCLient(_serverUrl);
4
5  // REST endpoint to be appended to server URL
6  _nodeName = "players";
7
8  // Return the PlayerModel from the database with ID=id.
9  public async Task<PlayerModel> GetAsync(int id)
10 {
11     // Make the HTTP request.
12     var json = await _client.GetStringAsync(_nodeName + "/" + id);
13     // Use the built-in deserialiser to create a PlayerModel instance from the json
       .
14     var player = JsonConvert.DeserializeObject<PlayerModel>(json);
15
16     return player;
17 }
```

Listing 4.20: Sample REST request and deserialisation from RestPlayerRepository

User Interface This interface was intentionally kept simple and straightforward since its only use is for the input of player profile data. Data validation has been implemented on the input form to limit undesirable null or empty data fields being populated in the database. Additionally, fields expecting data such as email addresses are validated as such through the use of method utilising regular expressions to ensure that the entered data is of an expected form. This email validation code can be seen in Listing4.21. The field validation of the input form can be seen in Figures 4.23 and 4.24.

```
1  public class EmailFormatValidationBehaviour : Behavior<Entry>, IValidatorBehaviour
2  {
3      const string emailRegex = @"^(?("")("".+?(?<!\\)""@)|(([0-9a-z]((\.(?!\.))
       |[-!#\$%&'\*\+/=\?\^`\{\}\|~\w])*)(?<=[0-9a-z])@))" +
4      @"(?(\[)(\[(\d{1,3}\.){3}\d{1,3}\])|(([0-9a-z][-\w]*[0-9a-z]*\.)+[a-z0-9][\-a-
       z0-9]{0,22}[a-z0-9]))$";
5
6      void HandleTextChanged(object sender, TextChangedEventArgs e)
7      {
8          IsValid = (Regex.IsMatch(e.NewTextValue, emailRegex, RegexOptions.
       IgnoreCase, TimeSpan.FromMilliseconds(250)));
9          ((Entry)sender).TextColor = IsValid ? Color.Default : Color.Red;
10     }
11 }
```

Listing 4.21: Sample ViewModel Property

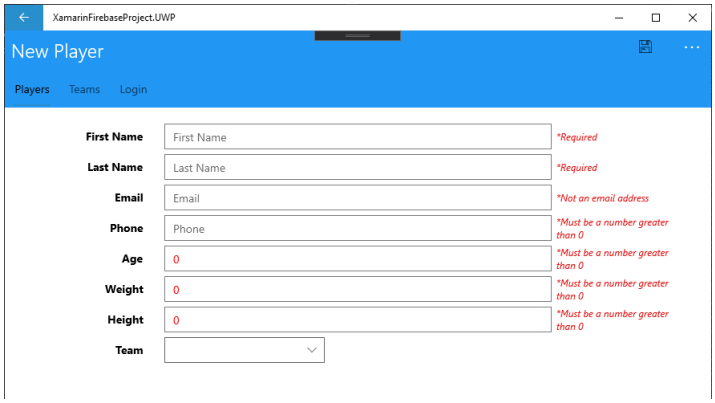


Figure 4.23: The Player Profile input screen with invalid data.

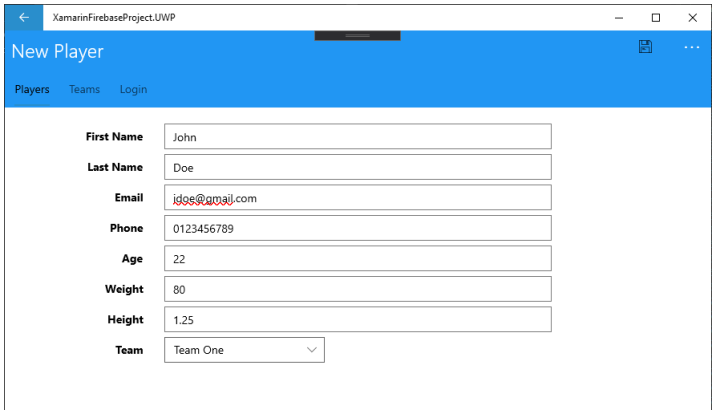


Figure 4.24: The Player Profile input screen with valid data.

Chapter 5

System Evaluation

5.1 Overview

To the best of the developers' ability, all software was written to conform to best practices and professional standards. The architectural design is loosely coupled and modular in every possible instance. Looking back at the objectives we set out during the Introduction and the requirements we identified during the Methodology section, and the meetings with clients and supervisor alike, the following sections is a retrospective look back at the development process. We aim to clarify our achievements and shortcomings, and elaborate on the things we learned during this development.

Application Objectives

- Deploy Mobile application with the following features:
  - Stream data via Bluetooth.
  - Gather real-time data from an athlete.
  - Option to enable and disable device Bluetooth.
  - Record IMU data.
  - Store IMU data.
  - Playback IMU data.
  - Target the UI towards a display of relevant data. ie: the angles of incline of the athletes spins and its rotation.
  - Intuitive and uncluttered UI.
- Deploy a cross-platform application to manage a database of teams and athletes.
- Develop a cloud based solution to manage teams and athletes.

5.2 UI Testing

Given the nature of the Unity UI framework and Unity's entity-component system, testing of UI elements in code was not practical so we took the approach of behaviour driven development for this aspect of the project. Expected functionality was laid out and the UI was tested on each iteration to verify that features still worked as intended. Following is a list of all menu features within the application and the expected behaviour from interacting with certain elements and the expected response from other elements, e.g. a certain UI element's icon should change.

5.2.1 Side Menu

- Always Openable.
- Click Menu - Opens Side Menu.
- Click Connect - Opens Connect.

- Click Profile - Opens Profile.
- IF Profile Selected:
  - Click Playback - Opens Playback.
  - IF Connected:
    - Click Record - Opens Record.

5.2.2 Connect

- Always Openable.
- Select Device is restricted until devices are refreshed and a device is selected.
- Start Stream is restricted until device is connected.
- Refresh Button refreshes list of paired devices.
- Selecting a paired device updates 'selected device' button to 'connect to device'.
- Selecting 'connect to device' establishes connection to selected device.
- Flash notification 'connecting' during attempted connection.
- IF Connection Established.
  - Flash notification 'connected' after connection established.
  - 'connect to device' changes to 'disconnect device'.
  - Start Stream is unrestricted and will begin streaming data from the device.
  - Start Stream updates to Stop Stream.
- * Android Only *
  - Bluetooth Panel Visible.
  - Bluetooth Icon Button enables/Disables Bluetooth on device.
  - Flashes permission request if clicked when bluetooth is disabled.
  - Status LED red for bluetooth off.
  - Status LED Green for bluetooth on.

5.2.3 Profile

- Always Openable.
- Select Player dropdown list populated with athletes.
- Refresh button refreshes dropdown list with available athletes.
- Selecting athlete from list populates Selected Player display.
  - Selected player displays the ID, First and Last name of the player.
  - Player Training Records displays the list of selected player's training records.
  - Refresh button refreshes the list of training records.
  - Each training record has ID, Created timestamp, and Load to playback button.
    - Load to Playback button loads the selected training record into playback
- Upload JSON
- Test Shimmer Data

5.2.4 Playback

- Requires an Athlete Profile to be selected.
- Requires a training record to be loaded.
- Led Playback Status.
  - Red while playback is disabled and green during playback.
- Playback Controls.
  - Rewind To Start.
    - Rewinds session playback to the start.
  - Rewind a frame.
    - Step backwards one frame in playback.
  - Play backwards.
    - Play the session in reverse.
  - Pause.
    - Pause the session.
  - Play forwards.
    - Play the session forwards.
  - Forward a frame.
    - Step forwards a frame.
  - Frame Slider.
    - Slide forwards and backwards through the playback using the slider.
- Load Session (Do We need this button? The user will be loading in data from the profile panel)
  - Opens a file picker to load a session for playback.

5.2.5 Record

- Requires an Athlete Profile.
- Requires a Shimmer Connection.
- Led Recording Status.
  - Red while recording is disabled and green while recording.
- Recording Controls
  - Start Recording.
    - Records the stream of data from the shimmer.
  - Stop Recording.
    - Ends the stream of data from the shimmer.
  - Save Location.
    - Opens a file picker to chose where to save the data.
  - Save Session.
    - Saves the recorded data to the chosen save location.
  - Upload Session to cloud.
    - Uploads the recorded data to the players profile in the cloud.

5.3 Testing the REST API

Given that the REST API was developed along with an integrated frontend, as outlined in the System Design section, testing of all REST API requests was performed through this interface. The Postman application for creating, testing and debugging HTTP requests and responses was used at length during this phase and became an invaluable tool in this regard. Postman allows for the creation of a suite of HTTP requests to be created and run end-to-end against a given API. Using this process, after any change to code within the server application, these tests could be rerun to verify that the expected server responses were as intended. Coupled with our REST design process using SwaggerHub and the OpenAPI standard, new test requests could easily be created by referring to the design specifications. An example of these requests within Postman can be seen in Figure 5.1. While more robust solutions exist for testing such applications, given the iterative approach of our design and implementation, we feel that this process worked well for us and allowed us to identify both the source and cause of any issues very quickly.

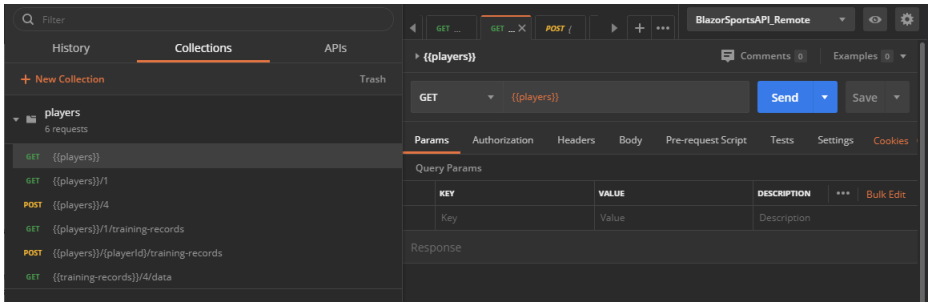


Figure 5.1: Example suite of requests used for testing the API

In conjunction with the above testing process, the security in place on the server application also required testing. For this, a set of modified requests were built in Postman incorporating the use of JWT authentication and the expected results compared against the actual HTTP responses. For example, if JWT was not used against any of the endpoints, a 401 response should always be expected, confirming that the endpoint is unauthorised.

5.4 Hardware Testing

Due to the extra layer of complexity introduced when working with hardware such as the IMU devices, it was difficult to devise a suitable testing solution to test all functionality: i.e. connection, disconnection, streaming, verification of received data. To this end, we first focused on being able to consistently connect and disconnect from the IMUs. Leading on from this, once connected to a device, we then verified that streaming could be enabled and disabled at will without causing any undesirable effects. Once we were happy with the connection and communication with the devices through the exchange of status updates, we then went about verifying that the data received from the IMU was in the expected format and was applicable to our use, namely, applying rotations to a 3D model and the ability to detect an impact through real time analysis of the accelerometer data.

To verify that the data being received from the IMU was as expected, we used a process of recording the output of data from the IMU by our application in repeatable positions, e.g. Y-axis straight up, X-axis aligned with a fixture, rotate IMU 90 degrees right, etc. Once we had recorded this data, we could then repeat the same process as above, but this time using the Shimmer provided software, Shimmer Capture, to record the data. By comparing the output CSV data we could then confirm that both data sets contained very similar data. When we take into account that these rudimentary tests were not performed with any type of precision and a certain margin of error must be allowed, we were happy that the results were similar enough that our application was indeed reading and outputting the data as expected. Further to this, once we had progressed further into development and the Unity application was maturing, we could see first hand that the movement of the IMU was replicated on-screen by the 3D object.

Chapter 6

Conclusion

Having compared studies on video and wearable sensor technology, the literature has shown that there have been significant advances in the understanding of injuries as they relate to sports. Injury is complicated as it can be difficult to quantify what constitutes injury [5], hence injury prevention can also be a difficult task. However, collecting and analysing data in real time with the use of wearable sensors and biosensors provides a level of detail and abstraction to the challenge of injury detection and prevention, such as to enable trainers, coaches and physicians to make informed decisions about their athlete's health and safety during training and performance. Video analysis is the benchmark used to establish a standard of accuracy for wearables, and has paved the way for wearables to become reliable solutions for gathering athlete data. It also has the added value of providing visual context for the more abstract data gathered by wearables. Wearables have advanced to include biosensors [4], which puts more actionable information in both the athlete's and coach's hands, informing important training and recovery plans by tracking player workload and exhaustion during training and performance. With regard to rugby, there have been advances made in the early detection of head injury with the use of wearable sensors [3]. This has player safety as its core concern and has the effect of making the sport safer for players. Such is the impact of wearable sensors and video analysis in sports in general, making the sports themselves safer for the athlete, and putting critical information in the athlete's and their trainers hands to tailor their training and recovery plans - key factors in preventing injury.

Throughout the research elements of this project, we were unable to confirm which genders were included in the testing of the wearable sensors. In the case of 'Automatic detection of collisions in elite level rugby union using a wearable sensing device' [3], this is of critical importance as there are differences in body mass and muscular distribution between the genders, which means injury detection and prevention plans will have different parameters. It seems this a good candidate for further research as there might be a bias towards one gender over another, which leaves gaps in the data. The research undertaken in Gardeners article in 2015 however does consider women's rugby, and concussion has a lower frequency than in men's rugby [6, p.1]. Our research findings also identifies a consensus with regards to wearable sensor technology. As highlighted, the technology has limitations which don't appear to have been solved yet. The trade off between device size and power management being the leading challenge, amongst other challenges related to the safety and integrity of the devices themselves, and the security of the data gathered. During development, the developers had to contend with the battery life of the Shimmer IMUs. They would often be powered on for long periods of time while testing implementation or connection status and while they did require recharging, it was by no means an obstacle to development. Similarly, it does not appear this would be an obstacle to training as the units performed adequately for hours on end without any power failure.

Much of the literature researched points out that video analysis is a key factor in verifying and validating data gathered from wearable technology. Video analysis when used in conjunction with abstract data from wearable technology has lead to a deeper understanding of the causes of injury, which in turn informs plans and strategies to prevent injury. Athlete preparation for their sport is critical to the prevention of injury. This is how they can build an objective understanding of their bodies limits, how they react to training and performance, and in addition, how they can optimize

their recovery from any sustained injuries. The athlete, and training staff can differentiate between objective stress and perceived stress, and the relationship between training and performance, and sustaining injury. As highlighted throughout this document, wearable technology shows great potential to prevent injury in rugby. While there are challenges yet to be resolved, it has been demonstrated and documented that data gathered from wearable sensors greatly increases the potential to identify causes of injury. Analysis of the data can assist in developing management plans, injuries can be prevented, mitigated and monitored to keep athletes health and safety as the primary focus.

From a development perspective, this project has provided invaluable experience to us both as final year students. We feel that the experience gained in liaising with prospective clients and end users, gathering requirements and building specifications gave us a diverse sampling of some of the key skills which would be expected of us in our future careers. We also gained great experience and a realisation that project requirements are not always set in stone and may require some evolution as a project develops in line with set objectives and user expectations. As has been elaborated on in the methodology section, we met with clients and supervisor frequently. This set standards for us and introduced ideas and challenges we might not have considered if we had developed the project without the insight of experienced developers in the case of our supervisor and the clients who represented the end users of the application.

Throughout the course of the development, we became aware of the impact of some early design choices which we had made based on our knowledge of problems we faced at the time. Some solutions we had invested development time into would turn out to be unsuitable for the project. This was a lesson in oversight, had we researched more in-depth with regard to the limitations of a technology, we may have avoided re-working some sections of code. Similarly, it is a reflection of not fully understanding requirements - a lesson that surely comes with experience. An example of this was scope creep. As the project increased to include storing bulk data, we found our decision to develop the database with Firebase was less than ideal and this issue has been explained in more detail in the Technology Review section. As a result of this, we referred back to our previous research and improved our storage solution in line with the demands of the data we were dealing with. Although this meant introducing a whole new application to the project, we felt that it was a necessary change to make, both from a performance perspective and for a better overall user experience.

Further to some shortcomings in the implementation of our solution which were rectified as they were encountered, a challenge we feel we did not complete and could be improved is the feedback from the application. As developers it made sense to us, we are seeing it on a daily basis for months on end and perhaps a bit of tunnel vision set in. When it came to having a mostly finalised application and we had the opportunity to gather user feedback on the visual appearance and feel of the application, this was the most frequently commented on aspect. The radial dials are ambiguous and this system while functional to some degree, leaves the user a bit confused as to what they're looking at. In our time at GMIT we have undertaken many projects and the majority of these were focused on individual effort. At the outset of this project, we were given the option to undertake our final year project alone or as part of a team. We chose the latter and both feel that it has served us well, primarily due to the fact that it has opened our eyes to the benefits of collaboration between developers with differing skill sets. We recognised that each of us had key skills in developing a project of this scope and this recognition was of key importance when it came to breaking the project up into a number of smaller, more manageable tasks. We each took responsibility for the tasks as we agreed to them, delegating the project in this manner which required maturity and commitment from both developers. Collaborating through Github was invaluable. Learning to manage repositories, branches, tracking issues and making many mistakes along the way was a fantastic learning experience, and another that will benefit future roles in software development.

Appendix

GitHub Project Repository

Bibliography

[1] Eline van der Kruk and Marco M Reijne. Accuracy of human motion capture systems for sport applications; state-of-the-art review. *European journal of sport science*, 18(6):806–819, 2018.

[2] Yewande Adesida, Enrica Papi, and Alison H McGregor. Exploring the role of wearable technology in sport kinematics and kinetics: A systematic review. *Sensors*, 19(7):1597, 2019.

[3] Daniel Kelly, Garrett F Coughlan, Brian S Green, and Brian Caulfield. Automatic detection of collisions in elite level rugby union using a wearable sensing device. *Sports Engineering*, 15(2):81–92, 2012.

[4] Tyler Ray, Jungil Choi, Jonathan Reeder, Stephen P Lee, Alexander J Aranyosi, Roozbeh Ghaffari, and John A Rogers. Soft, skin-interfaced wearable systems for sports science and analytics. *Current Opinion in Biomedical Engineering*, 2019.

[5] DJ Chalmers. Injury prevention in sport: not yet part of the game? *Injury Prevention*, 8(suppl 4):iv22–iv25, 2002.

[6] Andrew J Gardner, Grant L Iverson, W Huw Williams, Stephanie Baker, and Peter Stanwell. A systematic review and meta-analysis of concussion in rugby union. *Sports medicine*, 44(12):1717–1731, 2014.

[7] ShimmerSensing. Shimmer 9dof calibration user manual rev2.10a. http://www.shimmersensing.com/images/uploads/docs/Shimmer_9DOF_Calibration_User_Manual_rev2.10a.pdf, 2017.

[8] ShimmerSensing. Streaming to pc. http://www.shimmersensing.com/images/uploads/docs/Streaming_to_PC.pdf, 2019.

[9] ShimmerSensing. Streaming to android. http://www.shimmersensing.com/images/uploads/docs/Streaming_to_Android.pdf, 2019.

[10] ShimmerSensing. Shimmer user manual rev3p. http://www.shimmersensing.com/images/uploads/docs/Shimmer_User_Manual_rev3p.pdf, 2017.

[11] ShimmerSensing. Shimmer3 spec sheet v1.8. http://www.shimmersensing.com/images/uploads/docs/Shimmer3_Spec_Sheet_V1.8.pdf, 2019.

[12] ShimmerSensing. Logandstream firmware for shimmer3 user manual rev 0.11a. http://www.shimmersensing.com/images/uploads/docs/LogAndStream_for_Shimmer3_Firmware_User_Manual_rev0.11a.pdf, 2018.

[13] Andrew Dillon. *User Interface Design*. 01 2006.

[14] Martin O'Reilly, Darragh Whelan, Charalampos Chanialidis, Nial Friel, Eamonn Delahunt, Tomas Ward, and Brian Caulfield. Evaluating squat performance with a single inertial measurement unit. pages 1–6, 06 2015.

[15] Google Developers. Android SDK Platform release notes. https://developer.android.com/studio/releases/platforms/, 2020.

[16] Microsoft. Entity Framework. https://docs.microsoft.com/en-us/aspnet/entity-framework, 2020.

[17] Microsoft. Code First to a New Database. https://docs.microsoft.com/en-us/ef/ef6/modeling/code-first/workflows/new-database, 2016.

[18] Microsoft. The MVC Design Pattern. https://dotnet.microsoft.com/apps/aspnet/mvc, 2020.

[19] Microsoft. ControllerBase class. https://docs.microsoft.com/en-us/aspnet/core/web-api/?view=aspnetcore-3.1#controllerbase-class/, 2020.

[20] Oracle. Java Native Interface Specification. https://docs.oracle.com/javase/7/docs/technotes/guides/jni/spec/jniTOC.html, 2018.

[21] Shimmer Sensing. ShimmerCapture/Shimmer C# API. https://www.shimmersensing.com/products/shimmercapture, 2018.

[22] Microsoft. Events (C# Programming Guide). https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/events/, 2015.

[23] Deimgiant. DOTween. https://assetstore.unity.com/packages/tools/animation/dotween-hotween-v2-27676, 2015.

[24] Gökhan Gökçe. Unity Standalone File Browser. https://github.com/gkngkc/UnityStandaloneFileBrowser, 2018.

[25] Microsoft. Xamarin.Forms Documentation. https://docs.microsoft.com/en-us/xamarin/xamarin-forms/, 2018.

[26] Microsoft. The MVVM Design Pattern. https://docs.microsoft.com/en-us/xamarin/xamarin-forms/enterprise-application-patterns/mvvm, 2017.

[27] Microsoft. INotifyPropertyChanged Interface. https://docs.microsoft.com/en-us/dotnet/api/system.componentmodel.inotifypropertychanged, 2020.