University of Plymouth

School of Engineering, Computing, and Mathematics

COMP3000

Final Stage Computing Project

2020/2021

ICU Tracker App

Samuel Jordan

10611160

BSc (Hons) Computer Science

Acknowledgements

Many thanks to my project supervisor Liz Stuart, my family, my girlfriend and my mentor Dave who helped me through the project during the pandemic.

Abstract

Doctors currently have to manually track the progress of their patients during their time in ICU. Because of this, patients cannot easily view their progress. The ICU Tracker app is designed to allow this, as well as enabling them to perform exercises themselves, without the need for external help.

A large amount of research had to be done regarding both the technologies and the accessibility features needed to accomplish this. The main technologies used in the project were Xamarin and SQL Server, as well as a number of others. The combined use of these enabled me to finish the project within the deadline set.

This report details the aims and objectives for the project as well as the many planning and development steps I took to achieve these. It also covers the legal and ethical issues I encountered throughout.

The app was an overall success, with the app being both accessible, and containing the features needed for it to be fully functional within the ICU of Derriford hospital.

Contents

Statement of Word Count

Word Count: 9119

Code Submission

The developed app code can be found at:

https://github.com/SamuelJordan101/ICU-Final-Year-Project

1 Introduction

1.1 Background & Aims

This project is based upon Liz Stuart's idea for a Patient Rehab Tracker App. This app would be used inside of the Intensive Care at Derriford Hospital with the aim being to allow the patients to track their progress through the ICU and speed up their rehabilitation process.

Currently doctors have to manually track the progress of each patient and the patient is only able to see their progress during these face-to-face meetings. This app focusses on the patients, allowing them to view their goals and achievements, as well as giving them the ability to add their own. The app is also being developed to give patients access to exercises they can complete at home; this is due to its huge impact on the speed of which the patients are able to progress (Stiller, 2013).

The project also focusses on the accessibility to the users, this is due to it being developed for use within the ICU, where many of the patients will likely have some sort of a debilitating physical impairment. This is also why the project is based upon an app, so that it can be used anywhere, whether that be in the hospital or at home. This means that even if they have left the ICU of the hospital, they are able to continue their rehabilitation.

The app is designed to be used alongside another app to be developed by another student in which the doctors are able to set the patient goals, as well as look at how their patients are progressing.

1.2 Existing Solutions

While the app in this form doesn't currently exist, modern day fitness apps such as Google Fit and Nike Run Club offer a lot of the main features needed. These

however, cannot be used due to the specific nature of the project and the features that the apps need to have. For instance, the use of sensitive data that only Derriford Hospital would have access to. These apps are also very specifically based around fitness, whereas the ICU app being developed needed to include other features such as CPAX scores and physiotherapy-based exercises. Further research was done into these apps in section 6.1 to see what features I could incorporate from them.

1.3 Deliverables

The following is a list of the deliverables that are needed on the completion of the project:

- An Android and IOS ICU tracker app.
    - A user guide for the app.
- A report of the whole project (this document).
- A video detailing the app and background.

1.4 Minimum Requirements

By the end of the project, there are a number of minimum requirements that the app should reach to be functional. These are as follows:

- Must Run on Android.
- Must Run on IOS.
- Must use a database to store data.
- Must use an API to interact with the data.
- Must have some basic functionalities available to the users.
    - Viewing their own data.
    - Adding new data.
    - Deleting their own data.

2 Method of Approach

Before the planning of the project had started, I had to think about what the structure of the project was going to be. This includes what items needed to be done as well as how the project was going to be completed. This was essential in ensuring that the project would be completed on time to a good standard.

One such approach I used was the agile methodology. This was created around sprints, as a way to be able to iteratively develop the project in smaller sections rather than everything at the beginning, like with the waterfall methodology (Atlassian, 2021). This allows features to be added and removed at any point within development. Sprints also allow a more accurate timeframe of completion to be calculated, as it was easy to see if I was in front or behind of schedule. This was important due to the timeframe in which the project needed to be completed. Sprints are also looked at in more detail in section 4.2.

These sprints also worked alongside a product backlog; a set of tasks created at the start of development to allow for the quick understanding of what needed to be created for the app to be functional (Agile Alliance, 2021). This was incredibly important in looking at not only what needed to be done, but also looking how long

each item would take, which helped me organise what needed to be completed at each stage of development.

Chapter 4.1.2 details Trello, a technology I used to be able to track these sprints as well as tasks in them and their set due dates.

3 Legal and Ethical Issues

While working on the project, there were a number of legal and ethical issues that needed to be thought about before development started. This includes how data should be stored and the ethical issues surround an app in an ICU.

3.1 Legal Issues

There were a few legal issues I had to consider about when developing the app. The first of which was the handling of user's data due to the Data Protection Act 2018 (Edwards, 2018).

In the case of my project, the user's name and location as well as a number of other data sets are stored. To ensure that my app followed the principles laid out by the Data Protection Act, each piece of data stored from the user needed to be checked to make sure that it is completely needed. This was done prior to the creation of the ERD to ensure that it would only have to be dealt with once, as it would have required redesigning of the whole database.

As this is not a fully-fledged app, security is not a huge priority, however some basic security measures were put in place to keep it relatively secure such as the hiding of IP addresses. The app will also only be downloaded for those in ICU and not available for the public, so there is less of a problem for unauthorized access to the app. This is especially important as there is no password function, which would need to be implemented if the app were to be used in a public environment.

3.2 Ethical Issues

The main ethical issue that comes with the development of my app is due to the use of it in an ICU. It is important that care is taken to ensure that the app is usable for people with a multitude of different physical and/or mental disabilities, especially with it being used inside of a hospital. Failure to make it accessible could make it extremely hard to use, or even completely unusable for some users, which would undermine the whole meaning for the app.

Another ethical issue relating to my app is the storage of user data. While it is also a legal issue, it is important that only the bare minimum user data is stored that is necessary for the app to work. Generally extra data stored from the user is used for advertisement purposes, which in this case would be highly unethical.

4 Project Management

4.1 Project Management Technologies

4.1.1 GitHub

GitHub was used as an easy way to be able to backup and keep a track of the changes to the app (Kinsta, 2021). GitHub is both fast and secure, so was perfect for

the backing up of my project. One repository was used for both my code and the documentation that goes alongside it, this made it easy to see progress on everything that had been done in the project, including the planning at the beginning of the project and the development of the app later on.

Rather than using the command line interface of Git to backup to GitHub, I opted to use the GitHub Desktop program. This was useful as it simplified the process of committing, pushing and pulling from a GitHub repository. This meant that I didn't have to remember the commands to push and commit to my repository, which sped up the process of using GitHub.

4.1.2 Trello

Trello was used to be able to keep track of everything that I needed to do regarding the app. In the early stages of development, when documentation was being created, a backlog of items was created. Along with this, "Working On" and "Completed" sections was used to log what needed to be done, what had already been done and what I was working on at the current time (Figure 1). Each item on my backlog also had its own date upon which it should be completed. This helped me to keep on track so that the project would be completed on time.

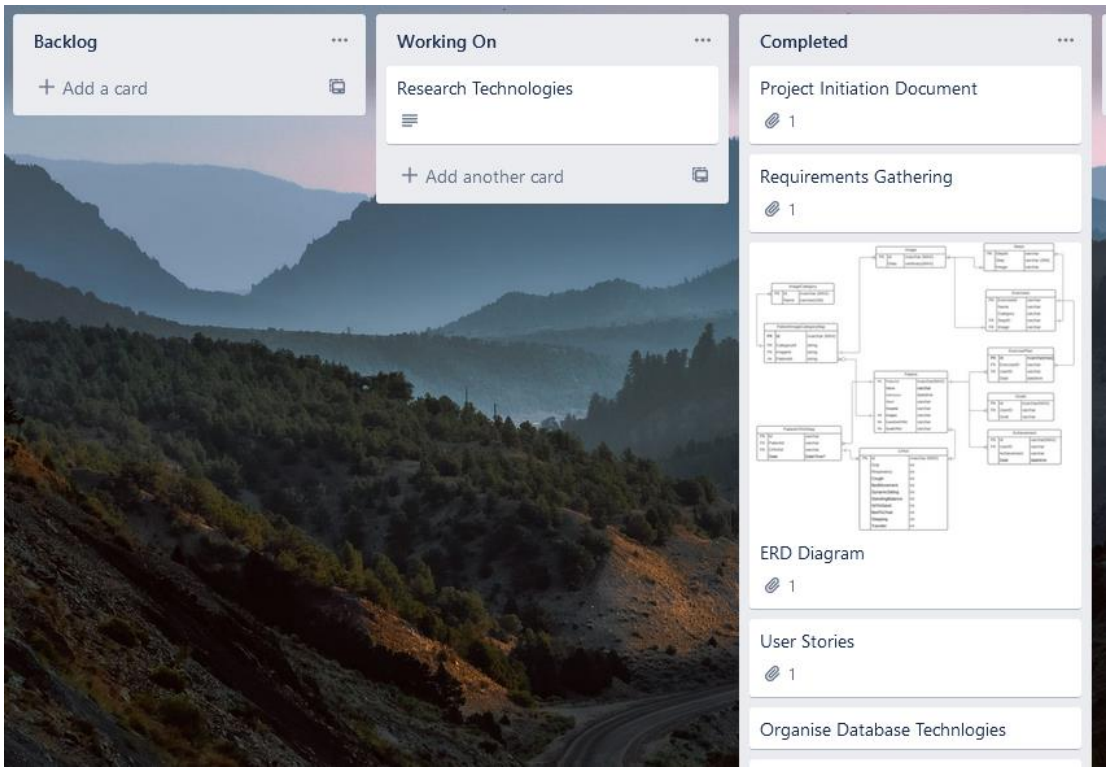


Figure 1. Trello board with backlog, working on and completed sections.

Further on in the project, a series of sprints were also created on my Trello board (Chapter 4.3, Figure 2).

4.2 Sprints

To ensure that my project would be completed on time, I used sprints to make developing the app more manageable and consistent. These sprints consisted of

fortnightly blocks which contained all the items that I would be working for in those weeks and when they should be completed by.

These sprints were added into Trello (Figure 2) which gave me the ability to quickly add or remove tasks as well as add individual dates for each item so that I could split the work down further. The use of Trello made it a lot easier to be able to track the progress of the app development and helped a lot with the management of time to ensure that the project was finished on time.

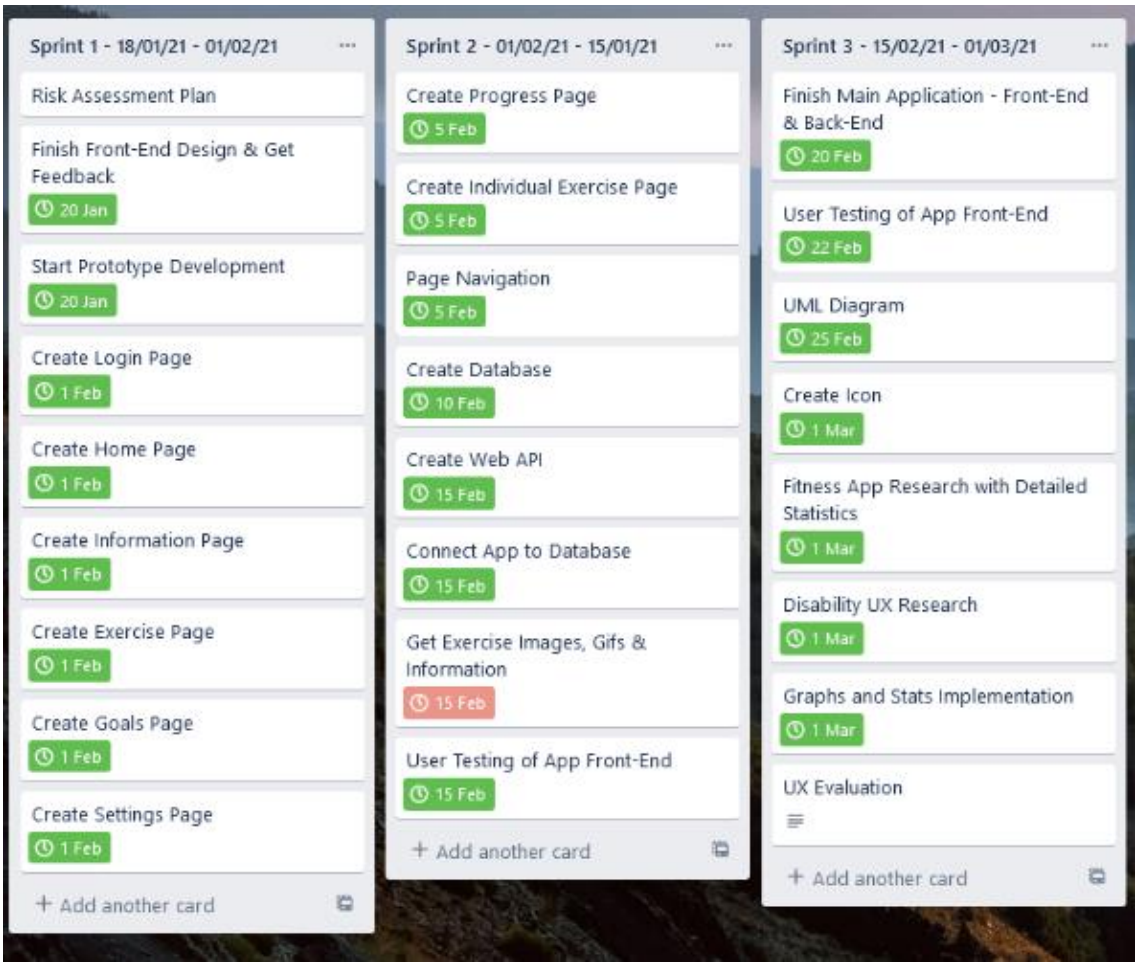


Figure 2. Trello board with sprints.

4.3 DevOps/Development Pipeline

My development pipeline consisted of regular pushes to my GitHub repository after any major change or feature had been made. This made it easier to keep track of any changes and made it easier to reverse changes if I were to add or change anything that broke the app.

4.4 Bi-Weekly Reports

With the project, I added bi-weekly reports to the DLE in a blog. These usually consisted of a couple short paragraphs detailing what I had been doing for the past fortnight as well as what I was planning to do in the next one (Figure 3).

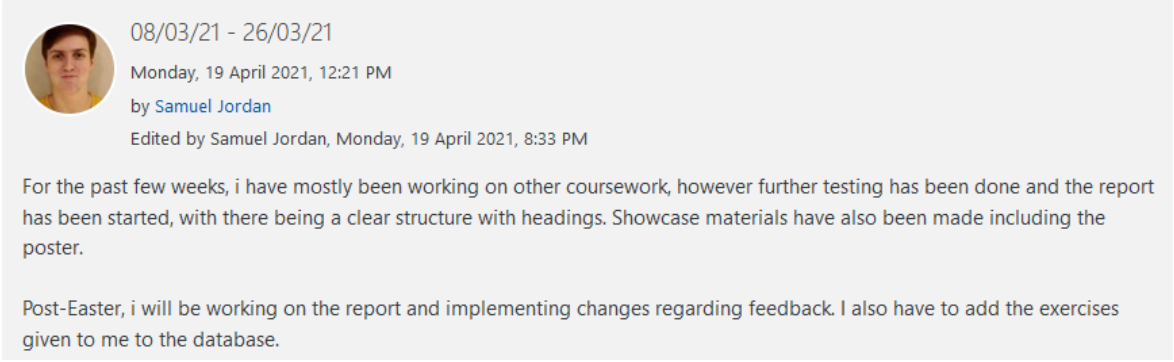


Figure 3. 7th Bi-weekly Report of the project.

4.5 Tutor Meetings

As well as the reports, I regularly met up with my tutor and a group to discuss what we were working on. These meetings were extremely helpful, as I managed to gain valuable feedback from other students who were doing similar projects. My tutor also helped a lot, as she had experience with similar projects before, so gave me some good advice regarding some parts of my app.

5 Requirements

A set of requirements were created based on what the app needed to be able to do at the end of the project. The requirements also included some optional features that could help to improve the main functionality of the app.

5.1 Functional Requirements

A list of functional requirements were made prior to the development of the app. This helped to target what was needed to be able to get the app to a working status, and what would be an optional extra to add later on. Table 1 shows a list of the functional and non-functional requirements for the app.

Table 1. Functional and non-functional requirements.

Functional Requirements	Non-Functional Requirements
Allow the user to login	Allow the user to see the exercise in video format
Allow the user to view their details	Allow the user to select blind friendly colours
Allow the user to view their progress through CPAX Scores	Allow the user to view a calendar with their goals
Allow the user to set their own goals	Allow the user to view their set exercises
Allow the user to be able to see and set achievements	Allow the user to be able to mark exercises as “Completed” or done

Allow the user to see their progress via a graph or chart	The user should be reminded to do their next exercise
Application should store data in a database	The app should be available on multiple platforms

5.2 Time Management

Time management was extremely important throughout the whole project due to the strict deadline at the end of it.

I used tools such as Trello to help keep track of all the items to be completed, in progress and already completed. This helped me to stick to a plan, continuously working on the project throughout the months. The use of sprints also helped greatly with this as it motivated me to complete sets of features by specific deadlines. Sprints also broke down the project into smaller, more manageable tasks which made it a lot easier to work on over time.

6 Research

As I had not previously created a mobile app, a lot of research needed to be done to look at what is expected from a modern-day mobile app. Research was also done on the usability of the mobile app due to it being used in the ICU of a hospital.

6.1 Fitness Apps

Research was done into both the features and overall aesthetic of current day fitness apps, as the app I was creating was very similar.

Two major leading apps were looked at, Nike Run Club and Google Fit, with Google Fit being more similar to the ICU app being created. A document was first created, including screenshots of all of the important pages to the app. I then went through both apps, noting down their pros and cons, looking at what could be a good addition to my app or a feature that might instead hinder it (Table 2, Table 3).

Table 2. Nike Run Club pros and cons.

Pros	**Cons**
Simple to navigate	A lot of empty space
Features are sorted on left menu bar	Unselected text is hard to see
Challenge’s feature – can add dates	
Simple contrasting colours	
Font is big and easy to see	

Table 3. Google Fit pros and cons.

Pros	**Cons**
Selection of dark/light mode	Dropdowns are hinted in wrong places

Home screen shows quick information	Some text can be hard to read
Drop downs used when possible	Text size is a little small (could be due to device settings)
Colours are used to accent clickable items	
Buttons are labelled	
Intuitive placement for items	
Settings allow customization	
Home chart page has a radial chart and text – either can be used	

These evaluations of current apps made a big impact to the features I decided to add to the final app, most notably the use of colours and font sizes.

6.2 Technologies

Research was done on the technologies I was going to use for the app, including the framework, API and database.

Some of the frameworks researched include Xamarin, React Native and Flutter, as well as the potential use of a PWA (Progressive Web App). All of these were looked at in detail, collecting it the major points within a document to quickly compare them. This contained each framework's pros and cons, as well as their features and compatibility with other software, such as database and API support.

The result of the framework research was that while React Native and Flutter have more native features, Xamarin would likely be better due to its support with other technologies such as ASP.Net Web API and SQL Server. While Xamarin does have less features, it also has a larger list of supported third party libraries so it would be easy to find the features that I wanted, if they weren't included with Xamarin. Xamarin is also supported by Visual Studio, which is something I have used many times before, so it would be easier to develop with than learning a new IDE.

As Xamarin was chosen as the framework for the app, the ASP.NET Web API framework was chosen to develop the API. This is due to their compatibility together, as they are both developed by Microsoft, they are natively supported by one another. ASP.NET also has a lot of documentation behind it too, which made it easier to develop with than a lot of other API technologies.

I had used SQL Server before so, I knew that it would be well suited to use as the database for the project due to its ease of use and scalability. The prior experience with SQL Server was also extremely useful in lessening the development time of the app as I didn't have to spend time learning how to use it.

6.3 Accessibility

Research also had to be done to fit the accessibility needs of many of the users. This is due to the app being used in an ICU, so many of the users of the app will likely have some sort of physical impairment, whether that hinders their eyesight or use of touch.

For this research I looked at the common mistakes people make when developing accessible apps and what features I should be adding to help make it more accessible. I noted all of these as notes in a document, which helped assist me in the development of the app. The main problems mentioned were to do with the use of colour and an emphasis on user choice. This includes font size, colours and anything else that could make the app more usable for the user.

7 System Architecture and Design

A number of different documents had to be created prior to the development of the app to ensure that it could be created with the technologies currently available and that when development did start, it went smoothly.

Every part of the app that I developed was designed before it was created, so that any potential problems could be resolved as early as possible. For instance, if two technologies didn't work together or the back-end wasn't implemented correctly for a feature to be fully functional.

7.1 UML

A basic UML was created of the API to assist in the development of it (Figure 4), the UML created contains all the models, controllers and context file as well as the rough structure of the API. The UML made it quick and easy to be able to create the API as all of the data had already been thought about before the implementation had started. Creation of the UML also helped me solve problems early on, such as what data would be needed for the app to be able to work fully and what isn't.

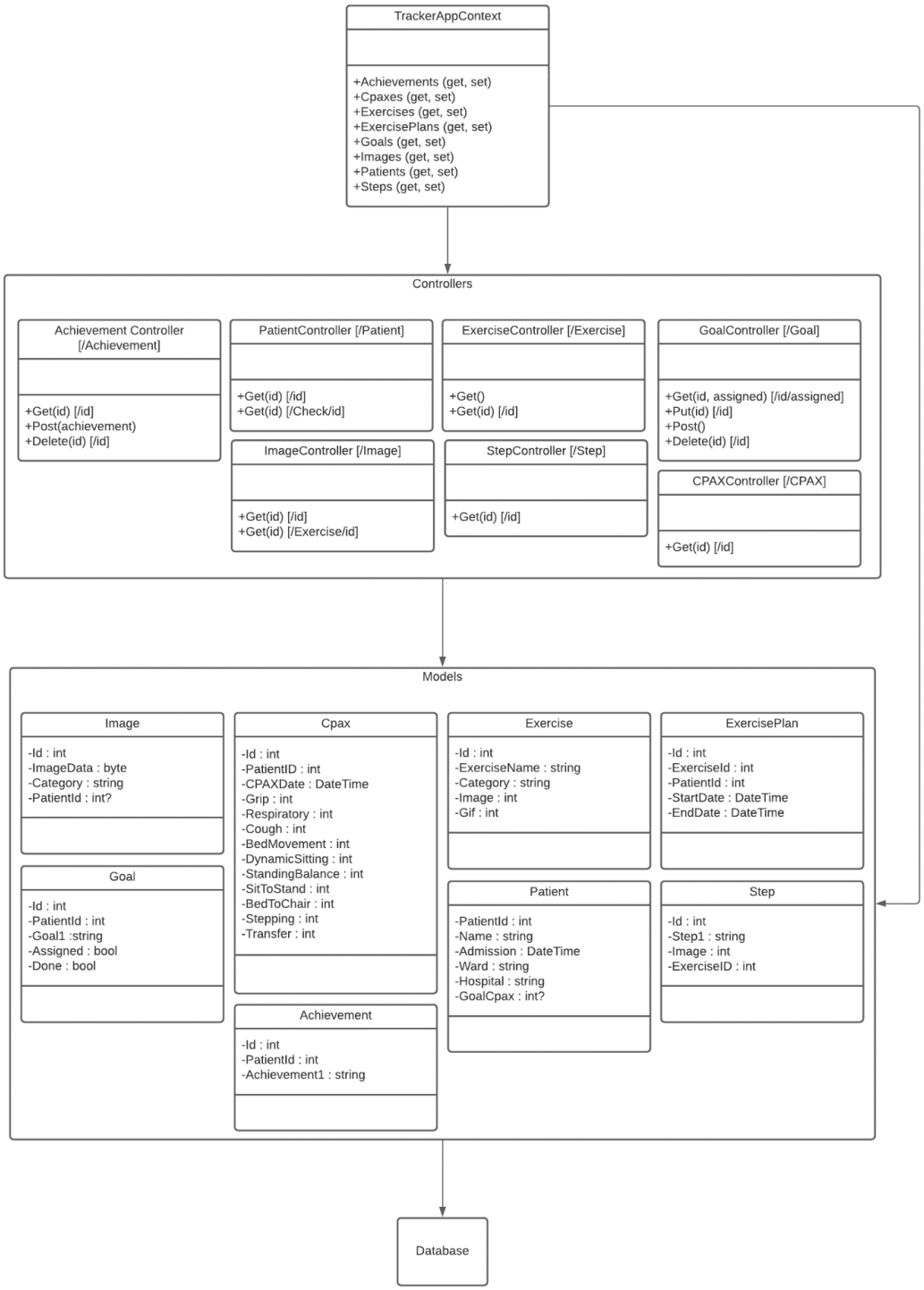


Figure 4. UML for the API

7.2 ERD

An ERD (Entity Relation Diagram) was created so that creation of the database could be straightforward, with all of tables and relationships being designed before implementation. With this, I had to think about all of the data fields that the app would need to use and have access to as well as what relationships needed to be created between these data fields so that the app wouldn't have data duplicates (Figure 5). I also had to think about redundancy in data, especially with data laws not allowing non-functional data to be stored.

The use of an ERD helped greatly with the development of the database for my app as it meant I had already considered what data needed to be stored. This made a lot easier and quicker to create the database, as I just converted the ERD to SQL queries and didn't have to worry about problems as they had already been thought about.

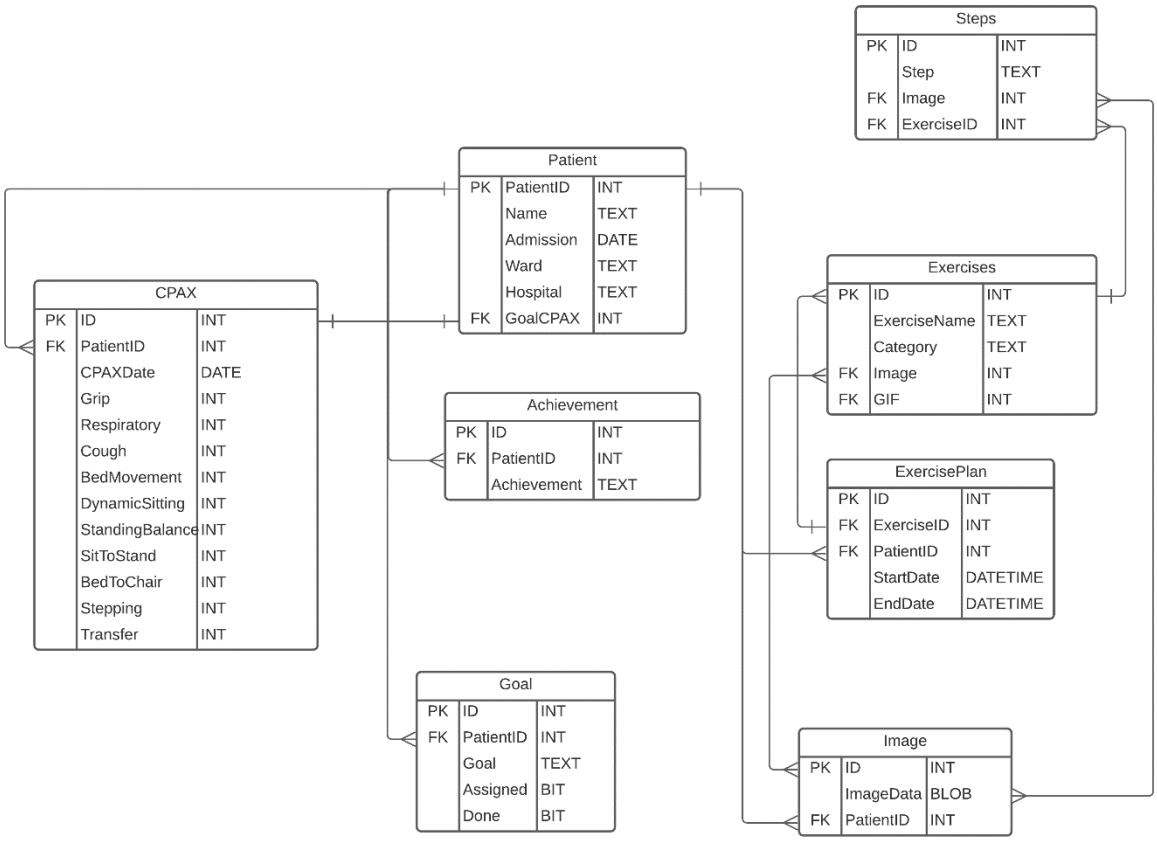


Figure 5. ERD created for development of the database.

7.3 PID

A project initiation document (PID) was created at the start of the project to outline what the project was about as well including a basic risk assessment. This gave a basic overview of what was being developed, why I think that it was needed and ultimately why it needed to be created, as without a purpose for the app being created the whole project would be unnecessary. The PID was important as it gives a starting point for anybody wanting to understand the project without having any prior knowledge of what it is.

7.4 User Stories

A number of user stories were created to help with the planning of the app. This shaped both the front and back-end of the application and helped me to understand some of the main functions that the app would need to be able to work. These user stories were used throughout development of the app to check that the fundamental parts of the app were working correctly. The user stories were also used later in development to base the user testing on. Table 4 shows the user stories created at the beginning of the planning phase.

Table 4. User stories.

User Stories
As a user I should be able to login
As a user I should be able to view my details
As a user I should be able to log out
As a user I should be able to view my goals
As a user I should be able to view my CPAX scores
As a user I should be able to view exercise gifs
As a user I should be able to view exercise steps
As a user I should be able to set goals for myself
As a user I should be able to mark goals as done
As a user I should be able to see achievements
As a user I should be able to set achievements
As a user I should be able to view a progress graph

7.5 Risk Assessment

A risk assessment was made to identify the risks that might be involved in the project, if any of them were likely to happen and if so, how I could mitigate the impact of these. This helped with the development of the project as it allowed me to overcome problems early on, rather than coming across them later into the project.

Table 5 shows all of the identified risks, their likelihood of happening and what can be done to reduce the probability of them occurring.

Table 5. Risk Assessment.

Risk	Impact (0-5)	Mitigation Techniques
Equipment failure	1	The likelihood of this happening is quite low; however, I have a number of devices on which the application can be developed. This should help to mitigate the impact if a device were to stop working.
Time Management	2	The use of sprints should help mitigate the chance for objectives to become late and overdue and ensure that the project is completed on time
Unfriendly UI	2	Research into the UX field, specifically to do with the use of technology for those with disabilities will be done to help reduce the chance.
Data Loss	2	Data will be backed up in multiple places, both locally and externally. This should help to mitigate the risk of data loss.
COVID	1	Much of the development can be done without COVID having an effect, however testing on IOS will be a problem. Safety measures will be in place to make sure that when borrowing a device to test the app on IOS, COVID will not be an issue.
Low Quality Code	2	General practices of good coding will be followed where possible. This includes actively commenting code and reviewing performance loss.

Software Learning	3	As I have little experience with the Xamarin and WebApp, work will be done early in the project lifecycle to ensure that the application is possible on the platform and can be completed in the time given.

7.6 Interactive Storyboard

In most software projects a storyboard is created in order to have a basic frame of the front-end layout (Figure 6). In my project, instead of this I decided to use Figma, which allows for the creation of interactive storyboards. This meant that the storyboard could be interacted with through a series of buttons, which were used to simulate navigation buttons throughout each page in the storyboard. That made it so that I could get some front-end feedback early on in the design stage before developing the front-end. This was vital as it allowed me to only create the front-end once and make minor changes, whereas if the front-end had been tested later on it may have required a full redesign which might have been very time consuming.

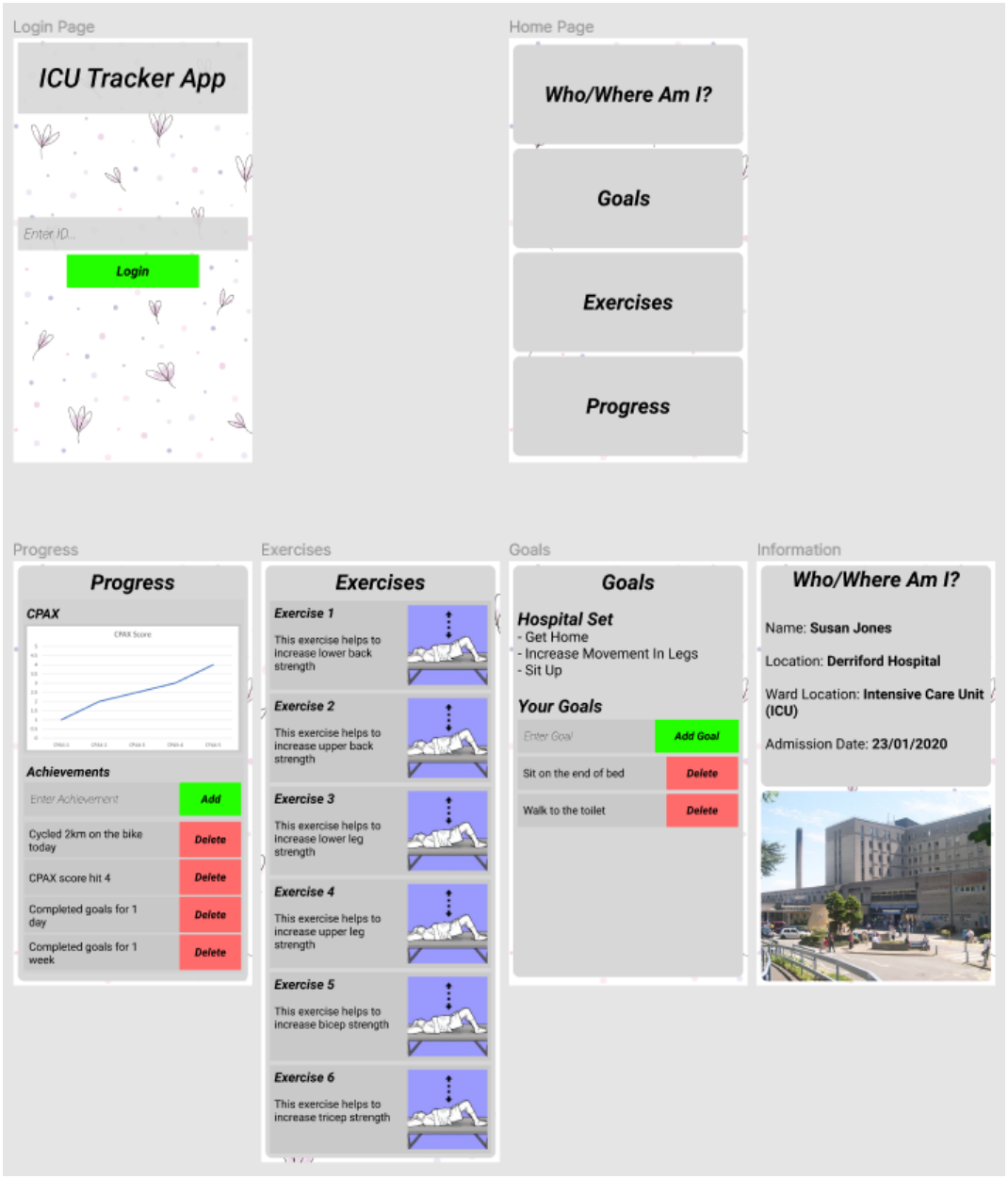


Figure 6. Storyboard for the main pages.

For the interactive part of the storyboard, this was done entirely through the browser. This made it easier to test the front-end as I didn't need to have somebody use my specific computer to test it, which would have been a major problem due to COVID. The interactive storyboard was also intuitive to use, which again made it easier to test with. Figure 7 shows what the testing users saw when they were using the interactive storyboard.

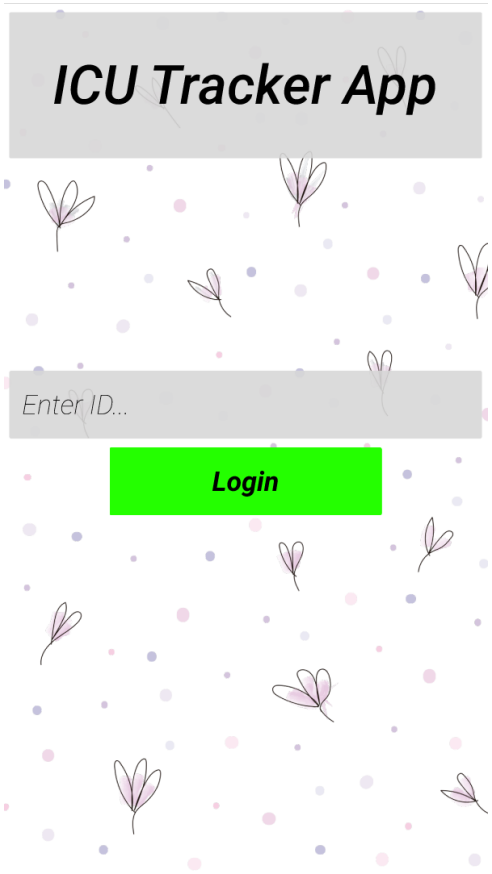


Figure 7. The login page when testing the storyboard.

7.7 Technologies

A number of technologies were used within my project to create the tracker app, all of which were researched previously, to make sure that they would work for the project in mind. A number of these I had used before, however a lot of them I had no prior experience with, so a lot of time was spent researching and performing exercises to learn how they work. A lot of the technologies chosen were due to compatibility and ease of use. For instance, Xamarin, SQL Server and ASP.Net Web API are created and supported by Microsoft, so have a lot of documentation on their use together.

The following sub-chapters detail each technology and why they were chosen.

7.7.1 SQL Server

SQL server was used as a database to store all the data for the app including the exercises, images and user information. This database was created upon the

previously made ERD which specified all the tables and formatting for the entries, which made creating the database relatively easy.

SQL Server was used mainly due to its compatibility, with it being developed and supported by Microsoft. This meant that there was a lot of online documentation, which made it a lot easier to develop with. SQL Server is also very widely supported which meant any problems that occurred while developing with it were easy to fix, as there are a multitude of online resources for help. The large amount of documentation also meant that moving the database in the future, which would be necessary for it to be deployed in the workplace, should be a lot easier than with other database technologies.

I also have prior experience using SQL Server, which made designing and creating the database a lot easier as I had developed with it several times before. Kumar, et al. (2014) describes a number of pros and cons of using SQL; I used this along with my own previous experience to form a table of its pros and cons (Table 6). This helped me to decide that SQL Server would be the best option for my app.

Table 6. Pros and cons to the use of SQL Server (Kumar et al., 2014)

Pros	**Cons**
Easy to create queries	Debugging can be tricky
Good range of data types	Hard to setup first time
Support for data migration	Can be costly
SQL language easy to learn and understand	Doesn't support some newer technologies
Widely supported and used	
Has a lot of documentation and online materials	
Scales well	
Easy to maintain	

7.7.2 ASP.NET Web API

The ASP.NET Web API is a framework built upon .NET which allows for the building of HTTP based services such as the API needed for this app (Microsoft, 2021). The API consists of a context file, model files and controller files which come together to specify the shape of the database and decide what data could be accessed or not.

As the ASP.NET Framework is developed by Microsoft, there was a lot of online documentation to help with the creation of the API. This also meant that there was a lot of documentation regarding the use of it with other Microsoft software, such as Xamarin, which I was using. This helped reduce development time as I had not previously used the framework so had to start from the ground up.

Another feature of ASP.NET Web API framework that made it easy to create an API was its integration with Visual Studio. This gave it features like building and deploying locally, which made it easier to test with compared to other API frameworks where I would likely have to manually reload. Using the framework also meant that I was able to create the database first, and then the ASP .NET Web API could create the models for itself. This massively reduces development time as creating the models is typically just translating the data from one language to

another. After the Framework had created a basic template from the database, the controllers needed to be made, allowing for the accessing of specified data through the API.

7.7.3 Insomnia

Insomnia is an API design platform that allows for quick and easy testing of API functions. It was used a lot within my project to test API functions to ensure that they were returning the right information. It was also used to check that it accepted the correct data, stored it appropriately and returned the correct codes. Figure 8 shows all of the calls that I used in insomnia to test the API.

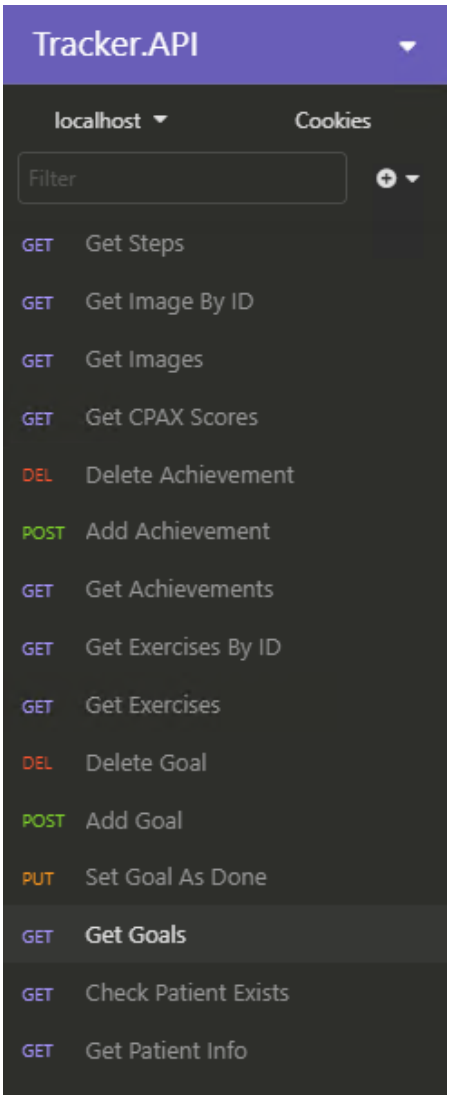


Figure 8. API calls within Insomnia.

These API calls ranged from getting data from the database, to submitting new data. Figure 9 shows the JSON for adding an achievement to the database using the API, along with the URL that it is using to do this. On the right of the window, the return code is also shown, which in this case was 200, which meant that the data was successfully added.

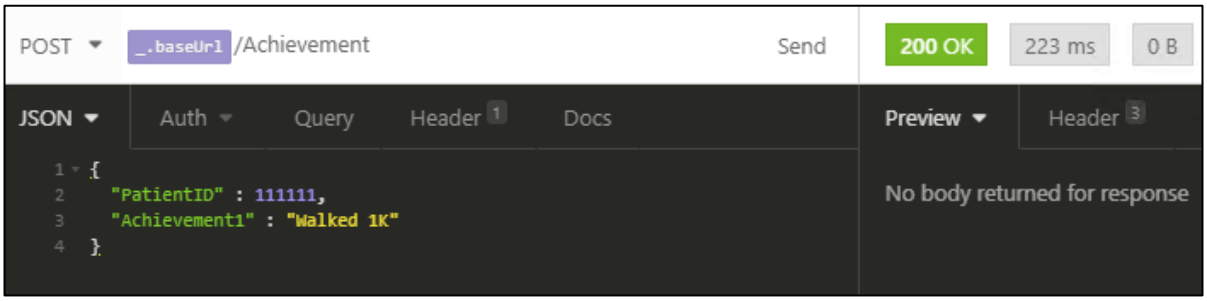


Figure 9. API call to add an achievement using Insomnia. The return code after sending is shown on the right.

7.7.4 Xamarin

Xamarin is an open-source platform created by Microsoft based upon the .NET framework which uses XAML and C# to allow for cross-platform apps to be created for Android, IOS, MacOS and Windows from one central codebase (Hermes, 2015).

Xamarin also handles the emulation of an android device natively on Windows alongside the Android SDK. This meant that the creation of pages and features was quick as it allowed me to create and test the app locally, without having to export the files to an android device. Xamarin also has a XAML hot reload feature which meant that I was able to quickly change something on the front-end code and it would be updated in the android emulator without having to reload the OS or rebuild the app. This greatly sped up development time of the front-end as it was easy to make small adjustments and see their effects in real time.

Xamarin was used inside of Visual Studio to code the app, which allowed for the automatic download of extensions and updates. It also has a number of built-in extensions such as Xamarin Forms which was used to quickly create the front-end of the app without having to build it in the phone specific language.

Xamarin allows for the changing of the files within each of the native OS's, in this case android and IOS, as well as the native Xamarin project. This meant that the app could be mostly developed in Xamarin, but for device specific functions such as where images are stored, these could be placed manually within the app files of the OS. Whilst not perfect, this was easy to do and other frameworks for Android and IOS such as React Native also manage images and videos this way too.

7.7.5 IIS

IIS is a function build directly into Windows that allowed me to host my ASP.Net Web API locally, on my home computer. This made it easy to develop and then test the API as it only had to be deployed onto the computer. This was further made easier by the fact that I was developing on Visual Studio, which meant that it could automatically build and debug the API straight to IIS, with both Visual Studio and IIS being created by Microsoft.

If the app were to be deployed publicly and used within the ICU, then the API could easily be moved onto a cloud platform such as Azure to allow it to be accessed from anywhere, however for local testing, IIS was the most suitable.

Figure 10 shows the IIS manager with the Tracker API running on it. With the IIS manager you can control all the APIs on the machine with basic functions such as removing, renaming and starting or stopping the API.

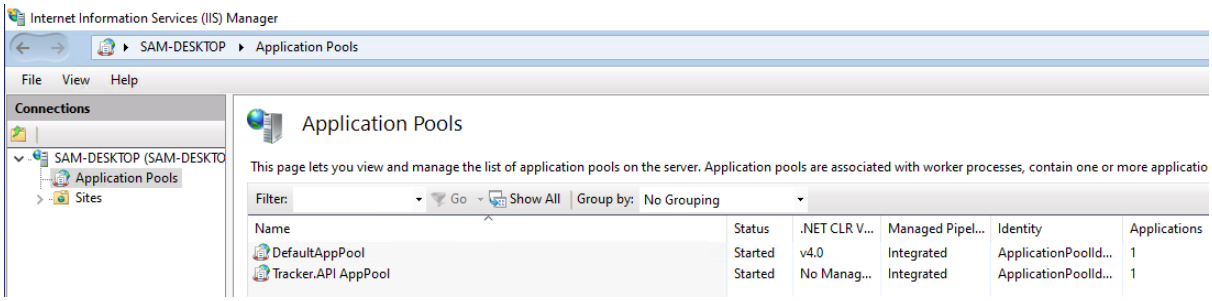


Figure 10. IIS Manager in Windows 10 with the Tracker API currently running.

7.7.6 Third-Party Technologies

A number of third-party libraries were used within Xamarin to allow for extra functionality which isn't natively supported. These libraries are listed and explained in detail in the following section.

Microcharts was a library used to be able to display charts inside of Xamarin, as it had no native library to handle diagrams or charts. Microcharts had a good amount of documentation and examples, which made it easy to be able to add it to my app and get it looking natural to the overall aesthetic (Xamarin Blog, 2021). There were some minor issues with the axis labelling and sizing of the graph, however the overall feature set of Microcharts meant that it was one of the only libraries capable of displaying the data the way I wanted.

Flurl was another library used in my app. This was used to connect the app to the API, handling the HTML requests to the API as well as the storing of the data once it had been received. Flurl was fundamental in the building of the app as without it, the API calling process would have taken a lot longer to develop and would have been much more complex (Menier, 2021). Flurl has a website with documentation, however it isn't very widely used so was difficult to setup and understand how to connect to the database. However, once the first few API calls had been created, it became a lot easier to use, with the code following a lot of the same patterns.

Loading was another third-party library used. This was used as Xamarin has no way to display to the user that an item is currently loading or unavailable. Loading allows me to block the screen so that the user isn't able to interact with the app while data is loading. This stopped the user from being able to change data which may cause the app to stop working or crash. Loading also has a visual element which notifies the user that an item on the page is loading which is also extremely useful as otherwise it would seem like the app has crashed or not responding.

The last library used within my app was FFImageLoading, which was used to display the GIFs for the exercise steps. This was used as while Xamarin does natively support the loading of GIFs, this doesn't work for my app, as the GIF is loaded after the page has, which the Xamarin library does not support. The FFImageLoading library allowed me load the GIF from the database and then display it on the front

end for the user to follow. The FFImageLoading library also allowed me to put a placeholder GIF in place so that if the GIF from the database didn't load, then a separate one could be used. For my app, I used a loading GIF as this placeholder to notify the user that the actual exercise step GIF had not loaded yet.

8 Development

For development of the app, sprints were used to split the load and make it more manageable. These lasted for 2 weeks each and consisted of tasks to do for that fortnight, with each task having their own individual due date and description.

Sprints were used to help split the load and keep development of the app on track throughout the year, each of which will be explained in this section.

8.1 Project Initialisation

Before any development started on the application, it was important to plan everything to ensure that the project was both feasible and any problems could be mitigated as early as possible. The planning documents include designs for the database and the API as well as front and back-end of the app itself.

To begin the planning process, Trello was used to create a backlog of items to understand what needed to be done to get the project underway. These consisted of planning documents and research to be completed as well as a rough outline of what the app would need to be functional.

These planning documents were used as a frame to base the development of the app from, which made it a lot easier to create as it meant I already knew what had to be created and how. The ERD, UML and Storyboard were created in this time as well as the research into current fitness apps and usability of apps.

Planning was also done on the technologies to be used to develop the app itself. Especially for an app, it is important that the right framework is used for the right job, which in my case was Xamarin. This made it easier for me as I knew Xamarin contained all of the features needed to be able to implement the ideas that I wanted, as I had spent time researching each main framework. From this SQL Server and ASP.NET Web API were chosen to accompany Xamarin because of their documentation and compatibility, due all three of them being produced and supported by Microsoft.

8.2 Sprint Planning

After the planning documents had been made, the sprints were created to lay out all of the tasks that needed to be done to get the app to a working condition. These also ensured that everything would be completed by certain dates, which was especially important with there being a strict deadline.

To start creating the sprints, I used the backlog made in the project initialisation section and expanded these out, breaking them down into smaller tasks. From there, I made fortnightly sections and assigned each task to a sprint, estimating how long each would take and looking at what order each part needed to be done in.

8.3 Sprint 1

Sprint 1 consisted of gaining front-end design feedback and subsequently starting to develop the front-end of the app.

The first item in sprint 1 was to finish and gain feedback on the front-end design of the app done in Figma. This was done in-person, with a document being created to note down the feedback that I got from the interactive storyboard. After gaining this feedback, the storyboard was quickly altered with these in mind which allowed me to make a final storyboard for the app (Figure 6).

The rest of the sprint consisted of starting development of the app. This included creating the project file structure and documents as well as creating the first few main pages of the app.

To create the app with Visual Studio, I used the “Mobile App (Xamarin Forms)” Template which allowed me to be able to develop the app for both IOS and Android simultaneously. From there, it automatically created the basic file structure, the files and built the environments for each OS. Following this, the front-end for the login, home, information, exercise and goals pages were created. This included the main functions such as navigation buttons and inputs that were necessary.

8.4 Sprint 2

Sprint 2 of development consisted of the creation of the database and the API. These were both fundamental to the inner workings of the app, so it was essential that these were created early on so that all of the features of the app could start to be implemented.

To begin creating the database, SQL Server was first installed onto my computer. This included SQL Server Management Studio (SMSS) which comes preinstalled with SQL Server and is officially supported which meant I could use to create and run queries for the database.

When creating the database, I stored all the queries inside of an SQL file, which would make it easier to recreate the database if necessary. It also meant that I could see all of the columns with their data types easily from one single file. This same SQL file was also used to store all the queries for entering test data into the database. With all of these queries I could recreate the database, including the data by running a single file which was extremely useful. This SQL file was used multiple times throughout development to reset the data for testing the app.

Following creation of the database, the Web API was able to be created. To start with this, the “Web API” project type was chosen inside of Visual Studio, this setup the basic files and file structure automatically for me. From there I used a migration, which is a feature within the ASP.Net Web API which allowed the API to connect to my previously created database and automatically build the models from it. Figure 11 shows the command used to be able to retrieve the tables from the database and generate a series of models from them. The migration also generated a context file, which in my case was called “TrackerAppContext”. This contained all of the

relationships between the each data object, so that the API could recognize what data can be entered or removed.

```
> dotnet ef dbcontext scaffold
"Server=.\;Database=TrackerApp;Trusted_Connection=True;"
Microsoft.EntityFrameworkCore.SqlServer -o Model
```

Figure 11. Migration command for retrieving models from database.

After creation of the models and context file, the controllers had to be made. These were used to control what data went in and out of the database, which was important in terms of both security and app functionality. Figure 12 shows the code for a post action inside of the “Achievement” controller which adds a new achievement to the database. Each action inside of the controllers uses the models and database settings in the context file to be able to manipulate the database to create a fully functional API. By the end of development there was a total of 7 controllers as seen in Figure 13.

```
[HttpPost]
0 references
public IActionResult Post([FromBody] Achievement achievement)
{
    TrackerAppContext.Achievements.Add(achievement);

    TrackerAppContext.SaveChanges();

    return Ok();
}
```

Figure 12. Post action to add a new achievement inside of the “Achievement” controller.

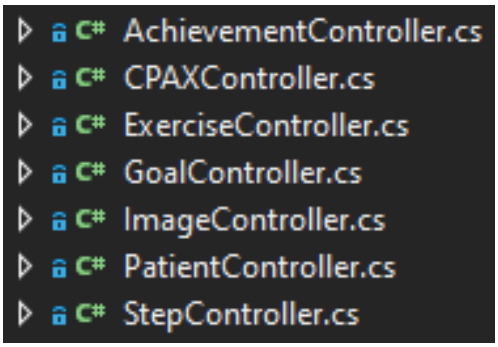


Figure 13. The controllers for the API.

After creating the API, the last task I completed in Sprint 2 was the connecting of the API to the app. To do this I used Flurl (7.7.6) which is a third-party library I used to create HTTP requests to communicate with the API. Figure 14 shows one such call where all of the goals for the user are requested from the API and then stored in the goal model in the app. At this point in the project, this data was not used for anything, however in Sprint 3 this data is connected to the front-end to create the basic functionalities of the app.

```
List<goal> HospitalData = await
URL.AppendPathSegment("Goal").AppendPathSegment(ID).AppendPathSegme
nt(true).GetJsonAsync<List<goal>>();
```

Figure 14. API call to get the goals for the user.

8.5 Sprint 3

The first item of Sprint 3 was to connect the data collected from the API with Flurl to the front-end of the app. Assigning of the data to their appropriate places on the front-end varied depending on what the data was and what formatting needed to be done to make it fit the storyboard.

For most of the pieces of data, this consisted of getting the stored value that was retrieved earlier with the API and then assigning it to text on the front-end of the app. However, this was not the same for all of them. The following section details the sets of data where more advanced data manipulation had to be done.

The first place where I had to do some manipulation is when I developed the graph for the progress page of the app. The first thing I did was get the 6 most recent months' worth of data and then store these in an array. This was due to the fact that the graph library that I used was not dynamic, so could only have a set amount of data. In my case I chose 6, which I thought would show a good picture of the user's recent CPAX scores. From here, I converted the month from numerical to text, so that it would be more readable for the user on the graph axis. Finally, I created the chart with the libraries code, using the array as the data and changing a few settings such as increasing the font size and axis colour. Figure 15 shows the finished chart with some test data to ensure it was working correctly.

Figure 15. Finished CPAX Score graph.

Another place where data manipulation was involved with the displaying of data was with both the achievements and goals. For both of these, they were put inline with a button, corresponding to that data's delete function. As XAML handles the displaying of data differently to other languages such as HTML, this meant having to use a grid

system to be able to group these. For these pages I used each row in the grid to show a different goal or achievement and used three columns to display the data, and their respective actions. To display these, a loop is used to go through each of the items and then create a new row in the grid with the encompassing data. This allows there to be any number of achievements or goals, rather than only a set amount. I also used a scroll section for these pages which allows the user to scroll down and view all of them, no matter how many there are. The last feature I had to implement for these was the delete and completed buttons. For these, I had to use the ClassId feature within Xamarin to associate the ID of the data with the button. This meant that when the user pressed the button, the data in the database with the ID of the button would be changed or removed. This allowed the buttons to use the same function to complete the action and work with any data, as the only other way to do it would be to only have a set amount of delete or completed buttons.

The last major place where the displaying of data differed is through the displaying of the exercises and exercise steps. For the exercises, these were each displayed in a tab with their respective text and image. As the images for the exercises were stored as blobs in the SQL server, this meant converting it back to an image so that it could be displayed for the user. This was done through the “ImageSource.FromStream” which converted the image and allowed me to display it on the front-end for the user. This was also used inside the steps for the exercise, as this too has the image for the exercise displayed within it.

As well as the reading and displaying of data, this sprint also included the development of the sending of data too. This was done a number of times throughout the app such as the user adding a new goal or logging into the app.

When logging into the app, the data entered by the user had to first be checked for invalid inputs and formatting issues, which in my case wasn’t a problem as both IOS and Android have a way to only allow numerical values to be entered (Figure 16). After this, the network connection was checked, as otherwise there would be no way for the user to be able to receive any of the data for their account. Finally, the input was sent to the API which checked to see if that account does exist and if it did, then it logged them in and stored the account number on the device for future API calls.

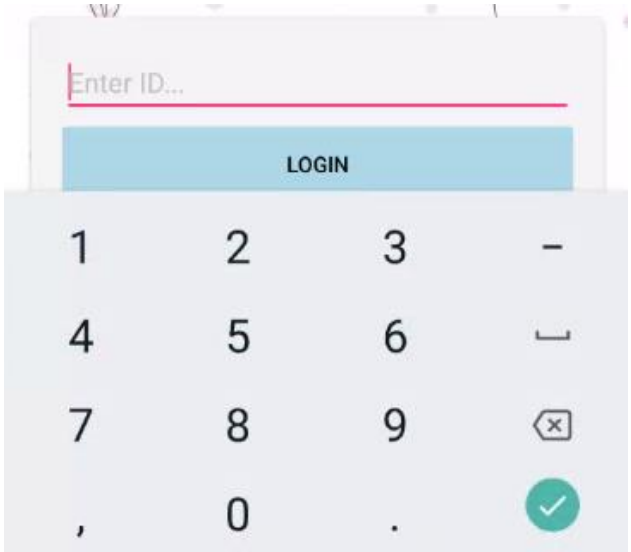


Figure 16. Logging in keyboard on Android.

Another place in the app where this happens is with the entering of new achievements and goals. These both have a section where the user can input text and then add it to the database as a new achievement/goal (Figure 17). As with the login section, the inputs on these also have some minor validation to check that the input isn't empty, and they aren't attempting to submit nothing as a new item. This also has an error linked to it, so the user is aware when they are making a mistake. Upon the user pressing the add button, an event is triggered which contacts the API with the user submitted data and attempts to add it to the database. If the data is successfully submitted, then the user is shown a "successful dialog" and the data on the page is reloaded, with the new data being shown to the user.

Figure 17. Adding a goal section.

At the end of the third sprint, most of the application had been completed, with just some minor tweaks having to be made to ensure the app was working fully and to add extra accessibility features.

8.6 Sprint 4

Sprint 4 consisted of the finishing of the main app as well as research into accessibility and user testing.

The research for the app was done on both the accessibility of the app as well as looking at the features of other fitness-based apps that could enhance my own. The outcome of this research was to make some minor changes to the colours and font sizes, to make it more accessible to those with sight difficulties. These are looked at in more detail in chapter 6.

Some minor additions were also added to the app in sprint 4, such as the loading library which helped to increase usability (Chapter 7.7.5).

User testing of the main app was also done at this stage. This was done through a series of tests that were created to allow the user to go through the whole app and give their thoughts on how it looked and worked, and to give feedback on these (Chapter 9.1).

8.7 Sprint 5

Sprint 5 consisted mostly of minor changes to the app and implementing changes brought up by users in the user testing. There were also some bugs that had been found through the user testing that had to be resolved too. This consisted of fixing some inputs to not allow all values and ensuring that all of the data buttons worked fully and as intended.

Another task done in this sprint was the removal of some of the extra features that were looked at in the beginning of the project. These Items were removed at this stage due to time constraints and them making the app unnecessarily complex.

8.8 Sprint 6

Sprint 6 was the last sprint in development of the app. This sprint included some final testing, bug fixes and changes as well as the adding of the final exercise data.

In sprint 6 I was provided with all the exercises that needed to be added to the app in the shape of a JSON file .This had to be converted to fit my tables and columns so that it could be shown properly within the app. This consisted of converting the JSON file to a SQL statement and then replacing the field names with the data names in my database.

Some final user testing was conducted on the app in this sprint as it was now fully functional and all the features had been implemented. This used the same set of tests shown in chapter 8.6 and detailed in chapter 9.1.

Before the app was fully completed, the app was thoroughly checked for any last-minute bugs and problems to ensure it was working as expected. Following this, the app was set to “release” mode and the IOS and Android apps could be built, allowing it to be installed on any devices necessary.

Finally, the app was tested on several Android devices to make sure it was working well natively, and not just on the PC. Testing on IOS was a big problem here as I didn’t have access to any Apple Devices and getting hold of one during COVID was very difficult. This led me to not being able to test the app on the platform, which would need to be done if the app was to be released in a business environment.

9 Testing

9.1 User Testing

User testing was completed a few times throughout development, most notably in sprints 4 and 6.

User testing consisted of 9 tests for the user to complete to be able to check and give feedback on the main features of the app (Figure 18). The user testing also helped me grasp how the average user would navigate through the app, allowing me to make changes to make it easier.

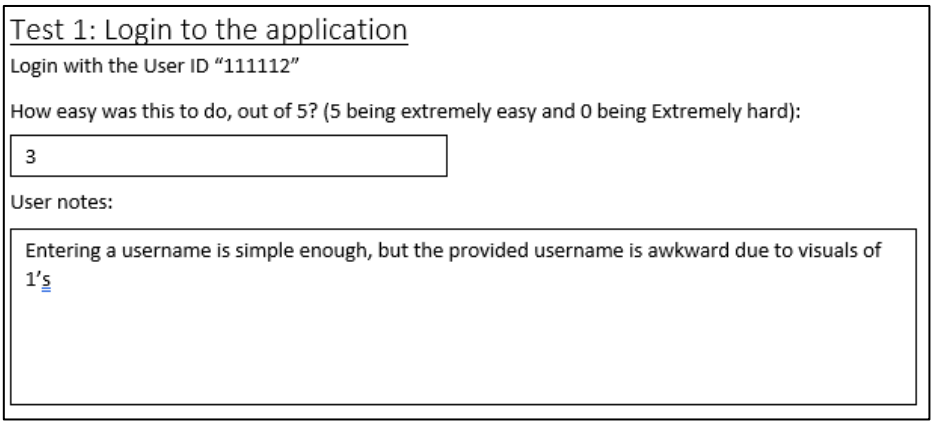
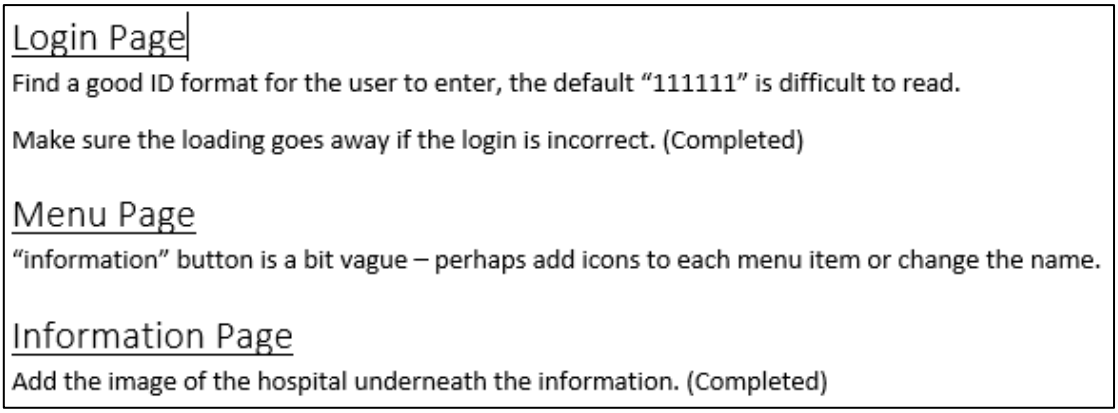


Figure 18. The first test in my feedback document, completed by a user.

Whilst testing the app, COVID played a big part as it meant that I was unable to test the app on a large amount of people, which would have been better for the development of the app. This was made harder by the fact that the app was unable to be tested remotely due to the use of a local API and database. This meant that only a few people were able to test my app, I did however gain some very valuable feedback from their inputs.

From the user testing I created a document of notes, looking at what I could change based on their feedback. The majority of these were improvement of life changes such as the moving of buttons and rewording of certain items. Some bugs were also found during user testing, like the achievement add button, which was not fully working. In my notes, I gave each item a tag to work out which were items that needed to be changed and which were optional (Figure 19).


Login Page

Find a good ID format for the user to enter, the default “111111” is difficult to read.

Make sure the loading goes away if the login is incorrect. (Completed)

Menu Page

“information” button is a bit vague – perhaps add icons to each menu item or change the name.

Information Page

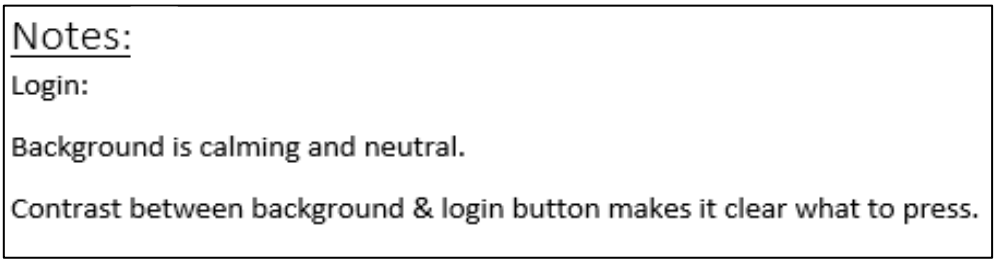
Add the image of the hospital underneath the information. (Completed)

Figure 19. First three pages notes with their respective feedback collected from user testing.

9.2 UX Testing

UX testing was completed both on the interactive storyboard as well as once the front-end of the app had been fully developed. This was especially important as the app is being used within the ICU of a hospital, so many users of the app will likely have some sort of physical disability.

For the UX testing, much like with the user testing, only a small amount of people were able to test the storyboard of my app, however I did get some very vital feedback (Figure 20). To test the storyboard, I had the user navigate through it and give feedback on each of the pages as well as the navigation throughout the app. This testing resulted in me changing many parts of the app such as the colours used and the size of the buttons and made the app much more usable.


Notes:

Login:

Background is calming and neutral.

Contrast between background & login button makes it clear what to press.

Figure 20. Notes made from storyboard feedback.

9.3 Insomnia

As described in chapter 7.8.3, Insomnia is a program I used to be able to test the functionality of my API and to ensure that it was working correctly. A number of tests were created alongside the API during the development so that I could create a section and then instantly test it, before moving onto the next one. This was incredibly useful for me, especially having never developed an API with ASP.NET before.

Figure 9 shows an example of one of the tests created to check the adding an achievement function.

10 Project Post-Mortem

10.1 What Went Well

In general, development of the app went well, having no major issues throughout the development of the app. All the planning and planning documents went smoothly in assisting the development and were extremely useful in pre-empting any problems or issue I might have run into.

The use of the interactive storyboard also went very well as it meant that I was able to get feedback on the front-end early on, allowing me to make any changes before development had fully started.

Development of the front end also went very well, as although I had to spend time learning the basics of XAML, there were no major setbacks. This was likely due to XAMLs similarities with HTML which made it intuitive to learn and develop with.

10.2 What Didn't Go Well

Development of the API and backend in general was a struggle as I hadn't used Xamarin or ASP.Net Web API before so it took a lot longer to develop then it should have. If a similar app were to be made again, development of the API should be a lot quicker due to me now having knowledge of how it all works. However, with this project it was a big delay to the development of the app that could have been spent implementing optional features.

Testing of the app was difficult to do due to COVID, which meant that in person testing was not possible, so an alternative method had to be found. This was made harder by the fact that the API and database were stored locally, meaning users testing the app would have to install a VPN to connect to these as well as the app itself. I did, however, manage to get some feedback on the storyboard at the beginning of development, the front-end design and the final app but more testing would have been extremely helpful.

10.3 What Would Be Changed?

While the project went well as a whole, there are also changes that I would have liked to have made if I were to create the app again or had more time to develop it.

Extra settings to help with the accessibility of the app would be a good addition. Allowing the user to change the colours, or the font size could be extremely helpful for some users, especially in the ICU.

However, with the limited previous experience with the technologies these were hard to implement, and further knowledge of the technologies would be needed to implement a number of them. If I were to create the app again, the whole process would be a lot quicker as I now have a lot more knowledge of the technologies used.

11 Conclusion

The aim of the project was to develop an app for the ICU that could be used by patients to track their progress through the ICU. This was completed with the app being fully functional and meeting all of the requirements set out at the start of the project. The final app was as planned and included the features that were expected as well as being completed on time with help from time management technologies and techniques such as Agile and Trello.

The app contains a lot of the accessibility features that I would expect for a modern app, so I think that the app could be used successfully in the ICU and would provide a useful tool for both doctors and patients.

12 References

Agile Alliance., 2021. What is a Backlog?. Available at: <https://www.agilealliance.org/glossary/backlog> [Accessed 12 May 2021].

Atlassian., 2021. What is Agile?. Available at: <https://www.atlassian.com/agile> [Accessed 12 May 2021].

Edwards, L., 2018. Data Protection: Enter the General Data Protection Regulation. Oxford: Hart Publishing, 1, pp.90-145.

Hermes, D., 2015. Xamarin Mobile Application Development. Berkley CA: Apress, 1, pp.1-2.

Kumar, R., Gupta, N., Charu, S., Bansal, S., Yadav, K., 2014. Comparison of SQL with HiveQL. International Journal for Research in Technological Studies, 1 (9), pp. 1439-2348.

Menier, T., 2021. Flurl. Available at: <https://flurl.dev/> [Accessed 12 May 2021].

Microsoft., 2021. ASP.NET Web APIs. Available at: <https://dotnet.microsoft.com/apps/aspnet/apis> [Accessed 12 May 2021].

Stiller, K., 2013. Physiotherapy in Intensive Care. Chest, 144(3), pp.825-847.

Xamarin Blog., 2021. Microcharts: Elegant Cross-Platform Charts for Every App. Available at: <https://devblogs.microsoft.com/xamarin/microcharts-elegant-cross-platform-charts-for-any-app/> [Accessed 12 May 2021].