A picture containing text

Description automatically generated

School of Electronics, Electrical Engineering and Computer Science

CSC3069 Software Engineering Enterprise Project Team Report

SWIFT Stroke Rehabilitation App

Michal Guzy

Sarah McMahon

Thomas Reid

Sophie Young

15th December 2022

**SCHOOL OF ELECTRONICS, ELECTRICAL ENGINEERING and COMPUTER SCIENCE**

**CSC3069 – SOFTWARE ENGINEERING ENTERPRISE PROJECT**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Declaration of Academic Integrity**    Before signing the declaration below please check that the submission:     1. Has a full bibliography attached laid out according to the guidelines specified in the CSC3069 Module Handbook and in the provided MS Word Template. 2. Contains full acknowledgement of all secondary sources used (paper-based and electronic) 3. The page number is within the specified page range (min 50 max 60), excluding bibliography and   Appendixes   1. Is clearly presented and proof-read, 2. Is submitted on, or before, the specified or agreed due date. Late submissions will only be accepted in exceptional circumstances or where a deferment has been granted in advance.     **We declare that we have read both the University and the School of Electronics, Electrical Engineering and Computer Science guidelines on plagiarism -**  **https://www.qub.ac.uk/directorates/sgc/learning/LearningResources/Plagiarism/ - and that the attached submission is our own original work. No part of it has been submitted for any other assignment and we have acknowledged in our notes and bibliography all written and electronic sources used.**    *Each Team Members Signature* *Date of Submission*       |  |  |  | | --- | --- | --- | | **Student Name** | **Student Number** | **Signature** | | Michal Guzy | 4026605 | Michal Guzy | | Sarah McMahon | 40257428 | Sarah McMahon | | Sophie Young | 40257097 | Sophie Young | | Thomas Reid | 40263793 | Thomas Reid |     A signed and completed declaration sheet must be included on the second page of the submission of the Software Engineering dissertation submitted for assessment.  Work submitted without a completed declaration sheet will **NOT** be marked. |

# Abstract

With the number of stroke survivors on the rise, there is increasing pressure on the health service to meet the demands for rehabilitation therapy. Dexterity is often impaired after stroke, and hinders the ability to use the weaker upper extremity for daily activities [1]. With the omnipresence of smartphones and mobile applications, interest has grown in using this technology for the purposes of providing stroke rehabilitation. Despite this interest, commercially available stroke rehabilitation apps are scarce, and some are not released for public use.

The Swift mobile application aims to be a rehabilitation tool to monitor an individual’s improvement in hand-eye-coordination via a games-based assessment. The intended beneficiaries of this application are those who have previously had a stroke and are in the process of rehabilitation. Results received from the game will allow an individual to track their progress over time, which can be relayed to their medical professional for further analysis to help shape their rehabilitation plan.

# Acknowledgements

Thanks to John Williamson, Computing Science, University of Glasgow for giving permission for EEECS to use and adapt their project report template.

Thanks to Dr Charles Gillan, Dr Richard Gault, and Dr Moira Watson, Queen’s University Belfast for the supervision and guidance of this project.

# Table of Contents

[Abstract 3](#_Toc121998693)

[Acknowledgements 4](#_Toc121998694)

[List of Figures 8](#_Toc121998695)

[List of Tables 9](#_Toc121998696)

[1 Introduction 1](#_Toc121998697)

[1.3.1 End User 3](#_Toc121998698)

[1.3.2 Medical Professional 3](#_Toc121998699)

[2 Requirements 4](#_Toc121998700)

[2.1 Requirements Elicitation 4](#_Toc121998701)

[2.2 Functional Requirements 4](#_Toc121998702)

[2. 3 Non-Functional Requirements 5](#_Toc121998703)

[2.4 User Stories 7](#_Toc121998704)

[3 Design 8](#_Toc121998705)

[3.1 Overall System Architecture 8](#_Toc121998706)

[3.2 Front End Design 9](#_Toc121998707)

[3.2.2 User Interface 9](#_Toc121998708)

[3.3 Game Design 12](#_Toc121998709)

[3.4 Back End Design 13](#_Toc121998710)

[3.4.1 Overall Back End Architecture 13](#_Toc121998711)

[3.4.2 User Management Service Architecture 14](#_Toc121998712)

[3.4.3 Analysis/Calculation Services Architecture Design 15](#_Toc121998713)

[3.4.4 Communication 15](#_Toc121998714)

[3.4.5 Database Design 16](#_Toc121998715)

[4 Implementation 17](#_Toc121998716)

[4.1 Front End Implementation 17](#_Toc121998717)

[4.1.1 Front-End Libraries and Dependencies 18](#_Toc121998718)

[4.1.2 Hand- Eye Coordination Game 19](#_Toc121998719)

[4.1.3 Displaying Results Analysis 20](#_Toc121998720)

[4.2 Back-End 21](#_Toc121998721)

[4.2.1 Containerisation and Orchestration 21](#_Toc121998722)

[4.2.2 Communication 22](#_Toc121998723)

[4.2.3 External Containers Used for Implementation 23](#_Toc121998724)

[4.2.4 User Management Service 23](#_Toc121998725)

[4.2.5 Results Analysis 25](#_Toc121998726)

[4.3 Cyber Security Considerations 26](#_Toc121998727)

[4.3.1 SQL Injection 26](#_Toc121998728)

[4.3.2 Security for Communication 27](#_Toc121998729)

[4.3.4 Security for Authentication and Password Storage 27](#_Toc121998730)

[5 Software Testing 29](#_Toc121998731)

[5.1 Testing Strategy 29](#_Toc121998732)

[5.2 Debugging and Manual Testing During Development 29](#_Toc121998733)

[5.3 Unit Testing 30](#_Toc121998734)

[5.3.1 Front-End Unit Testing 30](#_Toc121998735)

[5.3.2 Python Analysis Services Unit Testing 30](#_Toc121998736)

[5.3.3 C# User Management Service Unit Testing 31](#_Toc121998737)

[5.4 Integration Testing 32](#_Toc121998738)

[5.5 Acceptance Testing 33](#_Toc121998739)

[6 User Evaluation 35](#_Toc121998740)

[6.1 Alpha Testing 35](#_Toc121998741)

[6.2 Beta Testing 36](#_Toc121998742)

[6.3 System Evaluation 37](#_Toc121998743)

[6.3.1 Application Front-End 37](#_Toc121998744)

[6.3.2 Hand-Eye Coordination Game 37](#_Toc121998745)

[6.3.3 Application Back End 38](#_Toc121998746)

[7 Project Management 40](#_Toc121998747)

[7.1 Roadmap and Sprint Planning 40](#_Toc121998748)

[7.2 Tools and Techniques Utilised 41](#_Toc121998749)

[7.2.1 TimeTree 41](#_Toc121998750)

[7.2.2 Meetings, Microsoft Teams, WhatsApp and Miro 41](#_Toc121998751)

[7.2.3 SWOT Analysis 41](#_Toc121998752)

[7.2.4 Fortnightly Advisory Meetings 42](#_Toc121998753)

[7.2.5 GitLab 42](#_Toc121998754)

[7.2.6 Code Review 42](#_Toc121998755)

[7.2.7 Sprint Plans and Retrospectives: Azure DevOps 42](#_Toc121998756)

[7.3 Challenges Faced 42](#_Toc121998757)

[7.4 Risks 43](#_Toc121998758)

[8 Discussion and Conclusion 44](#_Toc121998759)

[8.1 Summary 44](#_Toc121998760)

[8.2 Impact 45](#_Toc121998761)

[8.2.1 Social Considerations 45](#_Toc121998762)

[8.2.2 Security & Privacy 45](#_Toc121998763)

[8.3 Maintenance 46](#_Toc121998764)

[8.4 Reflection 46](#_Toc121998765)

[8.5 Future Work 47](#_Toc121998766)

[8.5.1 Improvements to the Current Coordination Game 47](#_Toc121998767)

[8.5.2 Additional Rehabilitation Assessments 47](#_Toc121998768)

[8.6.3 Mitigating Risks that Currently Exist in the System 48](#_Toc121998769)

[8.6.4 Publishing to the App and Play Store 48](#_Toc121998770)

[8.6.5 Deploying to the Public Cloud Using Kubernetes 49](#_Toc121998771)

[Bibliography 49](#_Toc121998772)

[Bibliography 54](#_Toc121998773)

[Appendices 56](#_Toc121998774)

[Don’t include your source code in the appendices. It will be accessed via **Error! Bookmark not defined.**](#_Toc121998775)

[Appendix 1: Front-End Design Additional Wireframes 56](#_Toc121998776)

[Appendix 3: Additional Dependencies within Swift App 58](#_Toc121998777)

[Appendix 4: Additional Python Dependencies 59](#_Toc121998778)

[Appendix 5: Additional C# Dependencies 60](#_Toc121998779)

[Appendix 6: External Docker Image Versions 62](#_Toc121998780)

[Appendix 7: TimeTree Calendar 63](#_Toc121998781)

[Appendix 8: Acceptance Tests 63](#_Toc121998782)

[Appendix 9: Alpha Testing 69](#_Toc121998783)

[Appendix 10: Beta Testing 79](#_Toc121998784)

# List of Figures

[Figure 1: Overall System Architecture 8](#_Toc122000292)

[Figure 2: Front-End Navigation Flow Diagram 10](#_Toc122000293)

[Figure 3: Original Front-End Design Screenshots 10](#_Toc122000294)

[Figure 4: Revised Front-End Wireframes 11](#_Toc122000295)

[Figure 5: Swift API User Management Diagram 14](#_Toc122000296)

[Figure 6: Database Entity Relationship Diagram 16](#_Toc122000297)

[Figure 7: Pseudocode for Generation of Obstacle Positions 19](#_Toc122000298)

[Figure 8: Pseudocode for Calculation of User Accuracy 20](#_Toc122000299)

[Figure 9: Implementation of the Mediator and the CQRS Design Pattern [38] 24](#_Toc122000300)

[Figure 11: Preparing SQL Statements to Mitigate SQL Injection 27](#_Toc122000301)

[Figure 12: Validating a New User 28](#_Toc122000302)

[Figure 13: Example of Testing for Invalid Data in PyTest 31](#_Toc122000303)

[Figure 14: Example of Tests Passing for the Analysis Services 31](#_Toc122000304)

[Figure 15: Authentication Unit Tests Results 31](#_Toc122000305)

[Figure 16: Feature File for the Authentication Unit Test 32](#_Toc122000306)

[Figure 17: User Management Service Testing Results 33](#_Toc122000307)

# List of Tables

[Table 1: Functional Requirements 5](#_Toc122000396)

[Table 2: Non-Functional Requirements 6](#_Toc122000397)

[Table 3: User Stories 7](#_Toc122000398)

[Table 4: Summary of Acceptance Test Results 34](#_Toc122000399)

[Table 5: Survey Questions Used for Alpha Testing 36](#_Toc122000400)

# 1 Introduction

This chapter offers an insight into the problem domain of stroke rehabilitation and how the Swift app seeks to provide solutions to these challenges.

* 1. **Problem Area**

There are currently over 1.2 million stroke survivors in the UK, with many experiencing significant and long-term physical and psychological impacts. It is expected that the number of stroke survivors living with disability will increase by a third by 2035 [2]. With stroke survival rates improving, the focus now lies on how people can be supported following a stroke. Pressure on the NHS (National Health Service) is imminent to cope with the significant increase in demand for ongoing care and support following a stroke, as well as the economic costs associated with it [3].

Mild stroke accounts for 60% of all cases of stroke and, even though these individuals are considered not to have disabling functional deficits, they might experience motor, sensory, visual, speech or cognitive deficits. Studies have shown that stroke survivors have poorer hand-eye coordination, in terms of slower movement and reduced accuracy when using their affected hand [4]. These individuals are usually discharged home without further rehabilitation since they are expected to return to their premorbid status [1]. Stroke rehabilitation should be easily accessible and resource efficient to provide all stroke survivors with the opportunity to recover their functional abilities.

The first three months after a stroke are the most important for recovery and when individuals will see the most improvement, thus rehabilitative therapies should be started as early as possible to maximise functional recovery. Individuals need to practice many repetitions of movement at high intensity for efficient motor learning and neural plasticity to occur. However, individuals are engaged in treatment sessions for a very limited time each day in in-patient rehabilitation [1]. Self-directed training is a time-efficient and cost-effective way to maximise training time both during subacute rehabilitation and at home and has proved to be successful with favourable results for improved upper limb function in chronic stroke [5].

Games such as cards, pegs and blocks are commonly used in clinical practice to improve hand function, however these traditional treatment methods are often boring and not motivating and thus do not encourage intensive self-training. One study focusing on recovery from the stroke survivors’ perspective highlights the need to develop training strategies that match the needs and aspirations of stroke survivors and that place no time limits on recovery [6]. Training should therefore both be personalised and adjusted as individuals engage in therapy [7] to allow relevant and achievable goals to be set in conjunction with their physical or occupational therapist.

* 1. **Problem Analysis**

The Swift Rehabilitation Application addresses these issues by leveraging the use of mobile technology to provide personalised stroke therapy in the form of a games-based hand-eye coordination assessment. This offers a means of providing early and resource-efficient stroke rehabilitation that is inexpensive and accessible. The hand-eye coordination game is analogous to existing assessments carried out by medical professionals (see Section 3.3) and gives individuals an opportunity to continue their rehabilitation exercises in the comfort of their own home. This may help to transfer some of the responsibility of rehabilitation to the users themselves, thus possibly increasing their self-efficacy as well as improvement in dexterity and hand-eye coordination. Several studies on the use of tablet technology during stroke recovery indicate high patient satisfaction, with therapy independence and the convenience of being able to engage in therapy at home being specific positive aspects [8]. Through the gamification of standard therapeutic activities, Swift aims to make the patient rehabilitation experience more enjoyable and promote consistent therapy engagement. The application provides a platform to increase the variety and intensity of stroke therapy which can help to motivate individuals to keep up the consistent routine required for improvement.

Many hand-eye coordination game applications have been developed for the general population [9], but only a limited number might be successfully played by individuals with post-stroke motor impairments. The strength of the Swift application over other mobile-based therapies is the ability to monitor coordination progression, providing users with the reassurance that their rehabilitation plan is meeting their needs. Using the in-app email feature they can send a copy of game results to their medical professional, who is then able to use these results to make informed therapy adjustments.

The Dexteria mobile application has a similar concept to the Swift app and is recommended by the NHS for rehabilitating finger movement. Results from a clinical trial completed with this app showed statistically significant correlation between hand weakness and improvement in hand performance [10]. One of the advantages of Swift over Dexteria is the hand-eye coordination game and its ability to adapt the difficulty level based on a user’s previous results. This ensures therapy is personalised to the user’s needs and can be adjusted throughout their rehabilitation journey as required.

* 1. **Personas**

### 1.3.1 End User

David is a retired 55-year-old and lives alone. He owns a smartphone and regularly uses it to stay connected with family members, although he struggles with some of the more advanced smartphone features. He suffered from a stroke 2 months ago and has since struggled with arm weakness on his left side as well as a significant deterioration in his hand-eye coordination skills. He meets with his occupational therapist every few weeks, however he would like to be able to get progress reports more regularly to provide reassurance that his rehabilitation plan is effective, and that he is improving. The Swift Rehabilitation application provides him with a means to monitor his rehabilitation progress by playing the hand-eye coordination game daily and viewing his results from the last seven days. He is then able to send these results to his occupational therapist via the email feature within the app, or he can share them during his reviews. David finds the game engaging with the changing difficulty levels and feels motivated to continue with his rehabilitation plan when he sees his coordination results have improved.

### 1.3.2 Medical Professional

Alice is a 36-year-old occupational therapist who has been looking after David following his stroke. She has provided him with a rehabilitation plan, but it can be difficult for her to accurately track his progression as she only meets with him once every few weeks. The Swift Rehabilitation app provides David with a means to complete hand-eye coordination games on a regular basis, and the in-app email feature allows him to send these results to Alice. These results provide her with a detailed weekly view of David’s progress which can help to inform her of the adjustments which should be made to his rehabilitation plan.

# 2 Requirements

This section details the system requirements and elicitation process performed to derive them.

## 2.1 Requirements Elicitation

To begin the requirements elicitation process, an interview was conducted with a medical professional, to explain existing hand-eye coordination assessments carried out by physicians, and the metrics used to define coordination. The medical professional highlighted their desired features for the application from the perspective of a medic and discussed the needs of stroke survivors throughout the rehabilitation process. From this interview, a list of requirements was devised focusing on the development of a hand-eye coordination game and a means for users to view game results. These requirements were grouped into collections of related requirements and subsequently prioritised based on their significance to the end user and their importance for the operation of the application. Each non-functional requirement listed in Table 2 is related to a functional requirement with the corresponding ID in Table 1.

## 2.2 Functional Requirements

The functional requirements of the Swift app have adapted as the project has developed. Previously, focus had not been placed on the personalisation of the coordination game, which has now become a unique selling point of the app, with functional requirement 6.0 relating to this, and how the Swift app will adapt the difficulty level based on the user’s previous gameplay. Furthermore, functional requirement 12.0 has been introduced, as importance has been placed on the ability of the app to send coordination results to a medical professional; thus bridging the gap in time taken between in-person contact with a medical professional.

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Requirement** | **Area** | **Importance** |
| 1.0 | Provide the user with a functional graphical user interface | Front end | Essential |
| 2.0 | Present the user with options to register via Google or Swift | Front end | Essential |
| 3.0 | Store user credentials in the database | Back end | Essential |
| 4.0 | Present the user with the ability to log in to the application using a pre-registered email address and password, either through Google or Swift | Front end | Essential |
| 5.0 | Allow the user to log out of the application | Front end | Essential |
| 6.0 | Present the user with an interactive game to test their hand-eye coordination | Front end | Essential |
| 7.0 | Calculate the user’s accuracy and time taken on completion of the coordination game | Back end | Essential |
| 8.0 | Send user game results to the back end | Back end | Essential |
| 9.0 | Store user game results in the database | Back end | Essential |
| 10.0 | Display the user’s results from the hand eye coordination game | Front end | Essential |
| 11.0 | Allow the user to connect to the backend to access game results from the database | Back end | Essential |
| 12.0 | Allow the user to send their game results to a medical professional via an in-app email feature | Front end | Essential |

Table : Functional Requirements

## 2. 3 Non-Functional Requirements

The non-functional requirements of the Swift app have grown and adapted as the project has developed. In refining the non-functional requirements, the decision was made to replace the “practice run” game tutorial with a pop-up game tutorial (fulfilling non-functional requirement 6.1) that can be accessed at any time during gameplay, as this may be less confusing for the end user. Additional new non-functional requirements centralise around the back-end area, including containerisation of the back end using Docker (fulfilling non-functional requirement 11.1), and deploying services on the public cloud. Importance has also been placed on ensuring the end user is aware the Swift app is a non-diagnostic rehabilitation app and has clear understanding of how to play the coordination game and interpret the results (non-functional requirements 6.1, 10.1 and 10.2).

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Requirement** | **Area** | **Importance** |
| 1.1 | Allow the user to navigate through the application to any of the screens | Front end | Essential |
| 1.2 | Maintain a consistent layout for screens | Front end | Desirable |
| 1.3 | Ensure font size and buttons are large and easily readable for those with visual impairments | Front end | Desirable |
| 1.4 | Give the user the ability to change the main colour palette of the application to a “dark mode” style | Front end | Desirable |
| 1.5 | Provide accessibility to the Swift app on both iOS and Android platforms | Front End | Desirable |
| 3.1 | Store user credentials securely by hashing passwords | Back end | Desirable |
| 3.2 | Protect the database against attacks such as SQL Injection | Back end | Desirable |
| 4.1 | Authenticate user login details to ensure credentials are legitimate | Back end | Desirable |
| 5.2 | Implement secure key management | Back end | Desirable |
| 5.3 | Make use of JSON web tokens for user login and registration | Back end | Essential |
| 6.1 | Provide a tutorial to instruct the user on how to play the coordination game | Front end | Essential |
| 6.2 | Provide the user with a personalised difficulty level that adapts based on previous game results by decreasing or increasing the number of obstacles | Front end | Desirable |
| 8.1 & 9.1 | Implement HTTPS to provide secure communications between the front end and back end | Back end | Desirable |
| 10.1 | View previous scores via a results screen, with charts displaying overall metrics and progress over time | Front end | Essential |
| 10.2 | Provide justification and explain the overall metrics for the results | Front end | Essential |
| 11.1 | Containerise the back end using Docker containers | Back end | Desirable |
| 11.2 | Deploy services onto the public cloud | Back end | Desirable |
| 11.3 | Ensure secure connections between services using a messaging broker | Back end | Desirable |
| 12.1 | Provide a pre-written (yet still editable) email containing the user’s game results to send to a medical professional | Front end | Desirable |

Table : Non-Functional Requirements

## 2.4 User Stories

|  |  |  |
| --- | --- | --- |
| **Story** | **Persona** | **Description** |
|  | *As A:* | *I Want:* |
| 1 | End User | To be able to use the Swift application on my Android device |
| 2 | End User | To be able to use the Swift application on my iOS device |
| 3 | End User | To register for a Swift account using my Google account |
| 4 | End User | To register for a Swift account using my preferred email |
| 5 | End User | To login to the system with my pre-existing account |
| 6 | End User | Clear navigation throughout the app that takes me to the expected screen |
| 7 | End User | To be able to play the coordination game every day |
| 8 | End User | The coordination game to be simple as my hand-eye coordination skills are very limited |
| 9 | End User | The coordination game to be different every day so that it does not get repetitive |
| 10 | End User | To be able to see my results upon completion of the coordination game |
| 11 | End User | To be able to view my previous results from the coordination game |
| 12 | End User | To be able to receive feedback on my game results |
| 13 | End User | To be able to share my progress history with my doctor |
| 14 | End User | To be able to change the colour palette of the app |
| 15 | End User | To be able to view all text, buttons, and images clearly |
| 16 | End User | My password to be secure |
| 17 | End User | To be able to log out of the app |
| 18 | Medical Professional | To be able to view a patient’s results from the coordination game |

User stories were devised based on the personas outlined in Section 1.3. These were used to formulate the acceptance tests for front-end testing (see Section 5.5) to verify that the requirements had been met.

Table : User Stories

# 3 Design

The Swift Rehabilitation application is designed with the goal of satisfying the requirements defined in Chapter 2. Key design components were identified, and are described in Sections 3.1-3.4, along with a complete system architecture to demonstrate how they have been integrated together to shape the proposed Swift application.

## 3.1 Overall System Architecture

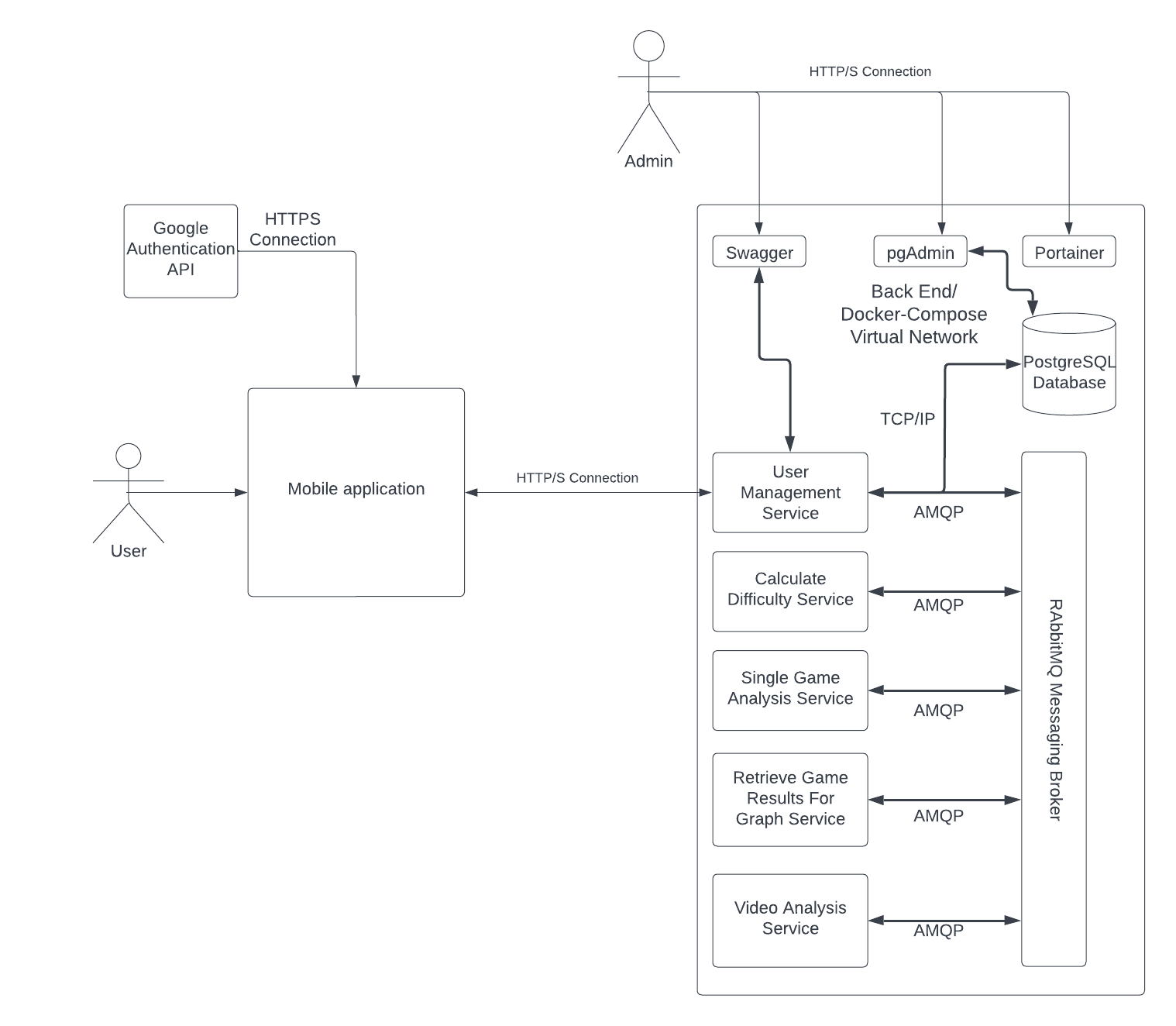


Figure : Overall System Architecture

Figure 1 depicts an abstraction of the overall system architecture. The front end of the system consists of the mobile application which makes calls to the Google Authentication API (Application Programming Interface) and the User Management Service.

The back end of the system consists of a containerised virtual network of services and management systems. The User Management Service acts as a reverse proxy, delegating any requests to the appropriate service via the messaging broker. This service also deals with processes including, but not limited to, storing and retrieving data from the database and authenticating users. The PgAdmin and Portainer management containers are used in development to check the data in the database is correct and that all containers are running successfully. The Swagger component is also used during development to manually test the endpoints on the backend.

## 3.2 Front End Design

The front-end design of the system plays a critical role in achieving the goal of the Swift app and providing the user with the best experience possible. The system must be able to provide the end user with a functioning UI that they can interact with through human computer interaction, to fulfil functional requirement 1.0, and non-functional requirements 1.1-1.5.

From the design of user personas, detailed in Section 1.3, and a focus on the typical end user being an elderly person who has suffered from a stroke and is currently in active recovery, the Swift app has been designed with that knowledge at the foreground, and places great emphasis on the user interface being understandable and interactive.

### 3.2.2 User Interface

The Swift app was designed with user experience at the forefront and aiming to make the UI as understandable as possible. The logo and splash screen of the Swift app were designed using Figma, a web application for interface design, which provided the best quality resolutions when designing, as it allowed for exact dimensions to be utilised [11].

Once logged in, a centralised “Home” Screen is shown, and from there, the user can navigate to and access different features of the app. A bottom tab navigator has been designed featuring key areas of the app (Home, Help, Settings), allowing for shortcut access to these screens, with understandable icons and large text.

As illustrated by Figure 2, an overall navigation flow chart of the system was created to delegate how each screen of the app is connected, and the flow of screens as the user interacts with the app. Navigation throughout Swift has been designed to be dynamic, and so all back buttons link to whichever screen flow path the user has taken; thus making navigating back and forward seamless.

Diagram, engineering drawing

Description automatically generated

Figure : Front-End Navigation Flow Diagram

The design of the front-end of the Swift app has undergone changes that have adapted with the direction of the project. Previously, as seen in Figure 3, the front end consisted of minimal screens, with skeleton functionality. The team felt there was limited identifiable brand structure or aesthetic, nor unique selling points in the design aspect. Navigation was not dynamic, and the app was not scalable for different device sizes.

Graphical user interface, application

Description automatically generatedGraphical user interface

Description automatically generated with medium confidence A picture containing text, electronics

Description automatically generated A picture containing graphical user interface

Description automatically generated

Figure : Original Front-End Design Screenshots

In order to align with the direction the Swift app would take, an overhaul of this front-end was decided upon. As can be seen in Figure 4, the new front-end focused on user experience, and initial designs were originally created using wireframes in Balsamiq [12], to grasp an understanding of what would later be implemented in development (additional wireframes can be seen in Appendix 1).

Graphical user interface, text, application

Description automatically generatedGraphical user interface, text, application

Description automatically generatedGraphical user interface, text, application

Description automatically generated

Figure : Revised Front-End Wireframes

With many revisions to the design, the consistent layout was maintained across all screens, to enhance the professional aesthetic of the app, and custom fonts are used for titles and other text, which reflects this unique and professional aspect. Further design choices include large titles and clearly readable text on each page, providing accessibility to those with visual impairments, and the Swift logo is displayed where appropriate to stay brand specific. All back buttons are in the same position, with a consistent colour scheme maintained throughout all screens, with this being chosen to maximise accessibility, and therefore is suitable for those with colour-blindness (see Appendix 2). These design decisions promote the Swift app being on-brand, professional, and user friendly.

As the target audience of the Swift app is an older demographic, it is fundamental that its purpose and content is appropriately conveyed to the end user. Due to this, the team placed great emphasis on designing features to be easily understandable, specifically the coordination game and results. A tutorial pop-up was designed to automatically load on the game screen, that the user must first read and confirm they understand, before progressing. The user can access this tutorial at any point during the game by clicking the clearly labelled “How to Play” button - this ensures the user can benefit as much as possible from the game. Furthermore, all results are clearly explained to the user, as each screen provides a detailed breakdown of each result. Further information on strokes has been designed to be easily accessed via the Help Screen, which a link to the NHS webpage dedicated to strokes [13].

Subtle design features such as user checks for certain navigation choices, in the form of an alert, heighten the user experience and accessibility. As the aim is to analyse hand-eye coordination, it must be considered that the app’s primary user may have weaker motor skills and may accidentally click a button. This design choice was applied to the logout button, as well as the back button on the coordination game screen. The email feature for sending and receiving results proves fundamental in bridging the gap between end user and medical professional, providing connection outside of medical rehabilitation appointments. This feature has been designed with a pre-written subject and body of the email, that can still be edited by the user, if necessary, to make this email function as easy and quick to use as possible. Prioritising social considerations, the app has been designed with extreme users in mind, as well as the typical end user – this accessibility has been addressed by ensuring the default colour scheme of the Swift app is suitable for colour-blindness. A dark mode feature has been designed too, that can be turned on and off with user preference.

## 3.3 Game Design

To begin the game design process, research was conducted into existing assessments for upper extremity function that are typically administered by trained physical or occupational therapists. The Fugl-Meyer assessment is widely used and internationally accepted and includes tests to assess coordination and speed [14]. One test from this assessment involves touching your finger to your nose repeatedly, with indications of weakness including inaccuracy in touching the nose and slow movements. The Nine Hole Peg Test [15] is commonly used as a timed, quantitative measure of fine manual dexterity, with scoring based on the number of seconds it takes for the patient to complete the test. This research, along with information received from the interview with a medical professional, helped to shape the design of the hand-eye coordination game, and indicated the metrics which should be used for providing a score.

The coordination game has been designed with a minimal user interface and straightforward instructions to ensure it is suitable for those of an older demographic who may have limited technical skills. To make sure that the game objective is clear, a tutorial pop-up screen is displayed before the game begins, and it can also be accessed throughout the game via the ‘How to Play’ button.

The aim of the game is to drag a circle through a set of obstacles to the centre of a target as quickly as possible. If the user hits an obstacle while dragging the circle, the game is reset, and the circle is regenerated at the starting point. The time taken is measured as the time from when the circle is first moved to when it is placed in the target. The accuracy is measured as the distance in pixels between the centre of the target, which is marked with a white cross, and the centre of the circle. The accuracy score decreases the further the circle is placed from the centre. With traditional therapeutic tools, it is difficult to train speed of motion and reaction time and almost impossible to objectively quantify performance for precise and objective evaluation and as well as for monitoring improvement [1]. By transferring rehabilitation assessments to a smartphone application, more accurate results can be recorded by tracking coordinates of finger movement across the screen. These results can then be stored for future reference to quantify an individual’s progression or regression.

Each time the game is played, the position of the obstacles and target will randomly change which is intended to make the game more engaging and feel less repetitive. The difficulty level of the game is adjusted each time based on the user’s previous game results. The difficulty value is used to determine the number of obstacles on screen and the distance between them. If a user has improved their score the difficulty will increase, meaning the number of obstacles will increase and the distance between them will decrease, and vice versa. This means that the game is personalised to each user and will adapt to their needs.

## 3.4 Back End Design

### 3.4.1 Overall Back End Architecture

The back end of the system is designed using the service-oriented architectural style. This allows the overall system to be more maintainable than a monolithic application by introducing loose coupling and high cohesion. Furthermore, service-oriented architecture (SOA) allows the system to be separated by enterprise scope, rather than application scope which is used in the microservices architectural style. Using application scope would drastically increase the amount of effort needed to implement the application and further complicate the maintenance of the system [16].

The microservices architecture was explored, however, this posed the risk of increased latency due to the increased number of connections between components. Additionally, each microservice would need its own communication protocol and API, but with SOA these services can communicate on a messaging broker [17].

### 3.4.2 User Management Service Architecture

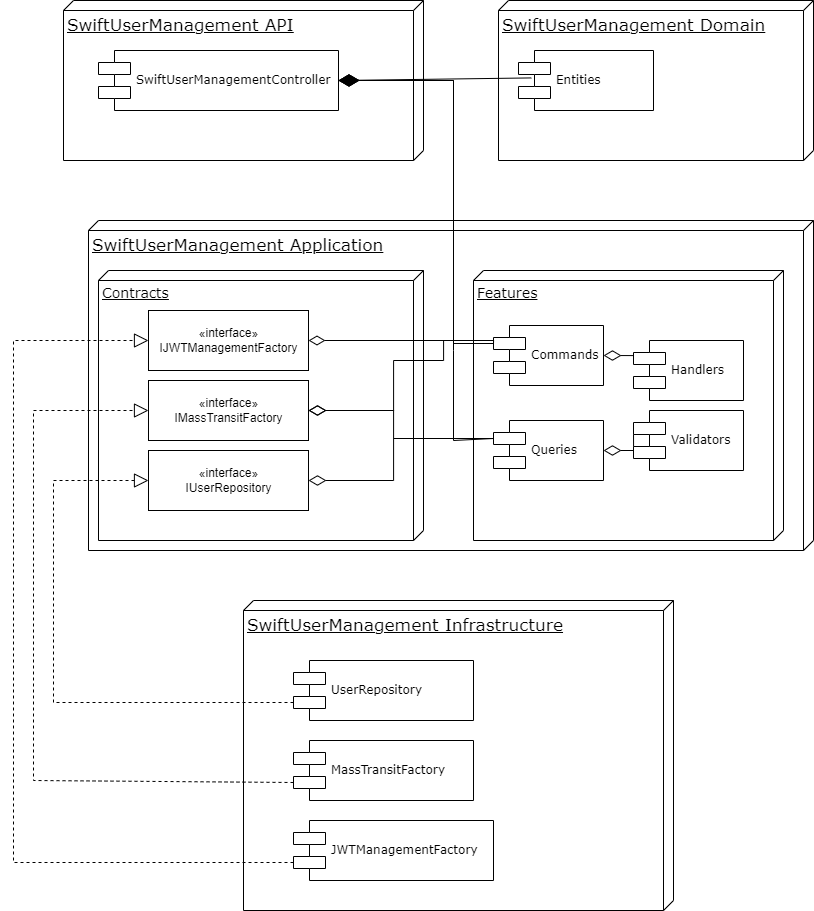


Figure : Swift API User Management Diagram

As seen in Figure 5, the user management service is a Web API (Application Programming Interface) structured using clean architecture [18]. This includes the API, Domain, Application, and Infrastructure layers. Clean architecture makes it easier for the programmer to follow SOLID [19] design principles so that the application is structured in the best possible way [20]. By structuring the user management service in this way, the application is much easier to maintain and can easily be extended in the future if any new requirements are created.

Another approach to structuring web APIs is MVVM (Model View-View Model) which requires less code to be written initially, speeding up development time. However, due to the user management service having a lot of features, clean architecture was chosen as it reduces the number of dependencies throughout the system. This allows components to operate independently and be extended or modified without interfering with other modules in the system. Clean architecture is also easier to test than applications with a monolithic architecture as all the code is broken up into lots of very small classes, which makes it much easier to find bugs in the system and allows tests to be written quicker [21].

The API component of the User Management service interacts with the client using HTTP/S and sends on any necessary requests to the rest of the back end. The application part of the system includes all the features of the code (e.g., creating a user, analysing game details) which include commands/queries, validators, and handlers. The command/query is responsible for holding all the data in the request and is passed down into the application section from the API. The validator is responsible for checking if the command/query is valid. Finally, the handler has all the necessary logic to complete the request. It uses the interfaces in the contracts section of the application to communicate with the database/ messaging broker or authenticate a user. The infrastructure component of the User Management service executes the commands contained within the handler. All the classes in the infrastructure inherit the interfaces in the application component; this allows the classes in the infrastructure to be substituted (e.g., changing database) without the rest of the service noticing.

### 3.4.3 Analysis/Calculation Services Architecture Design

The analysis/calculation services will be written as functions which receive and return data over the messaging broker to the User Management service. Each one of these services will perform as a specific task and they will be called upon by the User Management Service whenever they are needed.

The Calculate Difficulty service is called upon before a user starts a game and takes into account the previous results to give the user a new difficulty. If a user has been improving, it will give them a greater difficulty, whereas if a user is not improving then the difficulty score will decrease. The Single Game Analysis service is called upon at the end of a game in which the single result will be sent over to the back end and will be analysed to see if the score has increased or decreased and what the new score is. The Retrieve Game Results for Graph service would use all the results stored for the single user to analyse the results, and then organise them into a JSON that the front end can read so that the results can be displayed on a graph. The final Video Analysis service will analyse a video sent in from the front end to see if a user is displaying signs of arm weakness.

### 3.4.4 Communication

Communication between services will be achieved by using a messaging broker. This broker would use topics to send data between the User Management service and the rest of the services; these topics work like queues with a certain identifier that can be picked up by the publisher and the receiver. The service will use the AMQP Protocol (Advanced Message Queuing Protocol) which works in three major steps. A message is published by a producer which is sent over the queue/topic, and it is then consumed by a subscriber. This will allow the programmer to create services without having to create API interfaces and they can still communicate over the internet/ virtual network.

Communication between the User Management Service and the database will be achieved using TCP/IP; only the one service can communicate with the database to ensure that there is the least amount of dependencies possible in the system.

Communication between the back end and the front end of the system will be done via HTTP/S. The User Management Service would have endpoints in the API part of the application which would listen for HTTP/S requests from the front end. This Service also acts as a reverse proxy for the full system as any requests always go through it, and it then calls upon any extra relevant services to retrieve the data that it needs.

### 3.4.5 Database Design

Calendar

Description automatically generated with medium confidence

Figure : Database Entity Relationship Diagram

The application has been designed to store data on the user, their videos, and their game scores, ensuring the database stores the key data required for the Swift app. The tables connect with each other on the ‘User\_Id’ so that data can be retrieved from all tables. Every user can have many game scores, and every user can also have many videos.

# 4 Implementation

In conjunction with the design choices made, the implementation of the main components of the Swift app are detailed below.

## 4.1 Front End Implementation

Fulfilling functional requirement 1.0, non-functional requirement 1.5, and upholding the design choices made in Chapter 3, smartphone mobile applications across iOS and Android are used as the choice platform for implementation of the Swift app, to ensure it is as accessible as possible, to as many people, no matter what platform they align to.

The front-end is developed in JavaScript, within Visual Studio Code, with the app using Expo CLI v6.0.6 and encompassing React Native v18.0.0. The Expo Go app was chosen as it provides an “open-source client for running React Native apps on Android and iOS without having to build anything locally” [22]. The decision to use React Native as the development framework was not taken lightly, and other options were considered, such as Flutter and Xamarin. Flutter was the main contender as it is created by Google and, similarly to React Native, supports shared code for iOS and Android devices [23]. Xamarin is another powerful app development tool, however it makes use of C# as its development language [24], which the team were less confident in.

React Native is the best possible development framework for the Swift app, as not only does it provide shared components to work with multiple platforms, but also works in conjunction with the Expo Go app to allow for live reloading of the app, which can update with real time updates to the code. This benefited the implementation stage, as it both eased and accelerated the development process as changes occurred in real-time, which was advantageous due to the time constraints faced. As much as the Swift app is implemented with code that can be understood and run on both Android and iOS, it has been adapted to be, when necessary, platform-specific, regarding the styling (specifically surrounding dimensions and screen sizing). This allows the app to be accessible to all different device types and sizes, further fulfilling functional requirement 1.0, and non-functional requirements 1.1, 1.2, 1.3, and 1.5.

Aspects of the Swift app such as the Splash Screen have been implemented within the ‘App.js’ file, by importing and declaring assets such as the Swift logo. This allows for strong error handling, as if the Splash Screen is declared within ‘App.js’, if an exception is thrown (the app crashes), the Splash Screen can be shown to the user, which enhances the professional aspect of the Swift app.

### 4.1.1 Front-End Libraries and Dependencies

A wide variety of libraries have been used to implement the front-end of the Swift app, with the most utilised ones being *@react (v18.0.0),* and *@react-native (v0.69.0).* From these two imported libraries a plethora of different features are implemented into the Swift app, with *Stylesheet, Image, View, Text, Dimensions, SafeAreaView,* and *TouchableOpacity* being imported from *@react-native,* and used across all screens of the app. Other necessary components utilised include *Keyboard* (for user input), *Modal, Pressable* (for pop-ups), *Alert, Linking,* and *Platform.* Use of the Alert import is beneficial for social considerations, as this performs a check for more “serious” actions being performed e.g., logging out, or exiting the game mid-play. By implementing this check, the app is more user-friendly and provides the option to cancel this action.

Dependencies within the Swift app are managed by the package manager *Yarn*. Yarn was chosen as the package manager as it has the same features as existing workflows, whilst operating faster, more securely, and more reliably than npm [25]. Furthermore, Yarn has compatibility with npm workflows, and can resolve issues surrounding versioning by using lockfiles that can “lock” a dependency to a specific version, so that dependencies are not automatically updated which could cause dependency conflicts. These dependencies are stored within the ‘package.json’ file.

Relating to the *React* components imported, these are state-specific, and can be applied as Hooks to functions. Throughout the front-end, *useContext, useState, useRef,* and *useEffect* are implemented to parse through different functions. The implementation of the dark mode feature, for example, is created with the *useContext* hook, which is linked to the *themeContext.js* file created by the team. This stores the states and different hex codes each designated colour scheme has. For creating any modals or pop-ups on different screens, the *useState* hook will be used to define whether the pop-up should appear to the user, and when it should be hidden. Similarly, the ‘Email Results’ feature is implemented through a ‘const’ declaration of a *useState* hook. This allows a check to be run to see if the device supports email, and subsequently if email can be accessed from within the app.

The implementation of navigation within the Swift app is done through passing the {navigation} component within the declaration of the function. This can then be applied throughout the return statement of the screen, calling where necessary to either directly navigate to certain screens (e.g., Home Screen to View Results Screen), or dynamic navigation, such as the Go Back buttons provided on all screens (fulfilling functional requirement 1.0 and non-functional requirement 1.1). The screens to be included in this navigation are placed in a Stack, that is located within ‘App.js’. A complete list of all dependencies and versions imported for implementation of the Swift app, what requirement they relate to, and the feature they support within the front-end, can be found in Appendix 3.

### 4.1.2 Hand- Eye Coordination Game

Three JavaScript functions have been developed encompassing the circle, target, and obstacle coordination game components. The Circle component uses ‘PanGesture’ from the *@react-native-gesture-handler (v2.5.0)* package [26] to track finger movements across the screen. The gesture activates when a finger is placed on the circle. The ‘onUpdate’ callback continuously tracks the movement of the circle and updates the XY coordinates every time it moves a pixel in each direction. It is these XY coordinates that are used to calculate the accuracy, by measuring the difference in the position of the circle from the XY coordinates of the centre of the target. The *@react-native-gesture-handler* package is combined with the *@react-native-reanimated (v2.9.1)* library for performant gesture-based interactions [27]. The target is created in the ‘EndingCircle.js’ file using the Circle component from the @*react-native-svg* *(v12.3.0*) package [28]. The Circle component is used to define the radius, opacity, and XY coordinates of the centre of the target. The centre XY coordinates are specified in the ‘CoordinationScreen.js’ file, with the Y coordinate set to a constant value and the X coordinate set to randomly change with every iteration of the game. The radius of the target is set to 60 pixels, and the X coordinate is set to a random value within the boundaries of 60 and 320 (60 pixels from both edges of the screen), ensuring that the target will always be fully visible on screen. The opacity is set to 0.5 so the user can see the draggable circle when it is within the target. The obstacles are created using the ‘Obstacle.js’ file and their positions on screen set within the ‘CoordinationScreen.js’ file. As highlighted in Section 3.3, the number of obstacles and the distance between them is calculated based on the user’s current difficulty level stored within the app context. This is further discussed in Section 4.2.5. Pseudocode explaining the generation of obstacle positions is shown in Figure 7.

Text, letter

Description automatically generated

Figure : Pseudocode for Generation of Obstacle Positions

The user’s accuracy result is calculated as the distance between the centre of the draggable circle and the centre of the target. The X coordinates are used to determine the horizontal distance between the centres and the Y coordinates are compared to find the vertical distance between them. These values are then used as the input to the Pythagorean theorem to evaluate the straight-line distance between the centres. This distance is scaled to provide an accuracy result between 0 and 100, with 100 equating to perfect accuracy and 0 meaning the user has completely missed the target. Pseudocode for the accuracy calculation can be found in Figure 8. To calculate the time taken for the user to drag the circle to the target, the ‘Date.now’ method is first called within the ‘onStart’ callback and again within the ‘onEnd’ callback when the user has lifted their finger. The ‘onStart’ time is then subtracted from the ‘onEnd’ time and divided by 1000 to find the user’s time taken in seconds. The ‘runOnJS’ function is used to call the ’endGame’ callback asynchronously from the UI thread, ensuring the ‘endGame’ function is not called until the accuracy and time taken arguments have been calculated. The ’endGame’ function sets the time taken state and sends the accuracy and time taken results to the backend before navigating to the coordination results screen.

Text, letter

Description automatically generated

Figure : Pseudocode for Calculation of User Accuracy

### 4.1.3 Displaying Results Analysis

As discussed in Chapter 1, the primary goal of the Swift app is to provide individuals with a means of monitoring their rehabilitation progress over time. To achieve this goal, charts have been implemented in the front-end providing users with options to view their average weekly results or average daily results. Several charting libraries were considered for implementing the results analysis graphs, with pros and cons weighed for each. The ‘react-native-charts-wrapper’ and ‘react-native-chart-kit’ were strong contenders, however after careful deliberation the *@victory-native (v36.6.8)* library [29] was chosen as it is easy to use and fully customisable with a wide variety of chart types. *Victory-native* has a peer dependency to *react-native-svg*, which was already installed for creation of the target in the coordination game, as discussed in Section 4.1.2. With the flexibility of *victory-native*, charts can be personalised to the needs of the Swift application. Furthermore, victory-native charts are responsive, meaning they can adapt to the variety of screen sizes across iOS and Android devices. An advantage of victory-native over other charting libraries is the support of animations and transitions. The animate prop is applied to both charts to draw out the data points when the charts are first displayed, enhancing the user experience.

A radar chart is used to display the weekly average results which allows for performance analysis across the four key coordination game metrics (accuracy, time, difficulty, and engagement) within one summarising graph. Users can instantly see which metrics they have a deficiency in, and therapy exercises can be adjusted accordingly to focus on this pain point. The radar chart is developed using the ‘VictoryChart’ wrapper component with the ‘VictoryPolarAxis’ component.

A line chart is used to display the weekly average results, showing the user’s average accuracy and time taken metrics for the last seven days. ‘VictoryLine’ composed with ‘VictoryChart’ was used to develop the line chart. The *@react-native-simple-radio-button (v2.7.4)* package [30] is used to display a radio button above the graph, making the line chart interactive by allowing users to switch between viewing their accuracy or time taken results.

## 4.2 Back-End

### 4.2.1 Containerisation and Orchestration

Docker is a widely used technology which uses OS-level virtualisation to containerise applications [31]. The team has chosen to containerise each service to allow them to work independently from one another, whilst also allowing for orchestration to make the services communicate and work together. If containerisation was not used, then the system may behave differently once deployed compared to when it was run locally, and the system administrator would have to install all the necessary libraries to get the code to work. Whereas, when Docker is used all libraries are installed on the image and the code will work on any machine that has the Docker engine installed. A Dockerfile is written for every service to tell the Docker engine how to build an image and which files to copy over to the container. This ensures that the images are as small as possible, and are always the same, no matter what machine they are running on. Most Docker images also use a base image which would then be extended, for example the Python services would use the “Python:3.9” image so that the Python code can be interpreted. The rest of the docker images can be seen in appendix 6.

Container orchestration has been achieved using ‘docker-compose’ [32]. ‘Docker-compose’ uses a “.*yml”* file to execute automated building of docker images (by pointing to where the Dockerfiles are stored) and then running containers built from those images. In the ‘docker-compose’ file, environment variables are specified to store secrets like database connection strings and database usernames/passwords. Once the application is deployed, these secrets would be stored in a public cloud key vault, but during development this is possible by using the environment variables declared within the ‘docker-compose’ file. Furthermore, the ‘docker-compose’ file links containers to one another and forwards container ports onto specific host ports so that the containers that need to be exposed, such as the User Management service, can be seen from the host machine. Volumes are declared in the “*.yml”*  file to allow for persistent storage of data, for example, if the back end goes down then the data in the database will be stored on the host machine so that it can be copied back onto a container when the backend is rebooted.

When using a ‘docker-compose’ file, a virtual network is automatically created by Docker to allow containers to communicate with one another. Each container is assigned its own IP address and runs on its own port. However, the host machine would not be able to see any of these IP addresses or ports, which is why containers that need to be seen on the host need their ports forwarded, for example, the User Management service and the RabbitMQ management system. The rest of the containers would not have their ports forwarded for security reasons and so they cannot be accessed outside of the virtual network.

### **4.2.2 Communication**

Communication between the front end and the back end is achieved using HTTP/S over the web, and the front end can call the back end using asynchronous fetch statements. These fetch statements allow the rest of the application to carry on functioning whilst they are waiting for the data to be sent from the backend. This ensures that there are no performance delays, and the application runs smoothly for the user. Regarding the back end, the User Management service has endpoints which the front end can call upon; these endpoints follow a strict naming scheme of “domainName:port/api /swiftusermanagementcontroller/[endpoint]”. The first part of the URL is always the same, making it consistent and easier to call upon from the front end. All the HTTP Get requests use URL parameters which are added onto the end of a URL, whereas other requests would use a body or a form.

RabbitMQ is a message broker which allows the services to send data and communicate with each other [33]. The User Management service has a RabbitMQ factory on the infrastructure layer which connects to the messaging broker and is able to publish/subscribe to messages from the Python services. The Python services are set to always wait for messages, so that they can perform calculations as soon as an event is published by the User Management service. Topics are used to route messages between service which is achieved by binding to a topic on the message broker, and then publishing a message to that topic. The function that receives this message binds onto that topic and subscribes for that message. Using topics such as this allows there to be only one message broker, but ensures that the correct service always receives the correct data. RabbitMQ also load balances requests for the programmer, for example, if there are multiple Python services bound to that topic, RabbitMQ will send the messages to whichever instance is free to complete the calculation and will not overload any services. If a message is lost, RabbitMQ will also keep a copy of the message and will send it to the next available instance to guarantee message integrity.

### 4.2.3 External Containers Used for Implementation

Communication through RabbitMQ can only be achieved if there is a RabbitMQ management container running. The ‘docker-compose’ file specifies that the version of RabbitMQ being used is “rabbitmq:3-management-alpine” and the ports which are forwarded onto the host machine so the RabbitMQ management web page can be accessed. The ports used by RabbitMQ are 5672 for the management application and 15672 for everything else.

Portainer is a container management system which is used for holistically monitoring the back end and ensuring every container is healthy [34]. Portainer can also be used to check logs and specific properties of a container for debugging if issues arise. The version of Portainer used is “portainer/portainer-ce” and it is forwarded onto port 9000. PgAdmin is used for managing the database and ensuring all of the data is correct. The image for pgAdmin used is “dpage/pgadmin4” and it runs on port 80, but due to the port already being in use, PgAdmin is forwarded to port 5050 [35].

PostgreSQL is used as the relational database for the Swift application; it runs in a Docker container and is accessed through the User Management service. PostgreSQL supports image, video, and audio storage as well as support for graphical data which was essential for the previous implementation of the hand pronation assessment. Support for audio storage will be beneficial for future implementation of other neurological assessments, such as speech recognition. The docker image used is “postgres” and it runs on port 5432.

### 4.2.4 User Management Service

The entire back end was previously written in Python Flask, but this semester it was split into services and the user management component was re-written using C# ASP.Net core with .Net 6.0. C# makes it easier to structure the application into clean architecture as it is an object-oriented language, and it needs to handle all the requests sent to the back end due to it being the reverse proxy of the application. This means that it needs to handle the most throughput, have the lowest latency and lowest error rate possible. In a thesis written by E. Qvarnström and M. Jonsson, it was tested to see if ASP.Net Core is better than Flask and Express.js in metrics such as latency and error rate. ASP.Net Core proved best in every metric and therefore was the top choice for the User Management service [36].

To help with the passing of data through the multiple layers of the User Management service, MediatR v10.0.1 was used to decouple a system and reduce dependencies between objects. Within the controller, the “Mediatr.Send()” function is called, and it takes control of passing through a query or command straight to the handler. As seen in Figure 9, MediatR acts like a “middle man” to send data to the application layer, making the controller cleaner and easier to read. The complexity of sending data to the application layer can be fully abstracted and the programmer only needs to remember one method. When using MediatR, commands and queries were split to improve performance and increase security, which is the Command and Query Resource Segregation Pattern (CQRS) [37]. This allows the system to be more extendable, making it easier to add extra functionality in the future, and means that if performance ever becomes an issue then extra instances of the database can be added: one being a read and one being a write database, to once again increase performance.

Diagram

Description automatically generated

Figure : Implementation of the Mediator and the CQRS Design Pattern [38]

To ensure clean architecture is followed, the application is split up into one C# Web API project, and three C# class libraries. This allows for further decoupling of the project and makes it easier to find code in the large system. The C# Web API project holds the controller, which extends the ControllerBase class from the Microsoft.AspNetCore.Mvc package and holds all the endpoints/URL Mappings for the system. The application layer (the first C# class library) holds the commands and queries which are C# classes extending Irequest class from MediatR, and these classes have properties for all the necessary data to complete the request. Once the request is passed into the application layer, the validator, which extends the AbstractValidator class from FluentValidation, ensures that the data in the request is valid, by returning an error back to the controller if there is any invalid data. Errors which occur in the application layer are returned to the user as a HTTP 400 Bad Request status code, whereas if the request is valid then a HTTP 200 Ok status code is returned. After the request has been validated, it is passed through the handler which calls the necessary functions in the infrastructure layer (the second C# class library) to fulfil the requests. The final C# class library is the domain layer which contains entities (such as the User entity) which are called by the application layer when an entity is needed.

The Infrastructure layer contains all the required logic to communicate with the RabbitMQ message broker, to save and retrieve data from the database and to generate authentication tokens using JWT (JSON Web Tokens). The communication and authentication classes use the factory design pattern so that the application layer can choose which method to use for the given task. Communication with RabbitMQ is achieved using the RabbitMQ package and topics to route information, as explained in Section 4.2.2. JWT Authentication is achieved using the Microsoft.IdentityModel dependency to generate authentication tokens that can be used to access endpoints for ten minutes. Once the token expires, a new one must be generated for the user. The save to database class uses the repository pattern to make unit testing easier and keeps all the logic for stateful saving in the same place. The Npgsql library is used for connecting to the database and preparing statements to prevent SQL Injection. SQL Injection prevention is explained in more detail in Section 4.3.

Dependency injection is a core part of the ASP.Net Core architecture and to facilitate this, the dependencies need to be registered in every project so they can be used in the implementation classes (the dependency injection pattern is used for this). For a simple ASP.Net Core project, dependency registration is carried out in the one “program.cs” file. However, as the User Management service has multiple projects, each project needs its own “ServiceRegistration” file which is imported into the “program.cs” file in the top Web API project. This ensures that all dependencies can be registered for the correct assemblies, as the class libraries would not be able to use all the dependencies if they were not registered in the same project. For a list of all dependencies and libraries used, see Appendix 5.

### 4.2.5 Results Analysis

The application makes use of Python scripts that communicate with the C# backend via the messaging broker to analyse the users in-game data and return meaningful statistics. The decision was made to use Python scripts as it has a lot of libraries which are useful for AI and data analysis. The Python files have been written using Python 3.9, making use of several of Python’s standard libraries. Another benefit to using Python is that its standard library is extensive, offering a wide range of facilities which are used when analysing data. Some standard libraries imported are Json, datetime, and statistics. The project also makes use of the pika library which allows for binding onto the RabbitMQ messaging bus with the logic behind this explained in Section 4.2.2. The rest of the Python external dependencies can be seen in Appendix 4.

In the Single Game Analysis service, there are two main Python scripts being used to analyse the data, “getIndividualGameResults” and “getResults”. The first script, “getIndividualGameResults”, is given an array of all the user’s previous in-game scores, before comparing the most recent results to the previous game results. This will allow for calculating a percentage of how the results have changed and whether they have increased or decreased. These values are placed in a JSON object and are published to the C# backend, where it is sent to the front end and displayed to the user. The aim of this first analysis is to provide the user with an initial overview of their game results in a clear format, and whether they have made progress since the last gameplay. To access their results from the last week, the user can visit the “View Results” screen.

When a user navigates to the “View Results” screen, this triggers the Retrieve Game Results for Graph service. The second file, “getResults”, is sent an array of the user’s previous results and sends the data to three different subscripts, which each collect different metrics that can be displayed to the user. The three subscripts will return a list of daily averages for time taken and accuracy and returns a summary of the results to be displayed on the radar chart. The summary data consists of a one-week average for time, accuracy, difficulty, and a value for how many days the user has played the game in the last week, to encourage them to play daily. Once the main “getResults” file has received these values, it deserializes them into a JSON object before publishing to the C# backend where it can be sent over to the front end and displayed to the user.

As discussed in Section 3.4.3, the Calculate Difficulty service was created specifying that each user will start the game with a difficulty score of 50. When the user logs in and clicks on the coordination game screen, it triggers an event which retrieves the user’s last game score from the database and extracts the accuracy and difficulty values. These values are then used to calculate a new difficulty score for the next time the user plays the game. If a user scores an accuracy of 0 it will subtract 5 from their difficulty score, and if they get a score of 100 it will add 5 to their difficulty score. Users may be at different stages of their stroke rehabilitation, and so the Difficulty Calculation service automatically adjusts the difficulty level to be personalised to the user’s needs.

## 4.3 Cyber Security Considerations

### 4.3.1 SQL Injection

To mitigate against the risks of SQL Injection [39], the Npgsql package was used to connect to the database. Npgsql automatically prepares statements for the programmer if they are not explicitly prepared, however this decreases the performance of the application and it is therefore still recommended to prepare these statements manually [40]. Due to this, SQL statements were manually prepared to mitigate SQL Injection, as can be seen in Figure 11.

Text

Description automatically generated

Figure : Preparing SQL Statements to Mitigate SQL Injection

### 4.3.2 Security for Communication

As described in Section 4.2.2, communication between front end and back end is achieved using HTTP/S. In C#, this was a case of forwarding the correct ports in the Dockerfile (port 443) as HTTP/S is already set up for the user. The only complication faced was that Expo does not accept self-signed certificates, and so during development it was ensured that a closed network was used as traffic had to be run over HTTP. Once the application is deployed, this will not be an issue as the HTTP/S certificate will be from a signed certificate authority.

All communication between services is conducted on the virtual network within the Docker compose orchestration and is never exposed onto the host machine. This means that this network traffic cannot be tracked and is safe in the container. For extra security, RabbitMQ also supports TLS which stops any clients that do not have TLS connecting to the messaging broker and prevents any non-encrypted traffic from being transferred [41].

### 4.3.4 Security for Authentication and Password Storage

The Swift app has its own Swift login and registration feature which allows the user to create an account with their own email and password. Security had to be taken into consideration as brute force attacks are becoming increasingly prominent [42]. Consequently, strict rules have been put on the passwords to ensure that the password has an uppercase character, a lowercase character, a number, a special character, and has to be at least eight characters. When creating an account, the user also needs to have a valid email to ensure that any communications can only be sent to a valid email inbox. Validation is implemented using the FluentValidation.AspNetCore (v11.2.2) package which is used to check classes and is easy to understand with human readable language. Figure 12 illustrates how data is validated when creating a user on the application layer.

Text

Description automatically generated

Figure : Validating a New User

Session tokens have been implemented into the system using JWT (Json Web Tokens) which ensures that the given user has access to the application. When a user calls the “/auth” endpoint, it hashes their password and checks to see if the combination of their email and password matches what is stored in the database. Once a token is generated for the user, it then lasts for ten minutes until they must retrieve a new token to continue using the back end. If a request is sent to the back end for an endpoint that requires a token and one does not exist in the request, an unauthorised error will be thrown and the data would not be retrieved.

A way of reducing risk against data leaks is ensuring that all passwords are hashed in the database. If the passwords are hashed, there is no way of getting back to the plain text that the user inputs into the registration form. Bcrypt is used as the algorithm to hash passwords whenever they are saved into the database. As Dan Arias has said in an auth0 blog, “the largest benefit of bcrypt is that, over time, the [iteration count can be increased to make it slower](https://en.wikipedia.org/wiki/Bcrypt) allowing bcrypt to scale with computing power. We can diminish any benefits attackers may get from faster hardware by increasing the number of iterations to make bcrypt slower” [43]. For an extra layer of security, the passwords are also salted before being hashed [44].

# 5 Software Testing

## 5.1 Testing Strategy

Initial testing of the application will involve conducting manual function testing and debugging to ensure the individual functions are working as expected and to identify and fix any bugs or issues. The C# API endpoints will be manually tested using Swagger and Visual Studio will be used for debugging and creating C# test classes. PyCharm will be used to debug the Python backend analysis code and Visual Studio Code to debug the JavaScript front end code. The Expo CLI and Expo Go mobile application will be used to run the Swift app to identify any bugs or issues that require fixing. Following manual testing of the system, unit testing will be conducted in which individual units and components of the application will be tested in isolation to ensure they are functioning correctly. Unit testing of the front end will be completed using the Jest JavaScript testing framework. Backend unit testing will involve the use of the PyTest testing framework for the Python analysis services and the xUnit testing framework for the C# backend services. The SpecFlow testing framework will also be used for behaviour-driven testing of the C# services. On completion of unit testing, integration tests will be conducted, where the individual components will be combined and tested together to identify any issues with system integration. As with the unit tests, integration testing will be conducted on the C# services using a behaviour-driven testing framework. As a final step before conducting alpha testing, system testing will be completed to ensure that the entire application is functioning as expected and is ready for deployment. This will be completed in the form of acceptance testing, in which test cases will be written relating to requirements and verified by recording results as a ‘Pass’, ‘Partial Pass’ or ‘Fail’. After completing acceptance testing, the application will be demonstrated to alpha and beta testers to receive feedback from those with both a technical and non-technical background.

## 5.2 Debugging and Manual Testing During Development

When developing in C#, Visual Studio was used as the Integrated Development Environment (IDE) to develop the application. It was chosen as it supports IntelliSense for C#, and automations to accelerate coding in C#, such as automatically generating test classes or whole projects with the correct folder structure. Visual Studio also comes with a powerful debugger that lets the programmer run the code in real time and use breakpoints to spot logical problems in the code. ASP.Net Core apps also come with Swagger readily available for manual endpoint testing; Swagger uses the OpenAPI specification to call the necessary endpoints and receive data from the ASP.Net Core Web API [45].

PyCharm is a powerful IDE for developing Python-based applications. It is a useful tool for debugging code as it provides an interactive debugger with a GUI [46]. This allowed for breakpoints and watchpoints to be set, for code to be examined line by line, and for analysis of the program’s data flow. It also provided a powerful code editor with syntax highlighting, code completion, and refactoring features. PyCharm allowed for code to be debugged quickly and easily, ensuring mistakes could be quickly identified and fixed, thus saving time and effort. Additionally, the debugger was used to inspect the values of variables, find the source of errors, and step through code to better understand its logic.

Visual Studio Code was chosen as the IDE for developing and debugging the JavaScript front end application. With the use of the Expo CLI and the Expo Go mobile application, the system can be run on any smartphone in real-time, alerting the user with warnings and error messages if issues arise [47]. This was useful for debugging during development, and the manual system testing outlined in Section 5.5.

## 5.3 Unit Testing

### 5.3.1 Front-End Unit Testing

Unit testing is an important process in software development. It ensures that the code is working correctly, and that the functionality is meeting the expectations of developers. Within the React Native framework, the JavaScript language was used to develop the front end of the Swift app, and so initially Jest was selected as the testing framework, as it is the recommended testing framework for Expo applications [48]. Jest relies on the concept of “mocking” certain components, and this testing was to be based on the technical requirements of the system, detailed in Sections 2.2 and 2.3. However, due to an inability to successfully mock components of the Swift app, many issues subsequently arose when attempting to implement Jest testing, namely isolating code from other code it may need to interact with. Therefore, no Jest tests ran successfully, as the tests did not behave as expected when mocked, or hardcoded. To ensure the front end was still tested comprehensively, manual acceptance testing was conducted by the team, as detailed in Section 5.4, as well as alpha testing, detailed in Section 6.1, to ensure all components of the Swift app functioned as expected.

### 5.3.2 Python Analysis Services Unit Testing

The Python analysis services were tested using PyTest which is a unit testing framework for Python applications. This framework was chosen as it is lightweight and allows for parallel test execution, meaning that tests can be run quicker than if they were in sequence. Separate tests have been written for every function, and the tests used valid/invalid data to ensure the robustness of the system. As can be seen in figure 13, assertions were used to test the code with invalid data. The “calculateDifficulty()” function is expecting to receive two integers, the first being the accuracy rating achieved from the game, and the second being the difficulty of the respective level. In this example, the assertion is testing that when a string is entered as an input instead of an integer, it should output a prewritten JSON with an error message. Figure 14 illustrates PyTest results in the “test\_difficultyCalculator.py” file which passes all tests.



Figure : Example of Testing for Invalid Data in PyTest



Figure : Example of Tests Passing for the Analysis Services

### 5.3.3 C# User Management Service Unit Testing

Several C# testing frameworks were considered, but xUnit was chosen as it provides a better isolation of tests and does not need as many method attributes as Nunit and MSTest [49]. Moq was used as the package to mock dependencies and any external classes during unit testing, as the aim of unit testing is to only test the single class. A further benefit of Moq is that it provides an automated means of mocking classes and method calls [50]. When the team attempted to complete unit tests with packages such as RabbitMQ and Npgsql, it was found that, there were technical difficulties with mocking which led to the C# User Management service not being fully tested within the time constraints. The unit tests that were completed in time were the token authentication unit tests which can be seen in Figure 15.

A screenshot of a computer

Description automatically generated with medium confidence

Figure : Authentication Unit Tests Results

Behaviour driven testing has been used for the C# unit and integration tests as the majority of the team were unfamiliar with C#. Behaviour driven development (BDD) allowed the team to communicate what tests are created and how they link back to the requirements, as it uses scenarios and features written in Gherkin to explain what the test will do, and how it will achieve it. An example of a feature file written for the authentication unit test has been illustrated in Figure 16. SpecFlow was chosen as the BDD testing framework as it is well documented with learning materials on its website [51]. Each step is then defined in a step definitions file so that SpecFlow, can map the C# code to the feature file and run the test.

Text

Description automatically generated

Figure : Feature File for the Authentication Unit Test

## 5.4 Integration Testing

The C# User Management service acts as a middle ground between the Python analysis services and the JavaScript front end. This meant that complete integration testing could be performed, as multiple components of the back end can be tested working together. Behaviour driven development (BDD) was also used as the testing framework for integration testing as discussed in Section 5.3.3. As the lower-level packages are no longer mocked out using the Moq package meaning that the relationships between multiple components can be tested. Test Docker containers had to be instantiated by the test scripts when they were running. For the user repository tests, a PostgreSQL test database was instantiated and for the RabbitMQ tests a RabbitMQ instance had to be created. This was carried out by writing command line scripts with C# with these instances being created before all scenarios for each component.

The Infrastructure layer has been tested to check the correctness of results for valid and invalid data when sending data to RabbitMQ, and saving data into the database as those are the remaining two components of the system within the infrastructure layer after authentication. It can be seen in Figure 17 that all of these tests pass and the system works correctly. Due to time constraints the team was unable to fully test the application and API layers, so testing of those layers was carried out manually using Swagger and was also tested during acceptance testing in Section 5.5.

Graphical user interface, text

Description automatically generated

Figure : User Management Service Testing Results

## 5.5 Acceptance Testing

Acceptance testing was utilised to determine whether the Swift app meets the technical requirements it has been designed for, detailed in Sections 2.2 and 2.3. This form of testing is usually completed at the end of the development process before the software is released to users. The test cases were designed to mimic real-world scenarios and ensure the Swift app functions as expected – this was accomplished using a ‘Given – When – Then’ structure, with each test case having a relating requirement. The goal of acceptance testing is to identify any issues or defects in the app before it is released to users, so they can be fixed in advance and the Swift app can be delivered with a high level of quality.

The acceptance tests were performed by two teams of testers, one with an iOS device and one with an Android device. The success of each test case was recorded with a “Pass”, “Partial Pass”, or “Fail” result. A complete breakdown of each individual test case for acceptance testing can be found in Appendix 8. A summary of the overall results, categorised by key areas tested, is detailed below in Table 4.

|  |  |  |
| --- | --- | --- |
| **Test Category** | **Number of Tests** | **Summary of Results** |
| Navigation | 3 | 15 screens tested, 14 passed with 1 partial pass. This partial pass has since been resolved. All back buttons work. |
| Register | 4 | All tests passed |
| Login | 4 | All tests passed |
| Layout | 2 | All tests passed |
| Accessibility | 3 | 2/3 tests passed, 1 partial pass. |
| Playing Coordination Game | 3 | All tests passed |
| Viewing Results | 5 | All tests passed |
| Email | 2 | All tests passed |

Table : Summary of Acceptance Test Results

# 6 User Evaluation

This chapter discusses the approach taken for alpha and beta testing, and documents the changes made as a result of these findings.

## 6.1 Alpha Testing

Alpha testers consisted of a group of four Higher Level Apprentice students and a further three Level 3 Software Engineering students. The testers were given a brief introduction to the problem to be addressed and a demonstration on how to use the application, before proceeding to test the application for themselves. A survey was prepared containing a list of targeted questions relating to requirements outlined in Section 2 to receive feedback on key areas of the system. Testers were asked to rank the application features on a numerical scale from 1 to 10 (1 – very poor, 10 – excellent). These responses allowed application strengths and weaknesses to be identified so improvements could be made before completing further beta testing. Table 5 outlines the list of questions posed to alpha testers, and the requirement each question relates to, split by the key areas of the system. Results from alpha testing can be found in Appendix 9.

|  |  |  |
| --- | --- | --- |
| **Question Number** | **Related Requirement** | **Question** |
| **Registration and Login** | | |
| 1 | 2.0 | How would you rate the ability to register an account with Swift? |
| 2 | 4.0 | How would you rate the ability to login with Swift? |
| **User Interface** | | |
| 3 | 1.1 | How would you rate the navigation between screens? |
| 4 | 1.2 | How would you rate the layout of the user interface - is it easy to understand? Is the styling consistent across screens? |
| 5 | 1.3 | How would you rate the text and button size across screens? (1 = very difficult to read, 10 = very easy to read) |
| 6 | 1.4 | How would you rate the dark mode style? Are the colours contrasting enough? |
| **Game Design** | | |
| 7 | 6.1 | How would you rate the colour scheme and layout of the game? |
| 8 | 6.1 | How would you rate the size of the circles/obstacles - were they clear enough? |
| **Game Play** | | |
| 9 | 6.1 | How clear was the tutorial in explaining the game instructions? (1 = not clear at all, 10 = very clear) |
| 10 | 6.0 | How quickly did the game load? (1 = very slowly, 10 = immediately) |
| 11 | 6.2 | How would you rate the difficulty of the game? (1 = too easy/difficult, 10 = appropriate difficulty) |
| 12 | 6.2 | Did you enjoy playing the game? Did you find it engaging? (1 = didn’t enjoy it at all, 10 = really enjoyed it) |
| **Game Results** | | |
| 13 | 10.0 | How quickly were results displayed on completion of the game? (1 = very slowly, 10 = immediately) |
| 14 | 7.1 | How would you rate the scoring of the game – were you able to understand the results breakdown? |
| 15 | 10.1 | How would you rate the charts displaying the average game results – were they easy to understand? Were they clear to read? |
| 16 | 10.2 | How would you rate the explanation of result metrics? Was it clear what each metric was representing? |
| 17 | 12.1 | How would you rate the ability to send game results via email to a medical professional? Do you think this is a useful feature? |

Table : Survey Questions Used for Alpha Testing

## 6.2 Beta Testing

Beta testing was conducted to receive feedback from those with a non-technical background, to indicate how the Swift app might be accepted by the wider public. 5 participants between the ages of 14 and 50 were briefed on the purpose of the application and were asked to complete the same survey questions posed to alpha testers, shown in Table 5. Survey answers from beta testing can be found in Appendix 10.

## 6.3 System Evaluation

This section details the key feedback points received from alpha and beta testing, and the changes that have been made to the application as a result. Potential future improvements are also discussed, which were unable to be implemented due to time constraints.

### 6.3.1 Application Front-End

Overall, feedback was positive regarding the user interface of the Swift app. Both alpha and beta testers stated that all buttons navigated to the expected screens, confirming that non-functional requirement 1.1 was met. Testers commented that the styling was consistent throughout the application and buttons and text were generally easy to read, fulfilling non-functional requirements 1.2 and 1.3. Some points for improvement on the Home Screen included moving the logout button to the Settings Screen and increasing the size of the tab navigation icons. It was noted that the mail icon on the Home Screen did not activate the email feature, and users instead had to select the ‘Send Mail’ text under the icon, which was inconsistent with the other Home Screen buttons. There were several issues identified with the dark mode feature, such as the draggable circle in the coordination game blending in with the background colour, and the weekly progress results screen not changing colour when dark mode is selected. Based on the feedback received from alpha and beta testing, changes have been made to the application front-end to improve the user experience. To assist those with visual impairments, font sizes have been increased in the results screens and the icons in the tab navigation have been enlarged to improve readability. The logout button has been moved from the Home Screen to the Settings Screen to be more consistent with other mobile applications. Improvements were made to the dark mode feature to ensure colours are now contrasting in the coordination game and text in the results screens is now white to improve readability against the dark background, satisfying non-functional requirement 1.4. The team struggled with implementing multiple contexts within a class as all other screens have been implemented as functions, and thus have been unable to use the dark mode feature within the weekly progress results screen. The email feature has been updated and is now accessible by clicking the mail icon on the Home Screen, fulfilling non-functional requirement 12.1.

### 6.3.2 Hand-Eye Coordination Game

Feedback on the game design was generally positive across both alpha and beta testing. Testers remarked that the game layout was clear and easy to understand, and the colour scheme was consistent with the Swift application colours. Notable comments on the draggable circle included that the circle may be too small to see behind the user’s finger, and that the circle disappears when dragged over the target which created confusion when trying to place the circle in the centre. Most testers gave the tutorial a rating of 10, indicating that the instructions provided were clear which meets non-functional requirement 6.1. A particular positive aspect of the game was the personalised changing difficulty level, which users said was engaging and provided a sense of achievement when they reached a higher difficulty level, satisfying requirement 6.2. Some testers suggested having the option to select a difficulty level to give users more control over their gameplay. Overall, testers enjoyed the game but made some recommendations to improve engagement, including adding sound effects or an animation on completion. In response to this feedback, the opacity of the target was decreased so the draggable circle was visible within the target. The team was unable to implement a changing difficulty option due to time constraints and so this is a point for future development, detailed further in Section 8.6.1.

Testers responded positively to the coordination game results screen, with specific praise for the percentage change data as this gives a greater insight into the user’s results which satisfies non-functional requirement 10.1. However, in the scenario when a user plays the game for the first time, the percentage change is zero as there are no previous results to compare to, which caused some confusion for testers. Furthermore, the ‘time taken’ result is rounded to the nearest second, but if the user consistently completes the game within the same second, their percentage change will be zero. Therefore, it was suggested that the time be measured in milliseconds for a more accurate measurement of the user’s time taken result. A key feedback point on the weekly progress radar chart was that it was unclear what the ideal results should be as the user’s results are displayed as the top axis values and so the user has no goal to work towards. Testers thought the charts were well designed but would have like the option to view earlier results in the charts and to see their average difficulty in the daily progress line chart. The response was generally positive on the explanation of results, fulfilling non-functional requirement 10.2, but there were some inconsistencies noted in the description of the engagement metric. Due to time constraints, the team were unable to implement all suggested changes to the results screens, so decided to focus on changes to the weekly progress chart and explanation of results. To highlight the user’s ideal results, an additional layer was added to the radar chart and the axis values were scaled accordingly. The ‘explanation of results pop-up’ was updated so the metrics were consistent with one another, and units were added to the Y axis labels in the line chart. In future developments, the charts should include options to view historic results as well as a view for a breakdown of results for games completed within a day.

### 6.3.3 Application Back End

Both the registration and login processes were described to be relatively straightforward and smooth. Some testers noted that they would prefer the alert for invalid data to specify which data input was incorrect, and stated that the password requirements for registering should be displayed to the user before they enter the password, rather than in the alert message. The game and results were reported to have loaded promptly, which has improved upon a weakness stated from the previous project. Taking this feedback on board, the team added the password requirements to the Register Screen and removed them from the alert, making the alert message more concise so it is clear to users what error has occurred.

# 7 Project Management

This chapter details the team’s approach to working together throughout the course of the project, including the roadmap, tools and techniques used for project management, and the challenges and risks faced during this process.

## 7.1 Roadmap and Sprint Planning

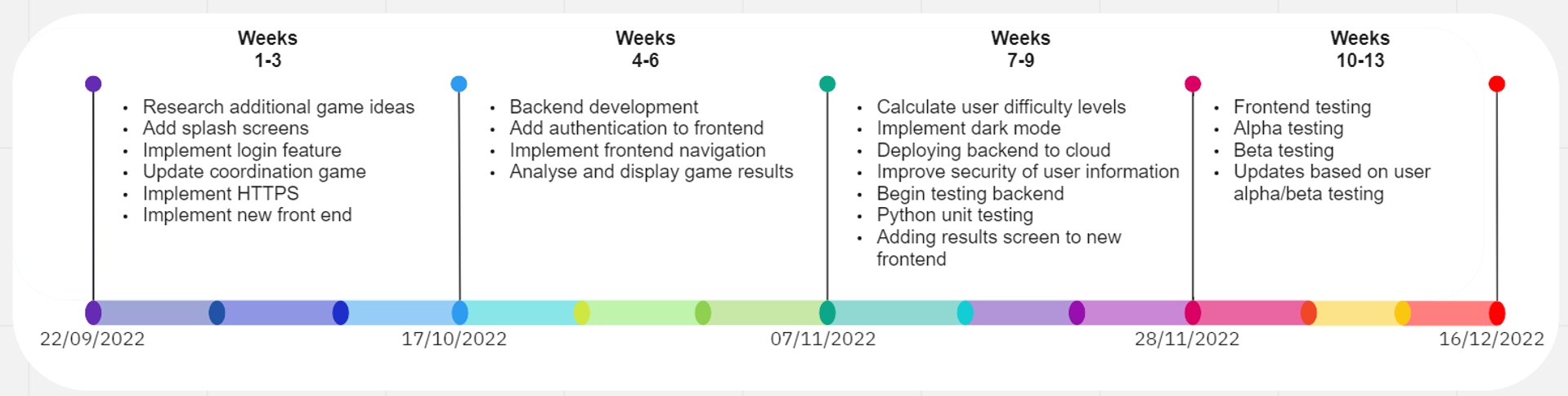


Figure 10: Roadmap representing project timeline

The roadmap provided a clear timeline of the tasks to be completed, enabling the team to plan effectively. Potential risks could be identified early on and mitigated to avoid delays in the later stages of the project. The roadmap was used as a reference point to track the team’s progress towards the key milestones shown in Figure 10.

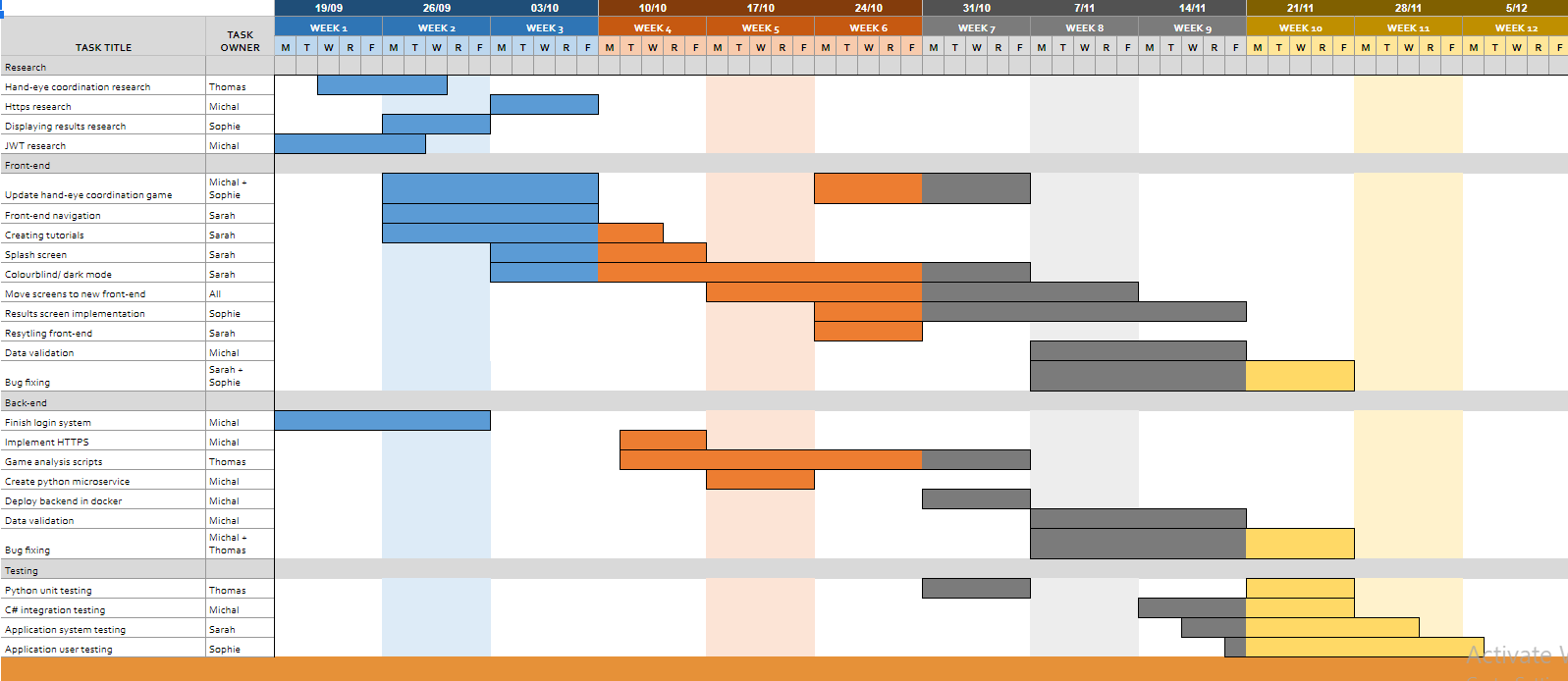


Figure 11: Feature File for the Authentication Unit Test

At the beginning of the project, a Gantt chart was created to assist in project management. Using the chart, the team was able to visualise the project timeline, set and adjust deadlines as required, and track progress. The Gantt chart shown in Figure 11 displays the allocation of tasks across the project timeline. This highlighted dependencies between tasks, enabling efficient planning and management of roles and responsibilities.

## 7.2 Tools and Techniques Utilised

Throughout the duration of the project, the team made use of a plethora of different tools and techniques relating to project management, teamwork, and maintaining strong communication.

### 7.2.1 TimeTree

TimeTree is a collaborate calendar app/website that was utilised by the team throughout the entirety of the project to keep track of schedules when organising team meetings. This allowed team members to add their individual commitments for each week e.g., other university modules, so all team members were aware of prior engagements, and able to plan team meetings around this (see Appendix 7).

### 7.2.2 Meetings, Microsoft Teams, WhatsApp and Miro

The team agreed to use a private Microsoft Teams channel as the centralised space for all online/hybrid meetings. Benefits of this included having call logs in one place, and the ability to add shared files to the channel, ensuring all team members had access to relevant documents. The team aimed to have meetings at least three times a week, making use of a combination of online, hybrid, and in-person meetings, however the number of weekly meetings often increased with workload. The team used a WhatsApp group chat as the main form of communication throughout the project, as this was found to be the most efficient way to contact one another. This group chat was utilised on a near-daily basis, with general updates and questions. Miro boards were utilised by the team during meetings to visualise and brainstorm elements of the project, such as user personas and the problem statement. This allowed for team collaboration both online and offline, with quality output to be referred to when necessary.

### 7.2.3 SWOT Analysis

At the beginning of the project, and again entering the second semester, a SWOT (Strengths, Weaknesses, Opportunities, and Threats) Analysis was performed to gauge an understanding of the state of the current system, and for the team to determine which aspects of the system required the most attention and development to improve the Swift app.

### 7.2.4 Fortnightly Advisory Meetings

Regular meetings were scheduled and held with the project’s champion, Dr Richard Gault, in which the team shared updates on project progress, any challenges faced, or questions they had. These meetings proved very beneficial in guiding the project’s direction and maintaining continuous progress. As the project’s workload grew, these meetings became more frequent, often happening weekly.

### 7.2.5 GitLab

The team used GitLab as the version control software development platform for the application code. This allowed team members to collaborate through all stages of the project, and ensured all code, whether related to front-end, back-end, or the coordination game itself, was stored in one place and easily accessible. The repository was cloned to each team member’s local repository (within VS Code) so that individual code contributions could seamlessly be pushed and pulled by all team members. Code review was implemented, further detailed in Section 7.2.6.

### 7.2.6 Code Review

As the project evolved, the team agreed to implement the process of code review. This was applied through merge requests on Gitlab and involved a team member creating a separate branch within their local repository, and subsequently committing and pushing code to the branch. A merge request is then created within Gitlab, which another team member reviews and approves. Once approved, the code is merged to the main branch on the remote repository, and additional branches are deleted to keep the working tree clean. Once approved and merged, all team members can pull this code to their own local repository, ensuring they are working from the most up-to-date version. The code review process has been constructive in upholding team cohesion and version control.

### 7.2.7 Sprint Plans and Retrospectives: Azure DevOps

The team held weekly sprint planning sessions that provided a designated space in which team members could share what they had been working on in the previous week, what they planned to work on in the upcoming week, and any challenges faced/questions to raise. Task allocation to each member was recorded within the team’s Azure DevOps board. By keeping a record of the weekly sprint plans, the team was able to reflect on work previously completed, as well as maintain a backlog of upcoming items. The team found that shorter sprint plans were valuable as it kept team members up to date on the progress of all aspects of the project and allowed for collaborative work.

## 7.3 Challenges Faced

One of the first challenges the team faced was understanding the development environment and software, which resulted in a significant learning curve from all team members. React Native and Expo Go were new to all team members, and many had limited experience using JavaScript. The team mitigated this challenge by devoting specific time to upskilling in these technologies, as well as learning on-the-go as the project naturally developed.

As the team originally took over an existing project, another challenge faced was understanding the subsequent code base. Issues arose surrounding outdated dependencies, and now-deprecated elements. Substantial time was devoted to attempt to integrate this code into the new system, which proved challenging. To overcome this, a total rewrite of the front and back end of the system was performed, as this provided more freedom to work outside the constraints of the previous codebase and tailor the direction of the project.

Furthermore, the use of GitLab originally proved challenging, as team members initially struggled with the concept of branches. Issues arose surrounding code on team member’s local repositories that conflicted with the remote repository. This was overcome by maintaining strong communication amongst the team when pushing/pulling code so everyone was aware of the current state of the system, as well as learning from experience how best to create merge requests.

## 7.4 Risks

A risk faced by the team involved the project taking place during the pandemic, from the aspect that a team member or their family may fall ill, as well as the covid restrictions in place. The team put measures in place to circumvent this risk, by following the pandemic guidelines in place at the given time, and incorporating a combination of online, hybrid, and when safe, in-person meetings. Team members upheld excellent communication with each other, and were flexible in working environments, as well as conducting regular health monitors to gauge the current team condition.

Furthermore, another risk the team faced was the possibility that pushing code to Git could potentially break aspects of the Swift system. To mitigate this risk, the team implemented code review within Git (as detailed in Section 7.2.6) and made it impossible to push code to the main branch on the remote repository without another team member approving and merging this code. Not only did this uphold strong coding practices, but also saved time as any bugs were spotted faster and a log of all merge requests/approvals can be accessed within the Git project.

# 8 Discussion and Conclusion

## 8.1 Summary

From taking over an existing project, the goal of the Swift app changed and developed throughout the project’s lifecycle and was ultimately successful in accomplishing its goal: a non-diagnostic rehabilitation tool, that can successfully analyse an end user’s hand-eye coordination following a stroke.

The Swift stroke rehabilitation mobile application provides an aesthetic and easy to understand user interface, in which a user can register and login to the system and play an engaging hand-eye-coordination game, that provides results assessing accuracy, time taken, engagement, and how this differs from previous game results. The user can send these results to their allocated medical professional, to draw further analysis, and provide both assistance and reassurance in their stroke rehabilitation journey.

In taking over an existing project, the team was successful in accomplishing all relevant previous goals for future work; any that weren’t achieved were not applicable to the new project direction. The current state of the Swift app means it is accessible on iOS and Android platforms, and HTTPS has been implemented within local development to provide a secure connection, with Section 4.3.2 detailing this further. Google authentication has the necessary code implemented and will be fully functional if set up in Google Cloud Console. Further stated in the previous future work was the possibility of implementing access tokens for authentication; this has now been achieved. Another future work goal was the deployment of the back end to the public cloud, as a containerised solution. The current state of the Swift app is ready for deployment, with further detail on this in Section 8.4. It was also stated that future work should involve improving the explainability of the system to avoid confusion for users; this has been achieved as a tutorial and explanations of results have been implemented, and these areas received positive user feedback during alpha and beta testing, detailed in Sections 6.1 and 6.2.

Overall, the goal of the project was achieved, with a plethora of features being implemented within the Swift app. All requirements, both functional and non-functional have been fulfilled, with Future Work, detailed in Section 8.6, specifying further improvements that could be made to the project, if more time was allocated.

## 8.2 Impact

### 8.2.1 Social Considerations

It was identified early within the project that the Swift app has the ability to provide reassurance to the end user on their rehabilitation progress, and act as a bridge between medical professional and patient. As much as this social impact is a positive, it must be stressed that the Swift app is by no means a replacement for a medical professional and is only designed to enhance the rehabilitation process. The team considered this and have stated within the app that this is simply an advisory and analysis tool and should not be used as a medical diagnosis.

Another social consideration that the team considered was how elderly people may be less confident in operating a smartphone than others. The team attempted to remedy this by ensuring the front-end is as user friendly as possible, and easy to understand. The app’s compatibility with both iOS and Android accentuates this accessibility, and allows for different device platforms and sizes, for ease of use.

### 8.2.2 Security & Privacy

Security and privacy was taken into consideration during each part of the development process. Threats that could affect the application adversely were researched, so that they could be mitigated against and, where possible, prevented. These threats include, but are not limited to, SQL Injection attacks, man-in-the-middle attacks, and DDoS attacks. Section 4.3 details how these threats have been mitigated against. That being said, there are some potential security threats that still exist as risks within the system, including secrets being stored as plaintext within the code or as environment variables within the ‘docker-compose’ file. This could not be fully mitigated, as the system would have to be deployed to use a secure key storage such as Azure Key Vault [52]. The same issue occurs with HTTPS, as the SSL certificate is only a local dev certificate, which React Native does not trust and so the application cannot communicate with the back end over HTTPS. Token authentication has been implemented and exists in the application, which is explained in section 4.3.4. Due to time constraints, the team was unable to implement any extensions of token authentication, for example, secret changes and blacklisting of invalid tokens.

Furthermore, the application does not follow GDPR regulations. For example, a data deletion and data retrieval endpoint would have to be added so a user can delete/retrieve their data upon request. A privacy policy has also not been created for the Swift app and would have to be written before the system could be deployed and published on an app store.

## 8.3 Maintenance

The back end of the system has been containerised using Docker and orchestrated using ‘docker-compose’ to allow it to be deployed to most major public cloud providers. Due to every service being a container, a “—scale” flag can be added to a ‘docker-compose’ command to increase the number of instances of a certain container if necessary. An example of a multi container orchestration on the public cloud is the “Azure multi-container group”, where a docker-compose deployment project can be published [53].

For maintenance purposes, ReadMe files have been updated to explain what the code does and how to run it with any deprecated code being put aside to ensure it is not mistaken for current code. The codebase has been commented well to guarantee that any algorithms can be understood. The code for Google authentication has been explained, and the configuration needs to be updated to work with a new Google Cloud Console application.

To keep dependencies at the correct versions, the back-end Dockerfiles specify any dependencies for the Python code. In C#, dependencies are managed as NuGet packages and versions are stored in the ‘.csproj’ file. They will stay constant until the application is built again and will not automatically update unless an explicit command is run to update these packages. Finally, dependency maintenance in the JavaScript front-end is achieved through the use of a ‘yarn.lock’ file that explicitly states which packages are necessary and what versions should be used. To rebuild and install new dependencies, a ‘yarn install’ command needs to be run. How to run the code and manage dependencies has been explained in further detail in the ReadMe files within the codebase.

## 8.4 Reflection

Communication was a key strength of the team, with clear communication channels established from the outset, as discussed in Section 7.2.2. The team documented a set of ‘Working Agreements’ outlining the core values and behaviours that should be adopted throughout the project to hold each other accountable for meeting arrangements and maintaining regular communication. The team was aware of the importance of hybrid working during a global pandemic, and so were flexible to conduct meetings online to ensure all team members could be present. Individuals were assigned primary responsibilities that aligned to their skillsets which ensured the workload was balanced and made the delegation of tasks in weekly sprints a much smoother process. Despite individuals having their own responsibilities, paired programming sessions proved successful in bringing the team together to resolve any coding roadblocks faced. Code review was a beneficial coding practice to help identify bugs or errors before code was merged to the main branch, however, this was only put in place halfway through the project and should have been implemented from the outset. Reflecting upon the work completed last semester, the team concurred that weekly sprints would be more effective as more specific tasks could be added to the sprint backlog, which gave a greater insight into how the project was progressing each week.

Before beginning the project, the team had no prior experience in the React Native framework which proved a challenge when taking on the existing codebase. Furthermore, the team had to develop skills in both C# and Docker for backend development, which also required additional research beyond the course content. In the early stages of the project, the team struggled to focus on a clear project direction which meant individuals did not have a shared understanding of the ultimate project goal. It would have been beneficial to streamline priorities earlier in the development process so that focus was given to the key project deliverables, rather than a range of potential solutions. Regarding the development of the coordination game, the major challenge lay in deciding upon which metrics should be used to measure a user’s coordination. It proved difficult to measure the difference between the centres of the circle and target to calculate the user’s accuracy as the X and Y coordinates provided by the Circle component were not at the centre of the target. The team completed several paired programming sessions in which the root of the problem was discovered, helping to resolve the issue.

## 8.5 Future Work

### 8.5.1 Improvements to the Current Coordination Game

As discussed in Section 6.3.2, multiple suggestions arose for improvements to user engagement within the current game. These included using a combination of sound and animations, as well as allowing the user to choose their own difficulty. This allows for user preference, as some users of the application may not want to be constantly challenged, and some days may want to play an easier level which is not as challenging. This could be implemented, for example, with a slider, before playing the game where the user could either play with their generated difficulty, or a manually chosen difficulty.

The addition of multiple game levels (Easy, Medium, Hard) is another potential means of allowing the user to choose how their difficulty is scaled. If a user chooses the easy level, and improves their game results, then their difficulty may only increase by one, whereas if they choose the hard level, their difficulty would increase by five. This would have to be implemented within the Python analysis service and a new value for the level would be stored in the database.

### 8.5.2 Additional Rehabilitation Assessments

One component that would have been beneficial to the front-end of the application is the addition of a second hand-eye coordination game. Initial designs were underway for this additional game, however due to time constraints this was migrated to future work. The idea for the additional game involved the user clicking a target that would appear in a random position on the screen, as many times as possible before a countdown timer ran out. When the user chooses to play the hand-eye coordination game, the application would randomly select one of the two games for the user to play. This would decrease the chances of the user becoming disinterested in completing their daily hand-eye coordination assessments, and increase their engagement.

In addition to the strength and hand eye coordination assessments, a speech test and facial drop detection could be implemented to store further results on the rehabilitation of a user. Implementing these improvements would give the medical professional a more well-rounded illustration of the user’s rehabilitation, and whether they are improving.

### 8.6.3 Mitigating Risks that Currently Exist in the System

A major risk that found during alpha testing was that if the back end goes down at any time when a user is logged into the application, the application will crash due to the lack of specific exception handling. Most of the time, there is a catch clause in the fetch statements, which will log the user out of the application and inform them they cannot currently access the application. The best way to prevent this from happening is to have a handler which searches for a HTTP 404 error from the server.

If a user’s password is compromised, or the application was subjected to a DDoS attack, there currently is no way of checking if the request is coming from a valid source. If the application had a way of checking which IP address a request comes from, it could ask the user to complete two-factor authentication when a new IP address is found. To mitigate a DDoS attack, if many requests are coming from the same IP address, that IP address could be blocked for a certain amount of time to prevent the system f.

Another key risk to the application is that if a malicious actor is able to access a valid authentication token. They would then be able to retrieve data for any user in the application if they knew their username or email. Authentication checks would have to be implemented to see which token corresponds to which user, ensuring they do not have access to the whole database.

### 8.6.4 Publishing to the App and Play Store

Once development of the Swift app is completed, it can be published to the Apple App Store and Google Play Store, for users to download the app to their devices. To accomplish this, the application would need to comply with GDPR regulations as well as the Apple and Google regulations.

### 8.6.5 Deploying to the Public Cloud Using Kubernetes

For the Swift app to be published to the public cloud, it is stated in Section 8.4 that this can be achieved using a multi-container orchestration with ‘docker-compose’. However, to allow for automatic scaling and load balancing on the public cloud, the application would have to be orchestrated using Kubernetes. A Kubernetes ‘swiftusermanagementapi-deploy.yaml’ already exists in the repository, however, it would need to be updated and optimised to ensure that the application functions to its highest standard. To further optimise the application, a load balancer would have to be created as a separate entity from the User Management Service to route traffic more efficiently.

# References

|  |  |
| --- | --- |
| [1] | D. Z. G. &. K. R. Rand, “Rehab-let: touchscreen tablet for self-training impaired dexterity post stroke: study protocol for a pilot randomized controlled trial,” Trials, TelAviv, 2015. |
| [2] | NHS, “The NHS Long Term Plan,” NHS, London, 2019. |
| [3] | A. B. V. K. D. Patel, “Current, future and avoidable costs of stroke in the UK,” Stroke Association, London, 2017. |
| [4] | K. L. N. S. S. K. J. W. C. R. T. &. T. W. W. Gao, “Eye-hand coordination and its relationship with sensori-motor impairments in stroke survivors,” Journal of rehabilitation medicine, 2010. |
| [5] | J. M. W. D. A. Harris, “A Self-Administered Graded Repetitive Arm Supplementary Program (GRASP) Improves Arm Function During Inpatient Stroke Rehabilitation,” University of British Columbia, Vancouver, 2009. |
| [6] | R. B. S. Barker, “Upper limb recovery after stroke: The stroke survivors' perspective, Disability and Rehabilitation,” 2009. |
| [7] | M. R. T. J. D. D. D. Pugliese, “Mobile tablet-based therapies following stroke: A systematic scoping review of administrative methods and patient experiences,” Ottawa, 2018. |
| [8] | R. T. S. R. M. K. Z. L. Pugliese M., “RecoverNow: A mobile tablet-based therapy platform for early stroke rehabilitation,” Ottawa, 2019. |
| [9] | Google Play, “Train your brain. Coordination,” Senior Games, 13 December 2022. [Online]. Available: https://play.google.com/store/apps/details?id=com.tellmewow.senior.coordination&hl=en&gl=US&pli=1. [Accessed 2022]. |
| [10] | NHS, “My Therappy,” Dexteria, 2022. [Online]. Available: https://www.my-therappy.co.uk/index.php?p=app/dexteria. [Accessed 8 October 2022]. |
| [11] | “Figma,” Figma, 2022. [Online]. Available: https://www.figma.com/. [Accessed 14 December 2022]. |
| [12] | Balsamiq, “Balsamiq,” Balsamiq, 2022. [Online]. Available: https://balsamiq.com/. [Accessed 14 December 2022]. |
| [13] | NHS, “NHS Stroke Overview,” NHS, 2022. [Online]. Available: https://www.nhs.uk/conditions/stroke/. [Accessed 14 December 2022]. |
| [14] | K. C. N. R. M. M. S. Salter, “Outcome Measures in Stroke Rehabilitation,” September 2013. [Online]. Available: http://www.ebrsr.com/sites/default/files/Chapter%2020\_Outcome%20Measures.pdf. [Accessed 15 October 2022]. |
| [15] | Physiopedia, “Nine-Hole Peg Test,” 2022. [Online]. Available: https://www.physio-pedia.com/Nine-Hole\_Peg\_Test. [Accessed 15 October 2022]. |
| [16] | Amazon Web Services, “What is Service-Oriented Architecture?,” Amazon Web Services, 2022. [Online]. Available: https://aws.amazon.com/what-is/service-oriented-architecture/. [Accessed 25 November 2022]. |
| [17] | IBM, “SOA (Service-Oriented Architecture),” IBM, 7 April 2021. [Online]. Available: https://www.ibm.com/cloud/learn/soa. [Accessed 17 November 2022]. |
| [18] | Microsoft, “Common web application architectures,” 11 January 2022. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures. [Accessed 2022 November 18]. |
| [19] | S. Oloruntoba, “SOLID: The First 5 Principles of Object Oriented Design,” 21 September 2020. [Online]. Available: https://www.digitalocean.com/community/conceptual-articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design. [Accessed 18 November 2022]. |
| [20] | B. Joshi, “SOLID Principles,” in *Beginning SOLID Principles and Design Patterns for ASP.NET Developers*, Pitruchhaya, Springer, 2016, pp. 1-85. |
| [21] | C. A. V. Solera, “Clean Architecture vs MVVM,” 17 December 2020. [Online]. Available: https://www.linkedin.com/pulse/clean-architecture-vs-mvvm-carlos-andrey-valverde-solera/. [Accessed 2022 November 18]. |
| [22] | Expo Go, “Expo Go Documentation,” Expo, 2022. [Online]. Available: https://docs.expo.dev/workflow/expo-go/. [Accessed 14 December 2022]. |
| [23] | Flutter, “Flutter Supported Platforms,” Flutter, 2022. [Online]. Available: https://docs.flutter.dev/development/tools/sdk/release-notes/supported-platforms. [Accessed 14 December 2022]. |
| [24] | TutorialsPoints, “Xamarin Tutorial,” Xamarin, 2022. [Online]. Available: https://www.tutorialspoint.com/xamarin/index.htm#:~:text=Xamarin%20is%20a%20software%20company,NET%20Framework.. [Accessed 14 December 2022]. |
| [25] | A. Krishna, “Yarn vs NPM: Which Package Manager is Best to Choose?,” 15 June 2022. [Online]. Available: https://www.knowledgehut.com/blog/web-development/yarn-vs-npm. [Accessed 10 December 2022]. |
| [26] | S. Mansion, “React Native Gesture Handler,” 2022. [Online]. Available: https://docs.swmansion.com/react-native-gesture-handler/docs/api/gestures/pan-gesture/. [Accessed 20 November 2022]. |
| [27] | Software Mansion, “React Native Reanimated,” 2022. [Online]. Available: https://docs.swmansion.com/react-native-reanimated/docs/1.x.x/. [Accessed 20 November 2022]. |
| [28] | npm, “react-native-svg,” 15 November 2022. [Online]. Available: https://www.npmjs.com/package/react-native-svg. [Accessed 20 November 2022]. |
| [29] | npm, “victory-native,” 15 September 2022. [Online]. Available: https://www.npmjs.com/package/victory-native. [Accessed 21 November 2022]. |
| [30] | npm, “react-native-simple-radio-button,” 2019. [Online]. Available: https://www.npmjs.com/package/react-native-simple-radio-button. [Accessed 21 November 2022]. |
| [31] | Amazon Web Services, “What is Docker?,” Amazon Web Services, 2022. [Online]. Available: https://aws.amazon.com/docker/. [Accessed 24 November 2022]. |
| [32] | Docker, “Docker Compose Overview,” Docker, 2022. [Online]. Available: https://docs.docker.com/compose/. [Accessed 25 November 2022]. |
| [33] | RabbitMQ, “RabbitMQ,” 2022. [Online]. Available: https://www.rabbitmq.com/. [Accessed 28 November 2022]. |
| [34] | Portainer, “Powerful Container Management for DevSecOps,” Portainer.io, 2022. [Online]. Available: https://www.portainer.io/. [Accessed 28 November 2022]. |
| [35] | pgAdmin, “Container Deployment Documentation,” pgAdmin, 2022. [Online]. Available: https://www.pgadmin.org/docs/pgadmin4/latest/container\_deployment.html. [Accessed 28 November 2022]. |
| [36] | Q. Eric and J. Max, *A performance comparison on REST-APIs in Express.js, Flask and ASP.NET Core,* VÄSTERÅS: DiVA, 2022. |
| [37] | Microsoft, “CQRS pattern,” Microsoft, 2022. [Online]. Available: https://learn.microsoft.com/en-us/azure/architecture/patterns/cqrs. [Accessed 10 December 2022]. |
| [38] | Microsoft, “Implement the microservice application layer using the Web API,” 9 September 2022. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/microservice-application-layer-implementation-web-api. [Accessed 10 December 2022]. |
| [39] | OWASP, “SQL Injection,” OWASP, 2022. [Online]. Available: https://owasp.org/www-community/attacks/SQL\_Injection. [Accessed 10 December 2022]. |
| [40] | Npgsql, “Prepared Statements,” 2022. [Online]. Available: https://www.npgsql.org/doc/prepare.html. [Accessed 10 December 2022]. |
| [41] | RabbitMQ, “TLS Support,” RabbitMQ, 2022. [Online]. Available: https://www.rabbitmq.com/ssl.html. [Accessed 10 December 2022]. |
| [42] | Help Net Security, “Increase in credential phishing and brute force attacks causing financial and reputational damage,” Help Net Security, 31 August 2021. [Online]. Available: https://www.helpnetsecurity.com/2021/08/31/increase-in-credential-phishing/. [Accessed 10 December 2022]. |
| [43] | D. Arias, “Hashing in Action: Understanding bcrypt,” 25 February 2021. [Online]. Available: https://auth0.com/blog/hashing-in-action-understanding-bcrypt/. [Accessed 10 December 2022]. |
| [44] | D. Arias, “Adding Salt to Hashing: A Better Way to Store Passwords,” 25 February 2021. [Online]. Available: https://auth0.com/blog/adding-salt-to-hashing-a-better-way-to-store-passwords/. [Accessed 10 December 2022]. |
| [45] | Swagger, “What Is OpenAPI?,” SmartBear, 2022. [Online]. Available: https://swagger.io/docs/specification/about/#:~:text=What%20Is%20Swagger%3F,you%20can%20write%20OpenAPI%20definitions.. [Accessed 11 December 2022]. |
| [46] | PyCharm, “PyCharm Features,” JetBrains, 2022. [Online]. Available: https://www.jetbrains.com/pycharm/features/. [Accessed 10 December 2022]. |
| [47] | S. A. G. Panagia, “What is Expo and why it matters for app development,” Moze, 21 February 2022. [Online]. Available: https://www.mozestudio.com/journal/what-is-expo-and-why-it-matters-for-app-development/. [Accessed 10 December 2022]. |
| [48] | Testim, “React Native Unit Testing: A Complete Getting Started Guide,” 18 March 2020. [Online]. Available: https://www.testim.io/blog/react-native-unit-testing/. [Accessed 11 December 2022]. |
| [49] | XUnit.net, “Comparing xUnit.net to other frameworks,” XUnit.net, 2022. [Online]. Available: https://xunit.net/docs/comparisons.html. [Accessed 11 December 2022]. |
| [50] | D. Shupli, D. Cazzulino, J. Meng, I. Shimko and F. Salvali, “moq/moq4,” github, 6 December 2022. [Online]. Available: https://github.com/moq/moq4. [Accessed 11 December 2022]. |
| [51] | SpecFlow, “Welcome to SpecFlow’s documentation!,” Tricentis, 2022. [Online]. Available: https://docs.specflow.org/projects/specflow/en/latest/. [Accessed 11 December 2022]. |
| [52] | Microsoft, “Key Vault,” Microsoft, [Online]. Available: https://azure.microsoft.com/en-gb/products/key-vault/. [Accessed 12 December 2022]. |
| [53] | tomvcassidy, v-alje, hyoshioka0128, v-kents and zr-msft, “Tutorial: Deploy a multi-container group using Docker Compose,” Microsoft, 17 June 2022. [Online]. Available: https://learn.microsoft.com/en-us/azure/container-instances/tutorial-docker-compose. [Accessed 2022 December 12]. |
| [54] | L. Johansson, “FAQ: What is AMQP and why is it used in RabbitMQ?,” 21 November 2019. [Online]. Available: https://www.cloudamqp.com/blog/what-is-amqp-and-why-is-it-used-in-rabbitmq.html#:~:text=Advanced%20Message%20Queuing%20Protocol%20(AMQP)%20is%20an%20application%20layer%20protocol,regardless%20of%20the%20technology%20used.. [Accessed 2022 November 22]. |

# Bibliography

[1]

BillWagner, “.NET documentation,” learn.microsoft.com. https://learn.microsoft.com/en-us/dotnet/

[2]

“Welcome to the Step-By-Step Getting Started Guide! — documentation,” docs.specflow.org. https://docs.specflow.org/projects/getting-started/en/latest/index.html (accessed Dec. 15, 2022).

[3]

“Introduction to Expo,” Expo Documentation. https://docs.expo.dev/

[4]

“Documentation: Table of Contents — RabbitMQ,” www.rabbitmq.com. https://www.rabbitmq.com/documentation.html

[5]

Docker, “Overview of Docker Compose,” Docker Documentation, Feb. 10, 2020. https://docs.docker.com/compose/

# Appendices

# Appendix 1: Front-End Design Additional Wireframes

Graphical user interface, text, application

Description automatically generatedA picture containing diagram

Description automatically generatedGraphical user interface

Description automatically generatedA picture containing diagram

Description automatically generatedGraphical user interface

Description automatically generated with low confidenceDiagram

Description automatically generatedDiagram

Description automatically generated

**Appendix 2: Colourblind Safe Colour Palette**

Bar chart

Description automatically generated

## Appendix 3: Additional Dependencies within Swift App

|  |  |  |  |
| --- | --- | --- | --- |
| **Relating Requirement** | **Dependency Imported** | **Version** | **How is it implemented/what is it used for?** |
| FR1.0, NFR1.2-1.3 | *@expo-google-fonts/roboto* | ^0.2.2 | Custom subtitle and regular text fonts throughout all screens |
| R1.0, NFR1.2-1.3 | *@expo-google-fonts/yeseva-one* | ^0.2.2 | Custom title fonts throughout all screens |
| FR1.0, NFR 1.1 | *@react-navigation/material-bottom-tabs* | ^6.2.4 | Navigation throughout the screens of the Swift app, the order screens generate in upon loading the the app |
| FR1.0, NFR 1.1 | *@react-navigation-stack* | ^6.9.1 |
|  | *@expo-app-loading* | ~2.1.0 | Deals with error handling – upon an error the app will not crash, but will instead display the splash screen |
| FR12.0 | *@expo-mail-composer* | ^11.3.0 | Email results feature of the app used on *HomeScreen.js* and send feedback feature used on *HelpScreen.js* |
| *@expo-print* | ^11.3.0 |  |
| FR1.0 (and error handling? | *@expo-splash-screen* | ~0.16.2 | Generates and shows the designed splash screen for the Swift app upon each load/reload |
| FR1.0, NFR1.2, 1.5 | *@expo-status-bar* | ~1.4.0 | Assists in styling dimensions between Android and iOS devices when necessary |
| FR1.0, NFR1.4 | *@react-native-event-listeners* | ^1.0.7 | Necessary for the toggle switch for changing between different colour schemes e.g. light vs dark mode |
| FR1.0, NFR1.4 | @react-native-vector-icons | ^9.2.0 | All icons within the Swift app |
| FR1.0, NFR1.2-1.3 | *@expo-google-fonts/roboto* | ^0.2.2 | Custom subtitle and regular text fonts throughout all screens |

## Appendix 4: Additional Python Dependencies

|  |  |  |  |
| --- | --- | --- | --- |
| **Relating Requirement** | **Dependency Imported** | **Version** | **How is it implemented/what is it used for?** |
| NFR11.1 | *Pika* | 1.3.0 | Used with RabbitMQ server |
| N/A | *Numpy* | 1.19.3 | Adding support for arrays |
| N/A | *opencv-python* | 4.5.4.58 | Used in computer vision model |
| N/A | *protobuf* | 3.19.1 | Used for protocol buffer |
| N/A | *mediapipe* | 0.8.3 | Used in computer vision model |
| N/A | *imageio* | 2.10.3 | Provides interface to read and write image data |
| N/A | *scikit-fuzzy* | 0.4.2 | Used for “fuzzy” logic algorithms |
| N/A | *scikit-image* | 0.17.2 | *Image processing library* |
| N/A | *scipy* | 1.5.4 | Used for scientific programming |
| N/A | *scikit-learn* | 0.24.2 | Machine learning library |

## Appendix 5: Additional C# Dependencies

|  |  |  |  |
| --- | --- | --- | --- |
| **Relating Requirement** | **Dependency** | **Version** | **How is it implemented/ what is it used for?** |
| NFR 1.4 | FluentValidation.  AspNetCore | 11.2.2 | Validating data entering the system |
| NFR 11.3 | MassTransit | 8.0.5 | Sending messages onto the RabbitMQ messaging broker |
| N/A | MediatR | 10.0.1 | Used to pass data from the API layer to the application layer and implement the mediator pattern. |
| NFR 5.3 | Microsoft.AspNetCore.  Authentication.  JwtBearer | 6.0.7 | Used to generate JWT authentication tokens |
| N/A | Moq | 4.18.2 | Used to mock out dependencies in tests |
| FR 3.0, FR 9.0 | npgsql | 6.0.6 | Used to connect to the database and execute SQL statements |
| NFR 11.3 | RabbitMQ.Client | 6.4.0 | Used to connect to the RabbitMQ messaging broker |
| N/A | Swashbuckle.AspNetCore | 6.2.3 | Used to create an API |
| NFR 5.3 | System.IdentityModel.  Tokens.Jwt | 6.21.0 | Used for JWT tokens |
| N/A | AutoMapper | 11.0.1 | Used for mapping between classes |
| NFR 1.4 | FluentValidation.  DependencyInjectionExtensions | 10.0.1 | Used for dependency injection with validation |
| N/A | MediatR.Extensions.Microsoft.  DependencyInjection | 10.0.1 | Used for dependency injection with MediatR |
| N/A | Microsoft.AspNetCore.Http.  Features | 5.0.17 | Used for HTTP connections |
| NFR 5.2 | Microsoft.Extensions.  Configuration.Abstractions | 3.1.18 | Used for utilising the configuration which can retrieve data from azure key vault, appsettings.json or env variables. |
| N/A | Microsoft.Extensions.  Logging.Abstractions | 3.1.18 | Used for logging out to the console |
| NFR 3.1 | BCrypt.Net-Next | 4.0.3 | Used for hashing passwords |
| N/A | Microsoft.Extensions.Hosting.  DependencyInjection | 10.0.1 | Used for depenecy injection |
| N/A | Newtonsoft.Json | 13.0.1 | Used for JSON |
| N/A | FluentResults | 3.11.0 | Used for testing |
| N/A | Microsoft.NET.Test.sdk | 17.3.1 | Used for testing |
| N/A | SpecFlow.Tools.MsBuild  .Generation | 3.9.74 | Used for generating specflow files |
| N/A | Specflow.xUnit | 2.4.2 | Used for testing |
| N/A | Xunit | 2.4.2 | Used for testing |
| N/A | Xunit.runner.visualstudio | 2.4.5 | Used for running tests |

## Appendix 6: External Docker Image Versions

|  |  |
| --- | --- |
| Docker image | What it’s used for |
| postgres | Used for the database |
| dpage/pgadmin4 | Used for database management |
| portainer/portainer-ce | Used for container management |
| rabbitmq:3-management-alpine | Used for messaging between services |

## Calendar Description automatically generated with medium confidenceAppendix 7: TimeTree Calendar

## Appendix 8: Acceptance Tests

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test ID** | **Related Requirement** | **Scenario** | **Given** | **When** | **Then** | **Result** |
| 1 | 1.1 | Navigating back from different screens within the Swift app | The user has navigated from one screen to another | The user clicks on the “back” button | The user is taken to the previous screen that had been accessed | PASS |
| 2 | 1.1 | Navigating around the Swift app | The user is logged into their account | They click any navigation buttons | They will be taken to the relevant screen | PARTIAL PASS |
| 3 | 1.1 | Navigating back from the coordination result screen | The user has completed the coordination game and is being shown their results | They click on the back button on the Coordination Result screen | The user is taken to the Home Screen, not the game screen | PASS |
| 4 | 1.2 | Ensuring all font sizes are correct | The user is logged into their account | They interact with the Swift app | All titles, subtitles and regular text are the same font size across screens | PASS |
| 5 | 1.2 | Ensure all logo images (excluding Home Screen) are the same size | The user is logged into their account | They interact with the Swift app | All logo images are the same size | PASS |
| 6 | 1.4 | Swapping from light mode to dark mode and vice versa | The user is currently in either light mode or dark mode | The user clicks on the toggle for “Dark Mode” | This will swap the colour scheme of the Swift app to the dark mode setting, or back to the original light mode setting | PARTIAL PASS |
| 7 | 1.5 | Accessing the Swift app on an Android device | The user has access to an Android smartphone | The user opens the Swift app | The user should be able to fully interact with the Swift app | PASS |
| 8 | 1.6 | Accessing the Swift app on an iOS device | The user has access to an iOS smartphone | The user opens the Swift app | The user should be able to fully interact with the Swift app | PASS |
| 9 | 2.0 | Registering the user to the Swift app with valid data | All data required to register a user is present and valid, and has not been used to register an existing user | The valid data is entered into all available user input boxes | The new user should be registered with the Swift app, taken to the Home Screen, and be able to then log in with these details in the future | PASS |
| 10 | 2.0 | Registering the user to the Swift app with invalid data | The data entered to register a user is not valid | The invalid data is entered into the available user input boxes | An alert is displayed to the user explaining that the data they have entered is invalid, and they should be prompted to re-enter | PASS |
| 11 | 2.0 | Registering the user to the Swift app with incomplete data | The user wants to register with the system | Data is not entered in all the required user input boxes | An alert is displayed to the user explaining that all required fields have not been filled out, and they should be prompted to re-enter | PASS |
| 12 | 4.0 | Logging the user into the Swift app with valid data | The user is logged out of the system, they have already registered to the Swift app, and all data required to log in the user is present and valid | The valid data is entered into all available user input boxes | The user credentials should be confirmed, they are logged into the system and taken to the Home Screen | PASS |
| 13 | 4.0 | Logging the user into the Swift app with invalid data | The data entered to login the user is not valid | The invalid data is entered into the available user input boxes | An alert is displayed to the user explaining that the data they have entered is invalid, and they should be prompted to re-enter | PASS |
| 14 | 4.0 | Logging the user into the Swift app with incomplete data | The data entered to login the user is incomplete | Data is not entered in all the required user input boxes | An alert is displayed to the user explaining that all required fields have not been filled out, and they should be prompted to re-enter | PASS |
| 15 | 5.0 | Logging out the user from the Swift app | The user wants to log out of their account, whilst currently logged in | The user navigates to Settings and clicks the Logout button | An alert is displayed to the user confirming they want to logout of the system: if the user confirms they want to logout, they will be logged out of their account and taken to the Sign In Screen | PASS |
| 16 | 6.0 | Accessing the coordination game | The user is logged into the system | The user clicks on the coordination game icon from the Home Screen | The user is taken to the game screen and initially shown a tutorial pop up which explains how to play the coordination game. The user can then close this pop up and play the game. | PASS |
| 17 | 6.2 | Playing the coordination game | The user is logged into the system, has read the game tutorial that appears upon entering the game, and clicked off it | The user interacts with the game | The user can drag the ball through the randomly generated obstacles, and none of the obstacles overlap | PASS |
| 18 | 6.3 | Playing the coordination game at different levels of difficulty | The user is logged into the system, has read the game tutorial that appears upon entering the game, and clicked off it | The user plays the game | Based on the user’s interactions with the game, different difficulty levels will be set – the harder the difficulty, the more obstacles are clearly shown | PASS |
| 19 | 7.1 | Understanding coordination game results | The user plays the coordination game | The user successfully places the circle in the target and completes the game | The user will be automatically taken to a Game Results screen that will show a pop-up explanation of what the results mean. | PASS |
| 20 | 7.1 | Viewing the coordination game results | The user plays the coordination game | The user successfully places the circle in the target to complete the game, and selects ‘View my Results’ button from the pop up | The user is shown statistics for their accuracy and time taken, along with the percentage change in results from the last time they played the game | PASS |
| 21 | 10.1 | Viewing previous game scores | The user has played multiple coordination games over time | The user clicks into View Results from the Home Screen | The user is shown text that is clear and easy to read explaining the two different charts available to view results. From here, the user can click into either Daily Progress or Summary Results to view the graphs | PASS |
| 22 | 10.1 | Viewing average daily game scores | The user has played multiple coordination games over time | The user clicks into View Results from the Home Screen, and then clicks the Daily Progress button | A line chart will be displayed to the user, in which they can view their average coordination game results from the last seven days, with the option to switch between viewing their average accuracy or time taken. | PASS |
| 23 | 10.1 | Viewing average weekly game scores | The user has played multiple coordination games over time | The user clicks into View Results from the Home Screen, and then clicks the Weekly Summary button | A radar chart will be displayed to the user, in which they can view their average coordination game results from the last week | PASS |
| 24 | 12.0 | Sending an email to the user’s medical professional that contains their game results | The user is logged into the Swift app and has played the coordination game | The user clicks on the “Send Email” feature from the Home Screen | An email pop up will be displayed that contains their game results, in which the user can then send these results to their medical professional | PASS |
| 25 | 12.1 | Editing an email to the user’s medical professional that contains their game results | The user is logged into the Swift app and has played the coordination game | The user clicks on the “Send Email” feature from the Home Screen | An email pop up will be displayed that already has a pre-written subject line and body, which can be edited by the user if they wish | PASS |

## Appendix 9: Alpha Testing

**Question 1:** How would you rate the ability to register an account with Swift?

|  |  |  |
| --- | --- | --- |
| **Tester ID** | **Rating (1-10)** | **Comments** |
| 1 | 10 | The registration process is straightforward and requires minimal effort |
| 2 | 10 | Users can easily create an account and minimal information is required |
| 3 | 10 | It is quick and easy to create an account |
| 4 | 10 | The register screen is clearly laid out with the right amount of text boxes which makes it easier to register. I like how when the user doesn’t enter a password the ‘Alert’ message appears specifying reasons why this might be. I like how it is clear when specifying a password, the alert lets the user know what it must include if not included |
| 5 | 10 | Simple process to register an account |
| 6 | 9 | Quite a lot of whitespace at the bottom of the screen so input fields should be spread out more |
| 7 | 9 | Would be better if the alert specified what the error was |

**Question 2:** How would you rate the ability to login with Swift?

|  |  |  |
| --- | --- | --- |
| **Tester ID** | **Rating (1-10)** | **Comments** |
| 1 | 10 | Logging in is straightforward and user friendly |
| 2 | 10 | The login process is smooth and easy to understand |
| 3 | 9 | The login screen has a simple design that is easy to follow, but the alert when there are invalid details should be more detailed as to what the error is |
| 4 | 10 | Similar to registering I like how the alert box appears right away when an error has occurred. Login is a very simple and easy-to-follow process. |
| 5 | 8 | Upon selecting the login button, the text on the button then displays ‘click to start’ rather than taking you to the home screen. It would be better to only have to click the login button once |
| 6 | 7 | The forgot your password option is displayed but isn’t yet working |
| 7 | 9 | Smooth process of logging in, would be useful if the alert specified which error occurred instead of just saying the details are invalid |

**Question 3:** How would you rate the navigation between screens?

|  |  |  |
| --- | --- | --- |
| **Tester ID** | **Rating (1-10)** | **Comments** |
| 1 | 10 | Screens flow well and everything links up together nicely |
| 2 | 10 | Bottom tab navigator is good, everything links up well with back buttons going to home screen |
| 3 | 10 | Back buttons going straight to home screen is good, so you don’t have to repeatedly click the back button to get to the home screen |
| 4 | 10 | Central point of navigation from the home screen which is easy to understand |
| 5 | 10 | Easy navigation between screens that takes you to the expected screen |
| 6 | 9 | The back button could be placed in the bottom left for easier access when holding the phone |
| 7 | 9 | The back button on screens looks slightly out of place and could be a bit higher up |

**Question 4:** How would you rate the layout of the user interface - is it easy to understand? Is the styling consistent across screens?

|  |  |  |
| --- | --- | --- |
| **Tester ID** | **Rating (1-10)** | **Comments** |
| 1 | 9 | Colour scheme is nice and easy to understand UI, logout button could be in a better position on the home screen – maybe in settings |
| 2 | 10 | All screens are consistent and styling structure is followed throughout |
| 3 | 10 | Clear style sheet has been followed with colours and position of buttons, cohesive design |
| 4 | 10 | Screens are consistent with logo and headings in same position so it is easy to follow and screens are not too cluttered |
| 5 | 9 | The login and register buttons are slightly inconsistent with the buttons in the rest of the application |
| 6 | 9 | On the home screen, to access the email feature you need to select the ‘send mail’ text instead of the icon like the other home screen buttons |
| 7 | 9 | Might be useful to have an about page to explain what the app does and what it is for |

**Question 5:** How would you rate the text and button size across screens? (1 = very difficult to read, 10 = very easy to read)

|  |  |  |
| --- | --- | --- |
| **Tester ID** | **Rating (1-10)** | **Comments** |
| 1 | 9 | The axis points on the graphs had quite small font, results screen could have the text in bold to make it easier to read |
| 2 | 10 | Font size is large and easy to read |
| 3 | 9 | Bottom tab icons should be bigger so they are easy to see |
| 4 | 10 | Icons are easy to understand and buttons are large, text is clear and easy to read |
| 5 | 9 | The text in the results screens might be a bit small for those with visual impairments, could maybe have an option to change the font size within the app |
| 6 | 9 | The text on the main part of the home screen is very easy to read; however the text size on the tab bar is a little small- not hard to read, but it could be a little bigger. The button size is very easy to read, and it is clear what it is doing. |
| 7 | 10 | The buttons and text are clearly legible and well-spaced on the screens |

**Question 6:** How would you rate the dark mode style? Are the colours contrasting enough?

|  |  |  |
| --- | --- | --- |
| **Tester ID** | **Rating (1-10)** | **Comments** |
| 1 | 6 | The view results icon in the home screen doesn’t change colour like the other icons |
| 2 | 5 | The tutorial and pop-up screens are still bright in dark mode |
| 3 | 5 | Really like the dark mode, like the option to choose |
| 4 | 6 | When playing the coordination game in dark mode, you cannot see the circle |
| 5 | 5 | The weekly summary screen with the radar chart doesn’t change colour in dark mode |
| 6 | 5 | Dark mode and light mode can be easily switched between |
| 7 | 7 | Dark mode is easier on the eyes and keeps consistent colours throughout the app |

**Question 7:** How would you rate the colour scheme and layout of the game?

|  |  |  |
| --- | --- | --- |
| **Tester ID** | **Rating (1-10)** | **Comments** |
| 1 | 7 | Not sure about the obstacles being red, but I like the rest of the colours |
| 2 | 10 | I like the red obstacles because they add some contrast to the game |
| 3 | 10 | Colour scheme is good and is consistent with the rest of the app |
| 4 | 9 | The draggable circle should appear in front of the target so users don’t think it has disappeared |
| 5 | 10 | Layout of game makes game objective clear so the user knows exactly what to do, would be relatively easy to understand even without instructions |
| 6 | 9 | Layout of game is good, would be useful to have the difficulty level displayed while playing the game |
| 7 | 10 | The colour scheme is consistent with the rest of the app and the colours aren’t too overpowering |

**Question 8:** How would you rate the size of the circles/obstacles - were they clear enough?

|  |  |  |
| --- | --- | --- |
| **Tester ID** | **Rating (1-10)** | **Comments** |
| 1 | 10 | UI design is good and circles and obstacles are clear |
| 2 | 10 | Circles and obstacles are large and clear to see |
| 3 | 10 | UI has been well designed and clearly thought out |
| 4 | 8 | The circle might be too small when playing the game and may be difficult to see when dragging it with your finger |
| 5 | 9 | Would be interesting to see if you increased the size of the circle, does it hinder or help with the results. |
| 6 | 9 | As the difficulty increases you could change the circle and target sizes. |
| 7 | 10 | The size of the obstacles and circles were very clear. The size was perfect as it allowed the user to complete the game while still providing a challenge. |

**Question 9:** How clear was the tutorial in explaining the game instructions? (1 = not clear at all, 10 = very clear)

|  |  |  |
| --- | --- | --- |
| **Tester ID** | **Rating (1-10)** | **Comments** |
| 1 | 10 | The tutorial was very easy to understand |
| 2 | 9 | Could maybe use some modals to guide the user as to where they should be moving the circle or a modal that says congratulations when they have reached the endpoint |
| 3 | 10 | Very clear as it was easy to understand what the game wanted you to do |
| 4 | 10 | The tutorial was concise and easy to follow |
| 5 | 10 | The tutorial was clear and I liked that you can access it at any time while playing the game |
| 6 | 10 | Instructions were easy to follow |
| 7 | 10 | The tutorial explains the game instructions well |

**Question 10:** How quickly did the game load? (1 = very slowly, 10 = immediately)

|  |  |  |
| --- | --- | --- |
| **Tester ID** | **Rating (1-10)** | **Comments** |
| 1 | 9 | When the data is loading you should use a loading icon instead of a pop-up. |
| 2 | 9 | Game had loaded by the time I read the instructions |
| 3 | 10 | The game loaded immediately |
| 4 | 10 | The game loaded without any lagging |
| 5 | 9 | The game took a few seconds to load but I thought it was good that the alert appeared to make sure you couldn’t access the game until it was ready |
| 6 | 8 | The game loaded in a few seconds |
| 7 | 9 | The game loaded quickly and was ready by the time I had read the tutorial |

**Question 11:** How would you rate the difficulty of the game? (1 = too easy/difficult, 10 = appropriate difficulty)

|  |  |  |
| --- | --- | --- |
| **Tester ID** | **Rating (1-10)** | **Comments** |
| 1 | 10 | Difficulty changes are seamless which is very impressive |
| 2 | 10 | Easy to use for target audience |
| 3 | 10 | The difficulty changes are a good way to keep the user engaged and encourage them to progress |
| 4 | 10 | Difficulty adds variation to the game, gives users a sense of achievement when they can see their progression |
| 5 | 10 | It scales well and there is a lot of different options for every user. |
| 6 | 9 | It would be good to have the option to choose the difficulty level. |
| 7 | 8 | Difficulty is always at the maximum of someone’s ability and they can’t make it any easier for themselves |

**Question 12:** Did you enjoy playing the game? Did you find it engaging? (1 = didn’t enjoy it at all, 10 = really enjoyed it)

|  |  |  |
| --- | --- | --- |
| **Tester ID** | **Rating (1-10)** | **Comments** |
| 1 | 8 | Some sound effects would make it a bit more engaging |
| 2 | 9 | Could make the game more reactive – congratulations or an animation to make it clear the game is over |
| 3 | 8 | The game could be made more engaging by adjusting the size of the circle and target with the difficulty |
| 4 | 9 | The game was entertaining and had a good level of challenge |
| 5 | 9 | The game was enjoyable and kept my interest |
| 6 | 7 | The game could be more interactive and could have the obstacles move with a greater difficulty level |
| 7 | 9 | I think the game has been well-designed and will be enjoyable for those in the process of stroke rehabilitation |

**Question 13:** How quickly were results displayed on completion of the game? (1 = very slowly, 10 = immediately)

|  |  |  |
| --- | --- | --- |
| **Tester ID** | **Rating (1-10)** | **Comments** |
| 1 | 7 | Could have a loading animation to indicate that the results are being retrieved |
| 2 | 10 | Results were displayed immediately |
| 3 | 7 | Results were displayed relatively quickly, an alert appeared saying results weren’t loaded yet when I first clicked the button but was able to see them after that |
| 4 | 8 | Results were displayed promptly |
| 5 | 9 | The results were displayed after I clicked the show results button |
| 6 | 10 | Results were displayed straight after completing the game, I like that there is a pop-up reassuring the user that it is not a medical diagnosis before results are displayed |
| 7 | 8 | Results were displayed fairly quickly |

**Question 14:** How would you rate the scoring of the game – were you able to understand the results breakdown?

|  |  |  |
| --- | --- | --- |
| **Tester ID** | **Rating (1-10)** | **Comments** |
| 1 | 10 | The data gathered for the game includes useful metrics and is an interesting way to assess hand-eye coordination |
| 2 | 10 | Method for the game scoring is good and the layout is clear and easy to read |
| 3 | 10 | Percentages can be easier to quantify than numbers, so it helps users get a better idea of what their score actually means |
| 4 | 9 | The results breakdown included appropriate stats that were relevant to the purpose of the game, the result titles could be in bold to put more emphasis on them |
| 5 | 8 | Confused at the start whenever the percentage change is 0%, make it known why it is 0% because the user may be a bit confused. |
| 6 | 9 | The time taken is done in seconds, would be better for it to be in milliseconds |
| 7 | 10 | Percentage change over time gives a lot of insight into the user’s data |

**Question 15:** How would you rate the charts displaying the average game results – were they easy to understand? Were they clear to read?

|  |  |  |
| --- | --- | --- |
| **Tester ID** | **Rating (1-10)** | **Comments** |
| 1 | 7 | Difficult to understand what the radar chart is showing. |
| 2 | 8 | Text being sideways makes it difficult to read the radar chart. |
| 3 | 9 | Line graph screen is very good, could have a function to change the dates on the graph so that you can look further into the past |
| 4 | 9 | I like that the radar graph is similar to scores received in the brain training game. The radar graph should have consistent axis values for all metrics so it is easier to see which areas are doing better than others |
| 5 | 9 | Would be good to be able to view results from games completed over the course of a day rather than just an average of results each day |
| 6 | 10 | The charts were easy to understand as they were labelled well and clearly showed the results using appropriate graphs. |
| 7 | 9 | The visuals were well designed and the results were displayed in an organised and intelligible manner |

**Question 16:** How would you rate the explanation of result metrics? Was it clear what each metric was representing?

|  |  |  |
| --- | --- | --- |
| **Tester ID** | **Rating (1-10)** | **Comments** |
| 1 | 9 | Explanation is clear but a bit repetitive that they all end with ‘the last seven days’ |
| 2 | 8 | Not sure if the engagement is worked out correctly as someone who has played the game many times each day is treated the same as someone that has played it once |
| 3 | 10 | The explanation was clear and concise. I liked how it was at the beginning of the results section, and the user didn’t need to go looking for it |
| 4 | 10 | The results were easy to interpret |
| 5 | 10 | Results are clear in what they are representing |
| 6 | 9 | Results are clear but the engagement result explanation is inconsistent with the others |
| 7 | 10 | Explanation of results was clear and explained the weekly summary chart well |

**Question 17:** How would you rate the ability to send game results via email to a medical professional? Do you think this is a useful feature?

|  |  |  |
| --- | --- | --- |
| **Tester ID** | **Rating (1-10)** | **Comments** |
| 1 | 10 | Definitely a useful feature, it helps the medical professional to keep up to date with the user’s progress |
| 2 | 9 | Results can be edited in the email which you might not want users to be able to change |
| 3 | 9 | You could have a microservice that just sends an email to the medical professional so the user can’t change anything |
| 4 | 9 | You could append the results after the email has been edited by the user so they can’t change their scores |
| 5 | 10 | This is a very useful feature as it saves the user time and effort writing down their results to create their own email on a different app. I like how the email is set up for the user |
| 6 | 10 | The email feature allows for quick and efficient communication between the user and their medical professional |
| 7 | 9 | I think this is a really valuable feature, the mail icon should be able to be clicked as it is currently inconsistent with the other home screen icons |

## Appendix 10: Beta Testing

**Question 1:** How would you rate the ability to register an account with Swift?

|  |  |  |
| --- | --- | --- |
| **Tester ID** | **Rating (1-10)** | **Comments** |
| 1 | 9 | Should tell you the requirements for a password before trying to register rather than in the alert pop-up |
| 2 | 7 | Instead of ‘user already exists’ it should say ‘email already in use’, password specifications should be on the screen instead of in the alert |
| 3 | 8 | The alert should indicate whether it is the email or password that is invalid |
| 4 | 10 | Smooth process for registering an account |
| 5 | 8 | Confusing that it tells you what the password rules are in the alert after you’ve already entered the password |

**Question 2:** How would you rate the ability to login with Swift?

|  |  |  |
| --- | --- | --- |
| **Tester ID** | **Rating (1-10)** | **Comments** |
| 1 | 8 | Should only have to click login button once, confusing that you have to select it again |
| 2 | 7 | Alert could specify which details are invalid |
| 3 | 8 | Alert should specify if the username or password is invalid |
| 4 | 10 | The login process worked smoothly |
| 5 | 7 | The change password feature is useful but doesn’t work |

**Question 3:** How would you rate the navigation between screens?

|  |  |  |
| --- | --- | --- |
| **Tester ID** | **Rating (1-10)** | **Comments** |
| 1 | 10 | Easy to move between screens, no lagging |
| 2 | 10 | Easy navigation between screens |
| 3 | 9 | Navigation takes you to the expected screen |
| 4 | 8 | I like that you can go back to the previous screen on iOS by swiping, when you swipe back to the game from the game results screen the game doesn’t reset |
| 5 | 8 | I like the use of the ‘coming soon’ screen as opposed to no navigation to screens that have not yet been implemented, the app breaks when the strength test is selected, the delete account button in settings takes you to the register screen, I like the alert for logging out in case you accidentally click the button |

**Question 4:** How would you rate the layout of the user interface - is it easy to understand? Is the styling consistent across screens?

|  |  |  |
| --- | --- | --- |
| **Tester ID** | **Rating (1-10)** | **Comments** |
| 1 | 8 | Styling is consistent, like that colour changes when switching between buttons in the tab navigation, some whitespace at the bottom of the settings and help screens so could move the buttons down a bit |
| 2 | 9 | Text is easy to read, confused as to why the buttons with links are blue, should be able to select mail icon to access email feature rather than the text below |
| 3 | 9 | Could give more explanation as to what each button is on the home screen, styling is consistent, settings button in tab navigation should say ‘settings’ instead of ‘profile’, buttons with links are blue which is inconsistent with the text colour in the other buttons |
| 4 | 10 | I like that the buttons change colour slightly to confirm you have clicked it, the tab navigation colour change is good |
| 5 | 8 | I like the font used throughout the app and the link to the NHS website, the leave feedback button on the help screen isn’t very clear and just looks like a share button so it should maybe link to a google form instead, the colour change of the tab navigation is nice but would be better if the screens changed colour as well as it is difficult to notice. |

**Question 5:** How would you rate the text and button size across screens? (1 = very difficult to read, 10 = very easy to read)

|  |  |  |
| --- | --- | --- |
| **Tester ID** | **Rating (1-10)** | **Comments** |
| 1 | 10 | Text is large and easy to read on home page which would still be clear to those with visual impairments |
| 2 | 9 | Tab navigation buttons could be slightly larger |
| 3 | 8 | Text could be a bit larger in help and settings screens to fill page more and accommodate those with visual impairments |
| 4 | 10 | Text and buttons are well spaced out and a good size |
| 5 | 8 | The results screen and home screen text and buttons appear overlapped on the Galaxy A51 |

**Question 6:** How would you rate the dark mode styling? Are the colours contrasting enough?

|  |  |  |
| --- | --- | --- |
| **Tester ID** | **Rating (1-10)** | **Comments** |
| 1 | 7 | View results button on home screen doesn’t change in dark mode, weekly summary results screen doesn’t change in dark mode |
| 2 | 5 | I personally don’t see the benefit of dark mode, results screen text is still in black and is difficult to see against the dark background, text is overall difficult to read in dark mode |
| 3 | 7 | Logo doesn’t stand out against dark background, view results icon doesn’t change in dark mode |
| 4 | 8 | The dark mode is a nice feature but the logo and text on results screens blends in with the background |
| 5 | 8 | I like the dark mode, but the logo and view results icon don’t change colour |

**Question 7:** How would you rate the colour scheme and layout of the game?

|  |  |  |
| --- | --- | --- |
| **Tester ID** | **Rating (1-10)** | **Comments** |
| 1 | 9 | The circle sometimes generates on the edge of an obstacle, contrasting colours are good to indicate where you’re supposed to go |
| 2 | 10 | Red is good for making obstacles stand out, how to play button is clear |
| 3 | 9 | Not a fan of the blue target, could make the white cross bigger and bolder |
| 4 | 8 | The circle first generated on an obstacle and the game had to be reset to play it |
| 5 | 10 | I like that the alert is displayed to check you definitely want to end the game |

**Question 8***:* How would you rate the size of the circles/obstacles - were they clear enough?

|  |  |  |
| --- | --- | --- |
| **Tester ID** | **Rating (1-10)** | **Comments** |
| 1 | 9 | Draggable circle could be a bit bigger as it may be difficult for those who have suffered from a stroke and may have shaky hands |
| 2 | 9 | Green circle could be slightly bigger |
| 3 | 7 | Green circle should be slightly bigger as it is difficult to see behind your finger |
| 4 | 9 | Circles and obstacles are clearly visible and a suitable size |
| 5 | 10 | The size of the circles and obstacles is ideal as they are not too big or small |

**Question 9:** How clear was the tutorial in explaining the game instructions? (1 = not clear at all, 10 = very clear)

|  |  |  |
| --- | --- | --- |
| **Tester ID** | **Rating (1-10)** | **Comments** |
| 1 | 10 | Instructions are clear on what the user is supposed to do |
| 2 | 10 | Clear instructions that are easy to understand |
| 3 | 10 | Instructions are very clear |
| 4 | 10 | Tutorial is good and clearly explains the game instructions |
| 5 | 10 | Tutorial provided a thorough understanding of how to play the game |

**Question 10:** How quickly did the game load? (1 = very slowly, 10 = immediately)

|  |  |  |
| --- | --- | --- |
| **Tester ID** | **Rating (1-10)** | **Comments** |
| 1 | 10 | The game loaded quickly without any issues |
| 2 | 10 | Game loaded immediately with no problems |
| 3 | 10 | Game loaded instantly |
| 4 | 9 | The game loaded reasonably quickly, an alert appeared when I tried to start the game saying the difficulty hadn’t loaded but was able to start the game after this |
| 5 | 10 | The game loaded in a timely manner |

**Question 11:** How would you rate the difficulty of the game? (1 = too easy/difficult, 10 = appropriate difficulty)

|  |  |  |
| --- | --- | --- |
| **Tester ID** | **Rating (1-10)** | **Comments** |
| 1 | 8 | Could have an option on settings screen to change the difficulty level of the coordination game, difficulty levels are reasonable for those in the process of stroke rehabilitation |
| 2 | 8 | I think the game is the appropriate level of difficulty for the target users, would be good to have the option to choose the difficulty level |
| 3 | 9 | Appropriate level of difficulty for target users |
| 4 | 9 | The game was not too difficult but provided a good level of challenge |
| 5 | 10 | I thought the game was a good level of difficulty and wasn’t too difficult that it became frustrating |

**Question 12:** Did you enjoy playing the game? Did you find it engaging? (1 = didn’t enjoy it at all, 10 = really enjoyed it)

|  |  |  |
| --- | --- | --- |
| **Tester ID** | **Rating (1-10)** | **Comments** |
| 1 | 9 | Easier level wasn’t very challenging but difficult level was more engaging |
| 2 | 10 | The game is engaging and motivates you to keep playing to make progress |
| 3 | 8 | Game is enjoyable for what it is trying to achieve in assessing coordination |
| 4 | 9 | I enjoyed the game and liked that it changed layout each time to keep it engaging |
| 5 | 10 | I thought the game was fun and was an interesting way to assess coordination |

**Question 13:** How quickly were results displayed on completion of the game? (1 = very slowly, 10 = immediately)

|  |  |  |
| --- | --- | --- |
| **Tester ID** | **Rating (1-10)** | **Comments** |
| 1 | 10 | Results were displayed straight away with no delays |
| 2 | 10 | Results were displayed immediately |
| 3 | 10 | Instantaneous results |
| 4 | 10 | Results appeared promptly on completion of the game |
| 5 | 10 | Results were displayed quickly and I didn’t have to wait long |

**Question 14:** How would you rate the scoring of the game – were you able to understand the results breakdown?

|  |  |  |
| --- | --- | --- |
| **Tester ID** | **Rating (1-10)** | **Comments** |
| 1 | 8 | Percentage change is 0 when playing game for the first time, should have something in place of that to say there are no previous results to compare to, like the fact you can compare results to the last time you played |
| 2 | 9 | Could reword the percentage change explanation as it is slightly confusing for the time taken metric |
| 3 | 10 | Results breakdown is easy to understand, shows main information you would want to see to highlight improvements or regression |
| 4 | 10 | I think the scoring was straightforward and clearly explained |
| 5 | 10 | I like the pop-up before results are displayed, results were easy to understand and used a variety of performance metrics |

**Question 15:** How would you rate the charts displaying the average game results – were they easy to understand? Were they clear to read?

|  |  |  |
| --- | --- | --- |
| **Tester ID** | **Rating (1-10)** | **Comments** |
| 1 | 8 | In the weekly summary chart, it should be clear what the maximum value is that can be achieved. Would be nice to be able to see results from further back in time |
| 2 | 8 | Would be good to be able to see the average difficulty each day in the line chart, put in brackets after axis labels the appropriate unit e.g., seconds for time, days for engagement, percentage for accuracy |
| 3 | 9 | Should point out what the ideal results should be in the weekly summary chart |
| 4 | 8 | The line graph animation was good, the radar graph should display the maximum values at the edge rather than the user results |
| 5 | 9 | The charts are well-designed and provided a comprehensive view of results |

**Question 16:** How would you rate the explanation of result metrics? Was it clear what each metric was representing?

|  |  |  |
| --- | --- | --- |
| **Tester ID** | **Rating (1-10)** | **Comments** |
| 1 | 9 | Instead of repeating ‘in the last seven days’ you should specify somewhere that all results displayed are the average of the last seven days. Aside from that the metrics are explained well |
| 2 | 9 | Inconsistency in use of ‘last seven days’ and ‘last week’, could state at the top that results are from the last seven days, I was able to understand the results and interpret them |
| 3 | 8 | Engagement metric is unclear as to whether that relates to consecutive days or not, ‘in the last week’ is inconsistent with ‘in the last seven days’, could just say ‘in the last seven days’ at the top to be more concise |
| 4 | 9 | Results metrics were clearly explained and their significance was well-defined |
| 5 | 8 | The explanation of results in the radar chart isn’t very clear |

**Question 17:** How would you rate the ability to send game results via email to a medical professional? Do you think this is a useful feature?

|  |  |  |
| --- | --- | --- |
| **Tester ID** | **Rating (1-10)** | **Comments** |
| 1 | 8 | You should be able to select the icon in the home screen rather than the text below it to be consistent with the other home screen buttons, could specify the average results are from the last week |
| 2 | 9 | Should be able to select the mail icon to access the email feature, the email is clear and useful to have results pre-populated |
| 3 | 9 | Results don’t specify the period of time the results are from |
| 4 | 10 | I think the email feature is a valuable addition to the game and saves the user time and effort |
| 5 | 8 | I like the email feature so users have the ability to send their results, however I was logged in under Sophie’s account and the email was set to send from my own Gmail account |

Primary Responsibilities

|  |  |
| --- | --- |
| Student Name | Primary Responsibilities |
| Sarah McMahon (40257428) | *Report:* Sections – 1.3, 2.2, 2.3, 3.2, 4.1, 4.1.1, 5.3.1, 5.5, 7.2, 7.3, 7.4, 8.1, 8.2.1  *System:* Front-end design & navigation, splash screen, send email feature, tutorial, dark mode feature |
| Michal Guzy (40266405) | *Report: Sections – 3.1, 3.4.1, 3.4.2, 3.4.4, 3.4.5, 4.2.1, 4.2.2, 4.2.3, 4.2.4, 4.3.1, 4.3.2, 4.3.3, 4.3.4, 5.2, 5.3.3, 5.4, 8.2.2, 8.3, 8.5.1, 8.5.2, 8.5.3, 8.5.4*  *System: Obstacle generation in game, setting up services with Docker, C# User Management Service, Communication between front end and back end, Communication between services, Setting up and storing in database, storing app context on front end, C# testing* |
| Sophie Young (40257097) | *Report:* Sections – 1.1, 1.2, 2.1, 2.4, 3.3, 4.1.2, 4.1.3, 5.1, 6, 8.4  *System:* Accuracy calculation for game, daily progress line chart, weekly progress radar chart, view results screens |
| Thomas Reid (40263793) | *Report: Sections – 3.4.3, 4.2.5, 5.2, 5.3.2, 7.1, 8.5.1*  *System: Game Design, python results analysis scripts, python single game results scripts, python difficulty scripts* |

Agreed Contribution (%)

|  |  |
| --- | --- |
| Student Name | Agreed % Contribution |
| Sarah McMahon | *25% Report*  *25% System* |
| Michal Guzy | 25% Report  25% System |
| Sophie Young | 25% Report  25% System |
| Thomas Reid | 25% Report  25% System |