



**UNIVERSITY OF
PORTSMOUTH**

MyCaffeine: Caffeine Tracker for iOS

Ryan Bush

UP904935

School of Computing
Final-year Project
Engineering

Abstract

Caffeine has a long history of human consumption and comes with both positive and negative health benefits. This project aims to produce an application to help users track and monitor their caffeine intake to understand their consumption better.

This project documents the processes used in developing the MyCaffeine application for iOS devices. Agile software development cycles are used to effectively design, implement, and test the application. All stages of the implementation are recorded, and the final application is analysed against initial plans. Research into caffeine consumption and application development is conducted to enhance understanding further.

Table of Contents

<i>Abstract</i>	2
TABLE OF FIGURES	5
TABLE OF TABLES	7
1. INTRODUCTION	8
1.1 PROJECT OVERVIEW	8
1.2 PROJECT AIMS & OBJECTIVES.....	8
1.3 PROJECT CONSTRAINTS	8
1.4 SOCIAL, LEGAL AND ETHICAL ISSUES.....	9
1.5 SUMMARY	9
2. LITERATURE REVIEW	10
2.1 INTRODUCTION.....	10
2.1.1 <i>Methods</i>	10
2.2 APPLICATION DEVELOPMENT.....	10
2.2.1 <i>Platform</i>	10
2.2.2 <i>Native vs Cross-Platform</i>	11
2.3 CAFFEINE	11
2.3.1 <i>Caffeine Consumption</i>	12
2.4 CURRENTLY AVAILABLE MOBILE APPS	12
2.4.1 <i>WaterMinder</i>	12
2.4.2 <i>HiCoffee</i>	13
2.4.3 <i>Barista</i>	14
2.4.4 <i>Application Comparison</i>	15
2.5 SUMMARY	15
3. METHODOLOGY & PROJECT MANAGEMENT	16
3.1 INTRODUCTION.....	16
3.2 METHODOLOGY.....	16
3.2.1 <i>Waterfall</i>	16
3.2.2 <i>Incremental</i>	17
3.2.3 <i>Agile</i>	17
3.3 CHOSEN METHODOLOGY.....	18
3.4 PROJECT MANAGEMENT	18
3.5 CONCLUSION.....	18
4. REQUIREMENTS	19
4.1 INTRODUCTION.....	19
4.2 REQUIREMENTS FROM RESEARCH.....	19
4.3 REQUIREMENTS FROM QUESTIONNAIRE	19
4.3.1 <i>Analysis of Questionnaire</i>	20
4.3.2 <i>Summary</i>	22
4.4 REQUIREMENTS	22
4.5 SUMMARY	23
5. DESIGN	24
5.1 PROJECT INITIATION	24
5.1.1 <i>Version Control</i>	24
5.1.2 <i>Database</i>	24
5.1.3 <i>Security & Privacy Considerations</i>	24
5.2 TESTING	25
5.2.1 <i>Design Feedback</i>	25
5.2.2 <i>Application Testing</i>	25

5.3 SEQUENCE DIAGRAMS	26
5.4 USER INTERFACE DESIGNS	27
5.4.1 House Styles	27
5.4.2 UI Wireframes	28
5.4.4 High-Fidelity Designs	31
5.4.5 Testing on High-Fidelity Designs	36
5.5 SUMMARY	36
6. IMPLEMENTATION	37
6.1 PROGRAMMING LANGUAGE	37
6.2 PROGRAMMING RESOURCES	37
6.3 INTEGRATED DEVELOPMENT ENVIRONMENT CHOICE	38
6.4 VERSION CONTROL	38
6.5 COMMON COMPONENTS	38
6.5.1 Global Styling	38
6.5.2 Application-wide Icons	39
6.5.3 Helper Functions	39
6.6 CORE DATA	40
6.7 USER INTERFACE	40
6.7.1 Launch Screen	41
6.7.2 Navigation	42
6.7.3 Home View	43
6.7.4 Add Drink View	45
6.7.5 Trends View	48
6.7.6 History View	49
6.7.7 User Notifications	50
6.7.8 Apple Health Integration	51
6.8 EXTERNAL LIBRARIES	52
6.8.1 SF Symbols	52
6.8.2 FlatIcons	52
6.8.3 SwiftUICharts	52
6.8.4 HalfASheet	52
6.8.5 TextFieldStepper	53
6.9 SUMMARY	53
7. TESTING	54
7.1 PRIMARY TESTING METHODS	54
7.2 FUNCTIONALITY TESTING	54
7.3 RESPONSIVENESS TESTING	56
7.4 SUMMARY	56
8. EVALUATION	57
8.1 EVALUATION AGAINST REQUIREMENTS	57
8.1.1 Failed Requirements	57
8.2 METHODOLOGY EVALUATION	58
8.3 TIME MANAGEMENT EVALUATION	58
8.4 FUTURE WORK	58
8.5 PERSONAL CONCLUSION	59
REFERENCES	60
APPENDIX A: PID	64
APPENDIX B: ETHICS REVIEW	71
APPENDIX C: GANTT CHART	73
APPENDIX D: QUESTIONNAIRE	74
APPENDIX E: QUESTIONNAIRE RESULTS	79

Table of Figures

Figure 1: Search results for mobile development	10
Figure 2: WaterMinder Screenshot	13
Figure 3: HiCoffee Screenshot	14
Figure 4: Barista Screenshot.....	15
Figure 5: Waterfall methodology	16
Figure 6: Question Results: Do you consume caffeinated drinks?	20
Figure 7: Questionnaire Results: What type of caffeinated beverages do you usually consume?	20
Figure 8: Questionnaire Results: Roughly how many caffeinated drinks do you consume a day?	21
Figure 9: Questionnaire Results: Are you aware of how much caffeine is in the drinks you consume?	21
Figure 10: Questionnaire Results: Do you currently track your caffeine usage?	21
Figure 11: Questionnaire Results: Would you find a system that makes it easy to track your caffeine usage useful?	21
Figure 12: Sequence Diagram: User adding a drink to the system.....	26
Figure 13: Application colour scheme	27
Figure 14: Insights Graphs Colour Scheme	27
Figure 15: Designs for Application Icon	27
Figure 16: Designs for Application Logo	28
Figure 17: Initial hand UI Wireframes	28
Figure 18: Annotated UI Wireframe for Home Screen	29
Figure 19: Annotated UI Wireframe for Trends Screen.....	30
Figure 20: Activity section in Apple's Fitness app.....	31
Figure 21: Caffeine Intake Highlight high-fidelity designs	31
Figure 22: Home Screen High Fidelity Design.....	32
Figure 23: High Fidelity designs for Trends screens	33
Figure 24: High Fidelity design for History screen	34
Figure 25: High Fidelity design for Adding Drinks.....	35
Figure 26: Changes made to the trends charts.....	36
Figure 27: Changes made to the list of drinks in the History screen	36
Figure 28: The three courses completed on Codecademy	37
Figure 29: The two courses completed on LinkedIn Learning	37
Figure 30: The Asset Catalogue showing colours, including the difference in light and dark modes.....	38
Figure 31: An example of how images are stored in the assets catalogue, including multiple sizes	39
Figure 32: Helper function to calculate caffeine metabolism	39
Figure 33: Helper function to retrieve the current caffeine percentage.....	40
Figure 34: Helper function to retrieve the user's bedtime caffeine level.....	40
Figure 35: Major UI views of the application	41
Figure 36: The implemented launch screen	41
Figure 37: Implementation of the TabView within MainView	42
Figure 38: TabView Implemented as shown in light and dark mode	42

Figure 39: Implementation of the percentage circle on the highlights panel	43
Figure 40: The result of the implementation of the highlights panel is light and dark modes	43
Figure 41: Implementation of Today's Drinks in the HomeView	44
Figure 42: Result of the implementation of Today's Drinks	44
Figure 43: Implementation of the 'Add Drink' button	45
Figure 44: Implementation of manual entry of caffeine	46
Figure 45: Implementation of the list showing users created drinks	47
Figure 46: Result of the implementation of the list view with the HalfSheet and TextStepper	47
Figure 47: Implementation to build a chart for caffeine consumption over the past week ..	48
Figure 48: Implementation of the list shown on the History View.....	49
Figure 49: Implementation required to delete the item from the list	49
Figure 50: Implementation of Notifications	50
Figure 51: Implementation of HealthKit functions.....	51
Figure 52: Implementation of HealthKit function to save caffeine data	52
Figure 53: A few of the icons used from Flaticons	52
Figure 54: Application running on three different simulated devices.....	56

Table of Tables

Table 1: Methods of tracking for currently available applications	12
Table 2: Non-Functional Requirements.....	22
Table 3: Functional Requirements.....	23
Table 4: Results of Functionality Testing	55
Table 5: Evaluation against Requirements	57

1. Introduction

Caffeine, found in various plant constituents such as coffee and cocoa beans and tea leaves, has a long history of human consumption (European Food Safety Authority, 2017). Most popularly consumed in coffee and tea, caffeine has numerous health outcomes, both positive and negative.

MyCaffeine will be a mobile application that aims to help users track and analyse their caffeine consumption by primarily focusing on consumption through drinks.

1.1 Project Overview

The issue that this application aims to resolve is the high caffeine consumption in young adults. The application will allow users to enter drinks they have consumed and analyse their caffeine intake. The application will initially be available on iOS devices, supporting mobile systems.

1.2 Project Aims & Objectives

This project will critically analyse and research similar mobile applications available to design and develop a caffeine tracking application. The application will enable users to add, track and monitor their caffeine intake. This application is being made to help users find out more about how much caffeine they consume and its effect on their bodies. Furthermore, it is often unclear how much caffeine is in foods and drinks, with companies not required to display caffeine content on their products. This application will show how much caffeine is in branded products and allow users to create their consumption habits.

Specific objectives will need to be established to achieve the aims detailed above. Firstly, a literature review will be conducted researching the finer details of caffeine and its usage in the modern day. The review will also describe various methods of implementing applications on an iOS platform, looking at native versus cross-platform methods. The system requirements will be crafted from the literature review and a short questionnaire. Various project methodologies will be analysed and one selected for this project, then designs will be produced for the applications GUI and database structures. The implementation of the artefact will be documented, ensuring it meets the requirements set out and tested to high standards. A conclusion will be drawn, and any future work will be discussed.

During this project, there are also several personal development objectives to improve knowledge of mobile application development.

1.3 Project Constraints

Several constraints will have to be considered when designing and developing the application that may drastically affect the end artefact. Depending on their nature, these constraints can somewhat be prevented or lessened.

The constraint that has the most damaging effect will be time. Roughly eight months are allocated to complete this project, and with such a short timeframe, there is a great risk of delay. Without a fixed deadline, there is no doubt that the final system would be substantially

more influential. However, with a deadline, designs and requirements must be more practical to complete the development in time. A GANTT chart (Appendix C) will be created to enforce better time management with chosen deadlines for each section.

A further constraint is the level of coding experience. A particularly low level of expertise when developing Swift applications will likely increase the time it takes to implement more complex aspects of the system. Knowledge of different programming languages is likely to help, as certain programming features are anticipated to carry over to the Swift language. A lot of time will be reserved for the implementation stage to combat this constraint.

Another constraint, which is likely to become less likely as the project evolves, is the COVID-19 pandemic. The risk of lockdowns or reduced social contact can change how certain aspects of the project are approached. One change because of this is how requirements are gathered. There will be an emphasis on requirements collected from reviewing existing systems and a questionnaire rather than holding face-to-face focus groups.

1.4 Social, Legal and Ethical Issues

This project will need to follow several relevant laws, notably the General Data Protection Regulation (GDPR). To help with data protection laws, personal data stored will be minimal.

All ethical policies within the university will be followed, and an ethics review will be conducted.

1.5 Summary

This chapter gives a brief overview of the project, including aims and objectives and why the project is needed. The next chapter will further analyse caffeine and existing systems within a literature review.

2. Literature Review

2.1 Introduction

While a basic understanding is currently known, this chapter will outline methods to build this knowledge. The following literature review will examine different elements of mobile development through books, journals, and other publications. It will also investigate caffeine consumption within the general population to better understand habits and draft requirements.

2.1.1 Methods

Various research tools were used in the creation of this literature review. To help find relevant documents, keywords were entered into Google Scholar and the university's online library tool. An example of how a search term was narrowed down is shown in Figure 1.

Terms	Engine	Results	What is this search?	Result
mobile application	ebSCO	5,338,534	Initial search.	Too many matches, narrowing needed. Majority of matches seem somewhat relevant
mobile application development	ebSCO	2,361,292	Added development focus	Fewer results, still too vague
"mobile application" development	ebSCO	112,030	Changed mobile application to phrase	Results include patents and wide year range.
"mobile application" development AND date 2014-2021	ebSCO	91,787	Focused date to more recent years (2014+)	Still a lot of patents and various languages.
("mobile application" AND development AND "Swift") NOT (patent) AND date 2014-2021	ebSCO	3,082	Removed patents from search results and added focus on Swift	More narrow results appearing, a lot of e-books
("mobile application" AND development AND "Swift") NOT (patent) AND date 2014-2022 AND "Academic Journals"	ebSCO	762	Focused on just academic journals, rather than e-books	Maybe time to use the 'refine results' panel.
("mobile application" AND development AND "Swift" AND "design") NOT (patent) AND date 2014-2022 AND "Academic Journals"	ebSCO	630	Added focus for design	Small reduction
("mobile application" AND development AND "Swift" AND "design" AND "iOS") NOT (patent) AND date 2014-2022 AND "Academic Journals"	ebSCO	202	Added focus for iOS	More manageable search result, first page looks very relevant

Figure 1: Search results for mobile development

2.2 Application Development

This section of the literature review will investigate various key application development concepts. This will ensure that the final product is completed to a high standard and follows leading industry standards.

2.2.1 Platform

The success of a software product for mobile devices will be conditioned by the popularity it receives (Delia et al., 2017). Smartphone usage has increased by 238% from 2010 to 2020 (Statista, 2021), now estimated 61.41 million users. With smartphones becoming increasingly popular, it is critical to ensure that applications are deployed to the right platform to maximise their potential target audience. The leading platforms for mobile development are Android and iOS, with just a 0.53% difference in market share within the United Kingdom (StatCounter, 2021a). Although maintaining an even market share in the UK, Android is seen as fast-growing and taking over from now obsolete systems, such as Blackberry OS and Symbian OS, and enjoys a worldwide market share of near 73% (StatCounter, 2021b).

Android has a bigger problem when it comes to project timelines. Developing Android applications generally take more time due to device fragmentation (Lamhaddab et al., 2019). Fragmentation occurs when there are many devices supporting Android, and it becomes challenging to create an application that works on them all. Therefore, the clear advantage worldwide market share is significantly reduced due to the developed application only reaching a fraction of that share. In contrast to Android, iOS does not suffer from the same issues as fragmentation (Grønli et al., 2014). This mostly comes down to Apple being the only creator of iOS devices, whereas Android has many manufacturers.

Whereas an application that solely targets iOS would reduce the potential target audience, the faster development time due to less device fragmentation would be ideal for combating the rapid development phase in most projects. The release of SwiftUI (Apple, n.d.-m) in 2019 provided a declarative framework for UI structure across all Apple platforms, compared to the traditional library approach beforehand. This has allowed for more rapid UI development, which is currently unmatched in the Android equivalent, Google Firebase.

2.2.2 Native vs Cross-Platform

Nowadays, most mobile applications are developed in three ways; native, web, and hybrid. Native applications have been the traditional way of developing applications and involve the use of languages and tools specifically designed for each platform. This allows the application to fully use the device's capabilities and often allows for better performance and minimal lag with CPU render time. A considerable consequence of this approach is that if an app should reach a multi-platform audience, the entirety of the app must be written twice, once for Android and once for iOS (Biørn-Hansen et al., 2020). It is pretty rare for modern applications to be built for a single platform, but native development is usually the best way to go if this is the case.

Due to the often-higher costs and knowledge levels related to developing native applications, numerous alternatives are often known as cross-platform mobile development. Typically, a single codebase can be used to develop applications across several platforms, often with little to no platform-specific modifications (Biørn-Hansen et al., 2020). These tools use popular programming languages, which allows for a smaller learning curve. React Native is often seen as the leader in cross-platform technology, having been released by Facebook in 2015. The apps are built using JavaScript but are indistinguishable from natively developed apps in Android or iOS (Shah et al., 2019). React Native is currently used in thousands of mobile applications, including large companies such as Uber, Shopify, and Facebook (*React Native · Learn Once, Write Anywhere*, n.d.).

2.3 Caffeine

Caffeine is one of the most widely used psychoactive substances in the world (Daly et al., 1998). Caffeine can be consumed in various ways, with the most common coming from beverages such as tea, coffee, and carbonated soft drinks. Higher doses of caffeine induce adverse effects such as anxiety, restlessness, insomnia, and tachycardia (Nehlig, 1999). While three of these effects can be relatively mild, tachycardia can be potentially life-threatening.

2.3.1 Caffeine Consumption

In the United Kingdom, approximately 95% of the population reports consuming caffeinated beverages at least once per week (Stine et al., n.d.), most commonly tea and coffee. In addition to caffeine in hot drinks, caffeine is artificially added to most carbonated soft drinks, such as Red Bull and Coca Cola. While high amounts of soft drinks rarely cause problems to most adults, teenagers aged 11 – 18 have consumed nearly 450 grams of soft drinks a day (Public Health England, 2014).

For adults, moderate caffeine intakes of approximately 300mg per day are well tolerated (Fitt et al., 2013). Although there is no dietary recommended amount in the UK, the consensus is that adults observe this level of consumption, with younger ages advised to consider a lower amount. In 2015, The European Food Safety Authority published a report stating that the consumption for adults should not exceed 400mg per day (EFSA Panel on Dietetic Products, 2015).

2.4 Currently Available Mobile Apps

This next section details research into mobile applications currently available on the iOS App Store relating to tracking caffeine consumption. Doing this research was critical to ensure user requirements could be drafted. The applications are detailed below (Table 1).

<i>App Name</i>	<i>Method of Tracking</i>	<i>Price</i>
<i>WaterMinder</i>	Manual	£4.49 one off
<i>HiCoffee</i>	Fixed drinks & manual	Free, Premium £5.99 or £0.99 per month
<i>Barista</i>	Fixed drinks & manual	Free, Premium £0.89 one off

Table 1: Methods of tracking for currently available applications

2.4.1 WaterMinder

WaterMinder (WaterMinder, n.d.) is a mobile application available on iOS, macOS and Android that primarily track a user's water intake; however, it recently added the ability to track caffeine. Tracking caffeine simply involves manually adding a quantity of caffeine directly into the application. Caffeine tracking isn't enabled by default, so it needs to be enabled from the settings. Other than viewing your intake history, there is no further mention of caffeine in the application.

The application supports Apple Health synchronisation but doesn't pull data from Apple Health into the application, so any caffeine added elsewhere isn't counted. There are also Apple Watch complications; however, none mentions caffeine.

Key Features:

- A basic way of entering caffeine consumption
- View history of caffeine consumption
- Synchronisation to Apple Health
- Light/Dark modes

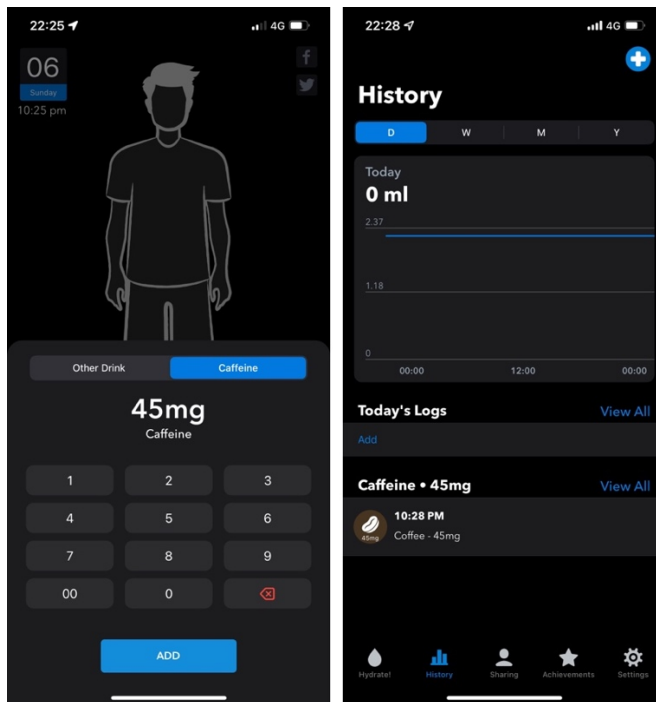


Figure 2: WaterMinder Screenshot

WaterMinder is a paid application available for a price of £4.49. A macOS version is available for £2.49, and an Android version is free to install but has in-app purchases.

Overall, WaterMinder offers poor support for tracking caffeine intake. This was expected as the application is primarily used for monitoring water consumption.

2.4.2 HiCoffee

HiCoffee (HiCoffee, n.d.) is a caffeine tracking application built for iOS devices. The user is presented with a clean user interface to see their current caffeine level, total drinks today, total daily caffeine level, and caffeine metabolism over the next 12 hours. The log screen shows a calendar view, which views the number of drinks consumed that day by changing the calendar colour. A simple list shows all the drinks and allows them to be edited, while the insights screen gives you different reports weekly and monthly. HiCoffee is a more complete application than WaterMinder but considering that the latter is designed mainly for tracking water intake, this is expected. User data can be synchronised to the user's iCloud account, allowing data to be backed up and viewed on several devices.

Key Features:

- Clear, user-friendly interface
- Add caffeine usage through drinks
- View history of caffeine usage through list and calendar
- View various insights into caffeine usage, including comparing week-on-week and month-on-month
- Custom metabolism preferences
- Apple Health synchronisation
- Light/Dark mode
- Notification reminders at custom times during the day

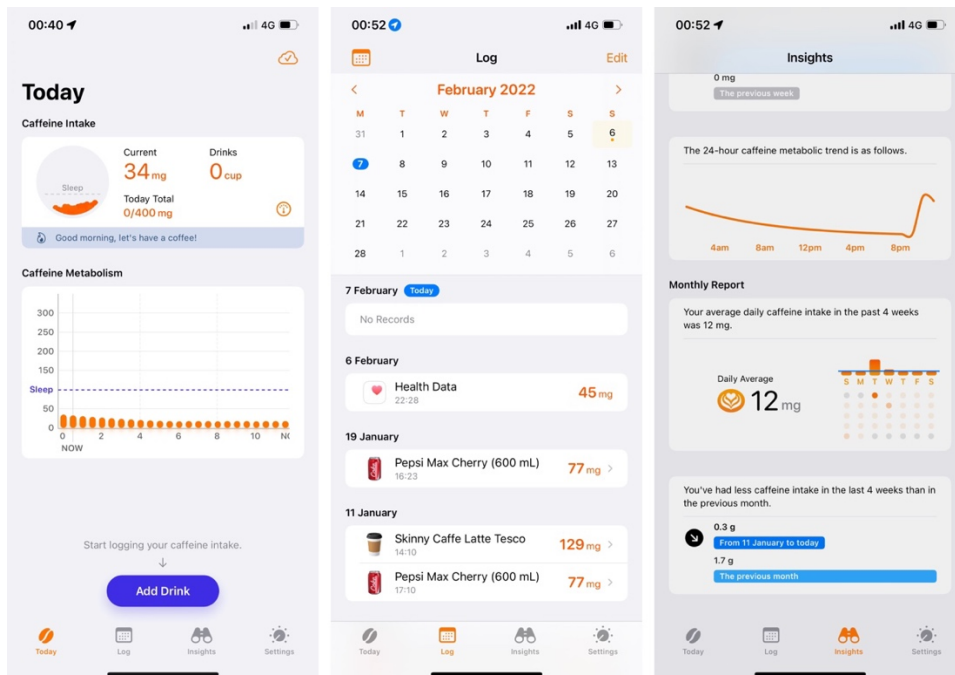


Figure 3: HiCoffee Screenshot

HiCoffee is a free application available to download on iOS and iPadOS devices only. A premium subscription is available, priced at either £0.99 per month or £5.99 for a lifetime subscription. The premium service allows users to access beverage data for popular branded drinks and allows the user to add custom drinks.

Overall, HiCoffee is a complete application for tracking caffeine consumption. It offers a clean user interface and displays just the right amount of data to read briefly. The ability to add drinks of popular brands allows users to ensure they add the correct caffeine levels.

2.4.3 Barista

Barista(Barista, n.d.) is an application for iOS devices that allows caffeine consumption to be tracked. It features a large user interface with quick add buttons to quickly add popular drinks. A graph shows your caffeine metabolism; however, it only starts when you add your first drink, not carrying over from the day before. The trend page features a calendar, which changes from blue to red on days caffeine has been consumed. On clicking on the date, you are shown the caffeine metabolism for that day, but this again starts from the day's first drink. The application also allows you to track specific symptoms that the user may experience.

Key Features:

- Large user interface design
- Add caffeine from quick drinks or branded drinks
- Track symptoms
- Notification reminder at one point during the day

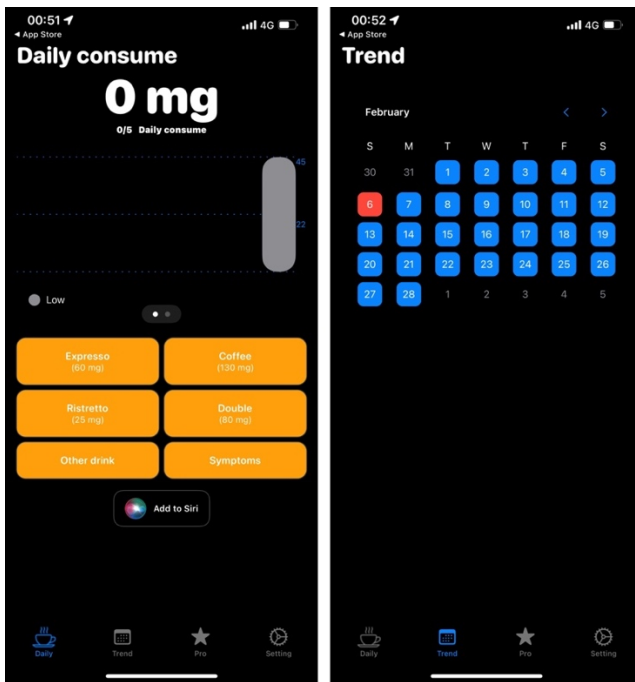


Figure 4: Barista Screenshot

Barista is available on the App Store for free, with a premium version available for £0.89. The premium version adds advanced monitoring and synchronisation of data to iCloud.

Overall, Barista offers an enhanced way of tracking caffeine compared to WaterMinder but falls short of HiCoffee. The large user interface can be user-friendly but feels too big on larger phones. How caffeine monitoring is hidden behind a paywall would likely discourage users from this application.

2.4.4 Application Comparison

The three applications that were compared had contrasting ways of tracking caffeine consumption. WaterMinder, by far, offers the most unfavourable way of tracking, although this was somewhat expected as the primary goal of the application is different to the others. Out of the three, HiCoffee was the best application considering both design and functionality; however, it lacked a simplistic way of adding caffeine by number. Showing insights into data was also superior to other systems and allowed the user to get a full grasp of their usage. Barista balanced adding caffeine consumption, offering both a manual method and branded drink options, but the user interface wasn't as clean as the others.

2.5 Summary

After analysing the literature above, many key features were found. For the overall design, the application should be clean and user-friendly. There should not be much information on the screen that the user must search through to find what they need. Allowing a user to track caffeine in both a manual method and automatically is a must for this system. Looking at the existing applications from an outside point of view allowed flaws to be found, which will be attempted to avoid when implementing MyCaffeine. The literature analysis and current applications will be used to craft some system requirements in chapter 4.

3. Methodology & Project Management

3.1 Introduction

This chapter will investigate varying software development methodologies, discussing the advantages and disadvantages of using each within a project. Three methodologies will be reviewed, and one will be chosen for usage in this project.

3.2 Methodology

Each project must go through a development cycle, often called a software development life cycle. It provides the basic framework for managing the project. Choosing the wrong methodology can have adverse effects. Studies have investigated reasonings for failed projects and found most projects fail due to (Lindstrom & Jeffries, 2004):

- Requirements that are not clearly communicated
- Requirements that do not follow the business problem
- Requirements that change prior to the completion of the project
- Software that has not been tested
- Software that has not been tested as the user will use it
- Software developed such that it is difficult to modify
- Software that is used for function for which it was not intended
- Projects not staffed with the resources required in the project plan
- Schedule and scope commitments are made prior to fully understanding the requirements or the technical risks

3.2.1 Waterfall

One of the first models to be introduced was the waterfall methodology which works sequentially by requiring each task to be fully complete before the next starts, as shown in Figure 5. It is designed to be simple and easy to put into use. Waterfall typically requires a long time of requirements gathering and project planning before any code is written (Shaydulin & Sybrandt, 2017).

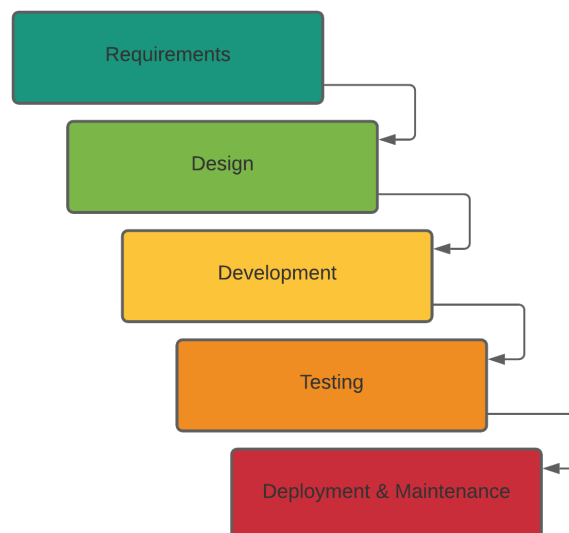


Figure 5: Waterfall methodology

The waterfall methodology has its advantages over others, such as maintaining a consistent approach over the different engineering models and allowing each stage to be monitored for progress. Due to the requirement for each step to be completed before the next can start, it can be handy as the design, development and testing phases rely on the preceding stage being completed before they can begin.

The waterfall methodology will not be used to produce this application, as it can reduce the focus on the implementation stage by requiring longer to gather requirements. This could lead to requirements not being added to the project.

3.2.2 Incremental

The incremental, also known as iterative, is a repetitive methodology that emphasises smaller steps toward the overall goal. Compared to the waterfall model, a working system is available earlier in the project, which is then improved and expanded over time. This model allows the project owner's feedback to be retrieved after each iteration has been completed, allowing the addition of requirements until the project owner is happy with the final product.

One of the significant advantages of this model is that a working version of the application is available early on, allowing for fine-tuning. This methodology is effective when working on larger projects, where multiple members can work on separate increments. Furthermore, if there is a change to requirements late into the development stage, this model is better adapted.

It can be hard to break tasks down into smaller chunks when working on a smaller project, so this model isn't well-adapted for smaller applications. There is also no clear endpoint to the incremental model, as more iterations can be added at any stage. For this reason, it will not be used for this project.

3.2.3 Agile

Agile development is a wide range of methodologies, primarily focusing on flexibility. It is a conceptual framework for software engineering that begins with a starting planning phase and follows the road toward the deployment phase with iterative and incremental interactions throughout the life cycle of the project (Alsaqqa et al., 2020). The iterative approach taken by agile methodologies means processes are improved upon each time an interval is repeated.

Extreme Programming (XP) is a form of agile development, primarily focusing on software development with a team of programmers. It focuses on regular minor releases concentrate on new features. Requirements are divided into several iterations, and each iteration ends with customer acceptance testing (Hamed & Abushama, 2013).

Since XP focuses on situations involving small teams, most extreme programming practices require teamwork. When a single person is working on their own, the personal extreme programming (PXP) methodology can be used, as outlined by Agarwal & Umphress, 2008. This follows the principles and values of XP, such as simplicity and feedback, but refines them to fit a lone programmer. This methodology includes principles such as (Dzhurov et al., 2009):

- Measure, track and analyse daily work
- Include use of continuous testing & integration
- Small releases
- Test-Driven Development

3.3 Chosen Methodology

PXP will be the methodology used throughout this project. Doing so will allow greater flexibility when gathering requirements and ensure that a working version is always available. This benefit can help mitigate the effects of extenuation circumstances and external factors affecting the development time for this project. While the waterfall methodology usually works well for pre-planned projects, the lack of flexibility due to a sole developer and time frame wouldn't allow for sufficient time to fix any issues. Incremental and Agile methodologies both accommodate changes in requirements, fitting this project better. Since incremental works best with a larger workforce, an agile methodology is best suited. PXP was chosen as it is best suited to working within a limited timeframe and implementing the project with a single person team.

3.4 Project Management

The GANTT chart (Appendix C) was the primary tool used for time management during this project. This allowed for rough ideas on how long each section would take to complete, allowing additional time if needed. Due to the anticipated more extended implementation stage, this allowed for greater planning.

More details relating to the project management will be detailed in the design phase.

3.5 Conclusion

This chapter helped to choose a software methodology that suited the uniqueness of this project. The chosen model, PXP, will be essential throughout the implementation of the application. Having a methodology is extremely useful to ensure a plan is in place to reach the final goal.

4. Requirements

4.1 Introduction

This section details how the requirements for the system will be produced, as well as the processes used to gather them. The methods used are discussed, the results will be analysed, and requirements will be formed. This will ensure that the users' needs are met, and competitiveness over alternative systems is maintained.

Furthermore, to get a clear understanding of these requirements and to help with evaluating them against the final project, the prioritisation will be detailed following the MoSCoW technique. The four categories of prioritisation used are as follows (Hatton, 2008):

- **Must:** Requirements are not negotiable; the failure to deliver these requirements would result in the failure of the entire project.
- **Should:** Features that would be nice to have if at all possible.
- **Could:** Features that would be nice to have if at all possible but slightly less advantageous than the "Should".
- **Won't:** These requirements are not unimportant, but they will defiantly not be implemented in the current software project. They may, at a later stage, be created.

Ideally, a range of methods would be used to gather requirements; however, this was not entirely possible due to the Covid-19 pandemic and various restrictions. As a result, the primary method for collecting requirements for this project will be analysing existing systems and a small questionnaire. These techniques haven't been affected by the pandemic and can be completed with the same efficiency as before.

4.2 Requirements from Research

The research conducted previously gauged an initial understanding of basic requirements. Further analysis of existing systems helped establish requirements by comparing features between applications and bridging any gaps found. Analysis of caffeine addiction's effects helped with critical components that can be implemented into this application.

4.3 Requirements from Questionnaire

To help understand the use of a caffeine tracking application in the general population, a questionnaire was distributed to fellow students and friends. Google Forms was used to create this (see Appendix D), using a range of caffeine usage and tracking questions. The questionnaire solely featured closed questions, with most questions being multiple choice. Although open questions allow for more probing answers, they weren't used to ensure the recipient completed the entire questionnaire. The questions were also kept simple and easy to understand to ensure the recipient understood the question and inputted a correct answer.

A total of 17 responses were collected and analysed. The questionnaire was popular, and all participants answered all questions.

4.3.1 Analysis of Questionnaire

The first page asked for some basic personal information, which could not be tied to each participant. This was included to see any correlation between the drinks consumed and the personal data provided. An example of data found was those participants in the older age brackets, 46 or above, mainly consume teas and coffees. In contrast, the younger age groups, 35 or below, mainly consume soft or energy drinks.

The first question (Figure 6) of the survey helped gauge how many of the participants consumed caffeinated drinks; as shown by the results, all surveyed participants consumed caffeinated beverages. This wasn't a surprise, as studies have shown that 95% of the population consumes at least one caffeinated drink once a week.

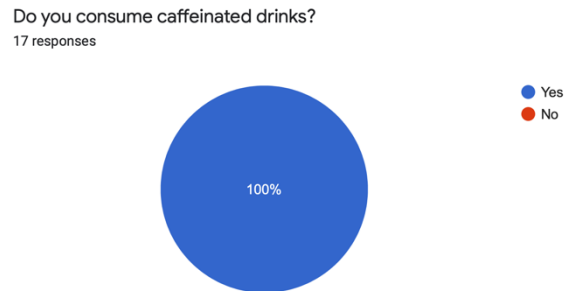


Figure 6: Question Results: Do you consume caffeinated drinks?

The next question (Figure 7) asked the participants which types of caffeinated drinks they usually consume, with the answer being split up into the appropriate groups. This question was asked to gauge which beverages are the most consumed and should be the focal point of the application. The results show that the primary beverage consumed is coffee and speciality coffees; therefore, the application will be slightly aimed at monitoring these drinks. While the results also show that the participants use all beverages, it is essential not to forget these either. The responses gathered in this question helped influence requirement F10 with the application allowing drinks added to be grouped into the categories asked in this question.

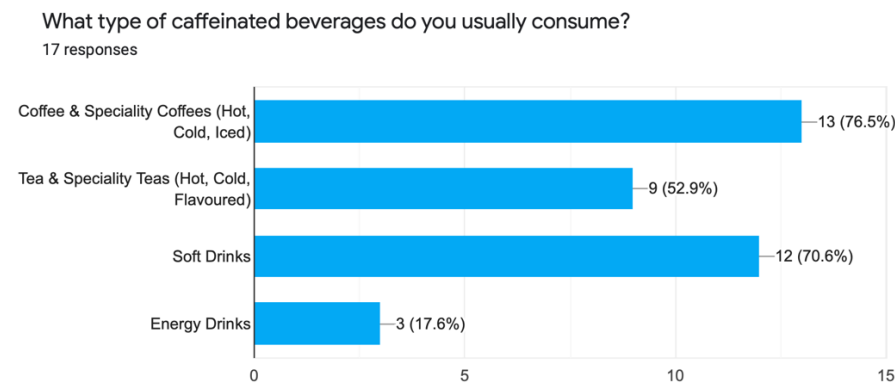


Figure 7: Questionnaire Results: What type of caffeinated beverages do you usually consume?

This question (Figure 8) asked the participants how many caffeinated drinks they consume daily. It was used to determine roughly how many drinks would be added to the system daily. Over 70% of participants recorded that they consume more than five caffeinated drinks daily, with nearly 30% consuming eight or more.

Roughly how many caffeinated drinks do you consume a day?

17 responses

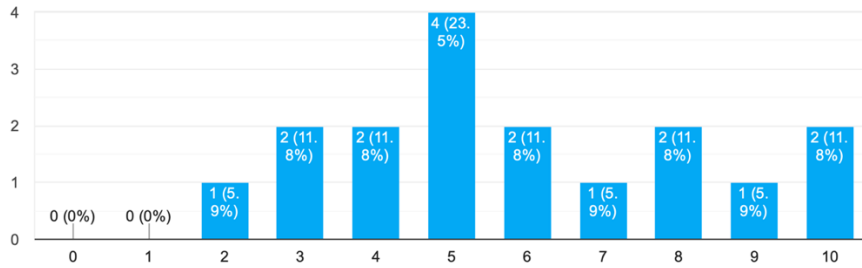


Figure 8: Questionnaire Results: Roughly how many caffeinated drinks do you consume a day?

This question (Figure 9) was designed to gauge how many participants knew how much caffeine was in their drinks'. The results show that only 35% of participants knew their drink's caffeine content. This highlights the importance of informing users how much caffeine is in their drinks and makes a stronger case to include drinks from major brands within the application. This question also raises concerns with users who might not know how to log their drink into the application if they do not know the caffeine content. Therefore, it might be useful to include helpful hints as to how users can find this information.

Are you aware of how much caffeine is in the drinks you consume?
17 responses

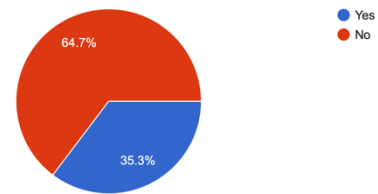


Figure 9: Questionnaire Results: Are you aware of how much caffeine is in the drinks you consume?

This question (Figure 10) was asked to see how many participants currently tracked their caffeine intake. This data was used to reaffirm the system's need further, as the results show that only 11.8% of responses now do so. Although this is a low percentage, it was somewhat expected, as tracking caffeine intake is not as mainstream as tracking calories or water.

Do you currently track your caffeine usage?
17 responses

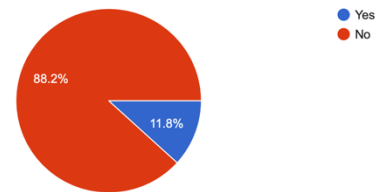


Figure 10: Questionnaire Results: Do you currently track your caffeine usage?

The next question (Figure 11) was designed to see how many participants would find it helpful to track their caffeine consumption. Of the responses, 52.9% said they would, and 35.3% said they might find it helpful. Considering that only 11.9% replied that they currently track their consumption, finding that 88% of participants would likely use the application further reaffirms the need for the system.

Would you find a system that make it easy to track your caffeine usage useful?
17 responses

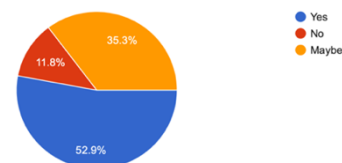


Figure 11: Questionnaire Results: Would you find a system that makes it easy to track your caffeine usage

4.3.2 Summary

After developing a questionnaire and analysing their results, the data was collated into functional and non-functional requirements, which are listed in the following section.

4.4 Requirements

Requirements for this project will be split between function and non-functional. Non-functional requirements outline underlying functions within applications, such as security, performance and usability and are crucial to making a system effective.

Non-Functional Requirements		
ID	Requirement	Priority
NF1	Support a minimum of iOS 14	Must
	The application should be able to run on any mobile device that runs iOS version 14.	
NF2	Responsive to all mobile screen sizes	Must
	The application should have a responsive user interface that is scaled to fit all iOS devices regardless of screen size.	
NF3	Easy to use	Must
	The application must be easy for all users to use.	
NF4	User-friendly interface	Should
	The user interface should be easy to navigate, and the user should not get lost in the application. The user interface should also be visually appealing and accessible.	
NF5	Follow Apple's Human Interface Guidelines	Must
	The application must follow all of Apple's Human Interface Guidelines (Apple, n.d.-j).	
NF6	Low or no internet connection	Should
	The application should continue to work seamlessly when the user has a weak or no internet connection.	
NF7	Security	Must
	The application must take all reasonable measures to keep user information secure. All applications uploaded to Apple's App Store or TestFlight systems must include encryption of user data.	
NF8	Code Documentation	Should
	All source code should be documented to a reasonable level.	
NF9	Data Backup	Could
	Data should be primarily available offline, with the option to backup data to a user's iCloud account should they request it.	

Table 2: Non-Functional Requirements

In addition to the non-functional requirements, functional requirements are also crucial by specifying precisely what the system should accomplish. They detail the system's basic facilities, and all these functionalities should be incorporated into the system.

Functional Requirements		
ID	Requirement	Priority
F1	Users must be able to access the system without login	Must
	Users must be able to open the application and be presented with the application without the need to login or register.	
F2	Ability to track caffeine	Must
	The user must be able to input the amount of caffeine they have consumed in one day. This can be done by any of these methods: <ul style="list-style-type: none"> Manually entering an amount of caffeine in milligrams (mg) Creating a custom drink where the user defines the parameters of the drink and how much caffeine there is 	
F3	Easy to track caffeine	Should
	The user should be able to track their caffeine consumption, as defined in F2, with relative ease.	
F4	Users can change caffeine metabolism preferences	Must
	The users must be able to alter the metabolism settings to best suit their bodies. There should be 'soft' limits on the settings to ensure appropriate values are entered.	
F5	Users must view privacy policy	Must
	Allow a privacy policy to be viewed within the application. This is a legal requirement for applications published to Apple's App Store.	
F6	Allow users to receive notifications	Should
	Users are to first be prompted to allow notifications and then receive notifications at set times throughout the day, which the user can customise.	
F7	Ability to view insights into caffeine intake	Should
	Users should be able to view various insights into their caffeine intake, such as averages, highs, lows, and various views in charts.	
F8	Onboarding	Could
	Allow new users to be guided through the application with an onboarding process, customising various settings using personal information, such as height and weight.	
F9	Deleting Drinks	Must
	The user must be able to delete any drinks added in case of mistakes.	
F10	Categorise Drinks	Could
	Drinks should be able to be grouped into categories, such as coffees and teas, to aid in analysis.	
F11	Ability to record branded drinks	Could
	Allow users to track drinks that are created by other companies, such as Starbucks or Coca Cola.	

Table 3: Functional Requirements

4.5 Summary

The requirements gathered above will be helpful in the latter stages of the project. They can be used as clear milestones of the development phase and, once implemented, will be tested using the methodology chosen. During the development stage, there is always the chance that requirements be changed slightly, and if this occurs, it will be stated in the implementation stage.

5. Design

The chosen project methodology, Personal Extreme Programming, requires detailed designs to ensure that the project is implemented efficiently. This chapter will document all design specifications for the application, including database designs, user interface mock-ups and testing plans.

5.1 Project Initiation

This section will detail core system functionalities to ensure the application is maintainable and usable.

5.1.1 Version Control

All source code will be stored under version control during the development phase. Doing so allows for high project maintainability with control over project revisions. GitHub will be used as it has high availability and great support for continuous integration (CI), allowing unit tests to be performed. GitHub will also be used to store any issues found during the implementation and testing stages.

The system will be built and tested under a 'dev' branch to ensure any changes made can easily be rolled back to a working version. Once committed and automated tests are completed, the branch will be merged into the 'main' branch.

5.1.2 Database

The application will be primarily developed for iOS devices and the chosen programming language, Swift, is the native development language for these devices. CoreData is an object graph and persistence storage framework provided for these systems (Apple, n.d.-a). CoreData allows seamless integration with CloudKit, allowing application data to be stored in the user's iCloud account for synchronisation between devices (Apple, n.d.-b).

CoreData will be used to store all the available drinks to be tracked, including any added by the user and a log of when the user consumed caffeine. This project will not be implementing an authentication system because all data is stored locally within the user's Apple ecosystem. CloudKit also provides the base framework should authentication be needed later.

A considered alternative was Google Firestore (Google, n.d.). Firestore is a real-time database which stores data in a JSON format. Whereas CoreData stores the user's mobile device data, Firestore stores data on a server. While keeping data on a server would help prevent any data loss from mobile data corruption, it requires a constant internet connection for the application to work. Since one of the requirements is to allow the application to be accessed from any location, CoreData was chosen as the system would still work offline, backing up data when an internet connection is available.

5.1.3 Security & Privacy Considerations

This section will explore different security issues and how they will be managed during the project development.

In addition to security considerations, numerous privacy issues need to be considered. Apple maintains a high focus on privacy, and numerous privacy issues had to be completed to allow the application to be uploaded to their App Store and TestFlight systems.

5.1.3.1 Local Data Storage

Data stored within the application that contains personal information, such as email addresses or passwords, would need to be kept securely. Since this application will not require the user to log in or register, personal data stored within the application will be minimal. If personal data is stored within the application, it will be held in the user's keychain using Apple's Keychain Service API (Apple, n.d.-c). This allows for small amounts of data, such as passwords or access tokens, to be stored within an encrypted database on the user's device.

5.2 Testing

Testing is significant to keep the products high quality and satisfy the end-user. Testing for this application will be done at various stages throughout the project and detailed below.

Testing will be completed both by myself and a group of alpha testers. The alpha testers will be responsible for providing feedback on designs and testing each build published.

5.2.1 Design Feedback

Throughout the designing process, high fidelity designs will be sent to the alpha testing group to gather feedback and help to improve the designs. Multiple designs will be created in some cases, and a 'favourite' is chosen. The group will look at various features of each design, including;

- layout
- accessibility
- features
- relevance to requirements

Their feedback will be used to produce the final artefact during the implementation stage.

5.2.2 Application Testing

The primary focus of the alpha testing group is to test the application on their own devices physically. Each build is published to Apple's TestFlight system, which allows applications to be distributed to testers and handle feedback and crash reports. With each build published, testing notes can be added to alert the testers on what features have been implemented or fixed and what they need to focus their time on.

5.3 Sequence Diagrams

A sequence diagram has been created to help visualise how certain functions are represented in the system. The diagram will show how processes and entities are involved with the application, which will be implemented into the application.

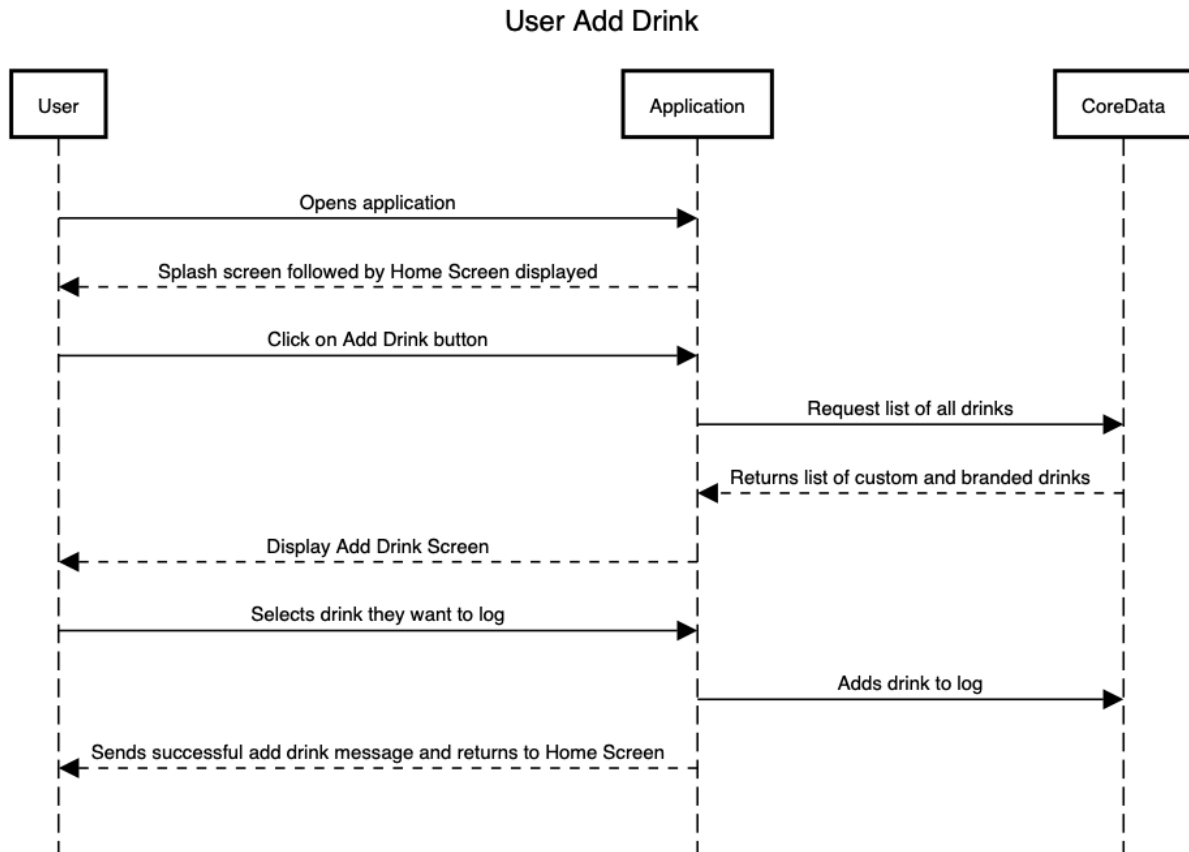


Figure 12: Sequence Diagram: User adding a drink to the system

The diagram above (Figure 12) shows the sequence of events if a user were to add a beverage they have consumed to the system.

5.4 User Interface Designs

This section will explore the different design aspects used as a reference during the implementation stage. All designs will follow Apple's Human Interface Guidelines (Apple, n.d.-j). The system's requirements in the previous chapter have helped draft these designs.

5.4.1 House Styles

Several house styles will be created and used to provide a consistent and clear user interface to ensure that the application is consistent in design.

5.4.1.1 Colour Scheme

The application will be designed with a simple and natural colour scheme. The primary colour used within this application design is light blue. With the rise of dark mode being used within devices, light blue fits well with both a light and dark mode setup.

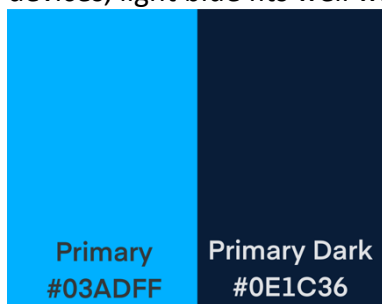


Figure 13: Application colour scheme

In addition to the primary colours shown above, colours are defined for various events throughout the application. Figure 14 shows the colour scheme used for all graphical representations of data. Higher caffeine consumption will result in data points changing colour up the scale, with red being used for the highest levels.

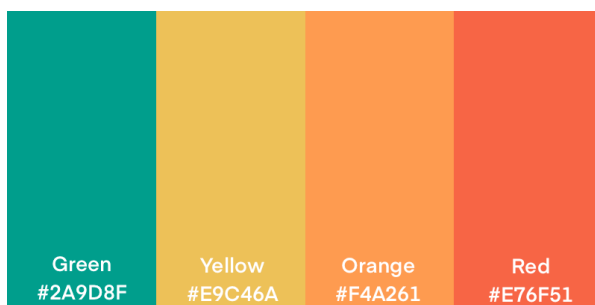


Figure 14: Insights Graphs Colour Scheme

5.4.1.2 Application Icon

An application icon is arguably the most vital feature when creating a brand. Simplicity is critical when designing an icon, and a complex icon might not fully engage the users. Some of the most popular brands have simple and easy to recognise icons. The icon should also be flexible, and work seamlessly on various forms of materials, such as applications and adverts, in light and dark mode.



Figure 15: Designs for Application Icon

The two icons above represent a certain degree of minimalism, which has become increasingly common alongside modern applications. Both icons are displayed on the primary colour detailed above.

5.4.1.3 Application Logo

Modern applications commonly have a logo to supplement an icon, usually containing the application icon alongside the application name. Again, a simple logo is far better than a more detailed one, and the logo designs aim to follow that.



Figure 16: Designs for Application Logo

The two logos above are both simple and feature the icons designed above alongside the application's name. Two logos were created and sent to the alpha testing group, gathering feedback. Overall, they found the second logo cleaner and thought it provided a better brand.

5.4.2 UI Wireframes

This section outlines the UI wireframes used for each main page. As the first physical designs of the interface, these low-level designs help provide a basic overview of how the interface would be displayed. Having these designs early in the life cycle is beneficial to the development, as later, more detailed designs are based on these initial ones.

Several simple wireframes were created at the start of the design process. These UI wireframes were designed with Procreate for iPad and detail the rough shape that each page offers.

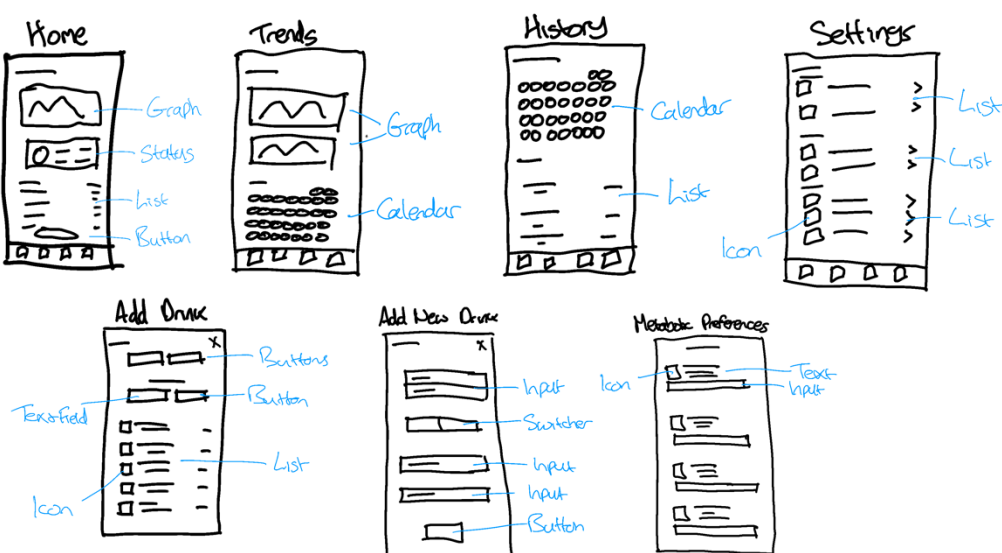


Figure 17: Initial hand UI Wireframes

The hand-drawn UI wireframes in Figure 17 shows how the application will look stripped down to basics. These have been inspired by previously researched applications (see literature review above) and are based on giving the user as much feedback as possible by using charts.

A UI Wireframe kit was used from these initial sketches to create a clearer version for each of the main pages, detailed over the following pages.

5.4.2.1 Home Screen

The first screen that the user will be presented with once opening the application will be the home screen. This will also be accessed by the user clicking the first tab in the navigation bar. A UI Wireframe is shown in Figure 18.

It will provide the user with the first look at their caffeine levels by showing a 'Highlights' panel. This panel will quickly inform the user how their current day is going and indicate how much caffeine they can consume throughout the day to stay within their limit.

Following this, a list will be displayed which shows all the drinks the user has consumed that day. It will display the drink name, date and time consumed and how much caffeine was in the drink.

The bottom of the screen will display a button, allowing users to add a new drink to their log.

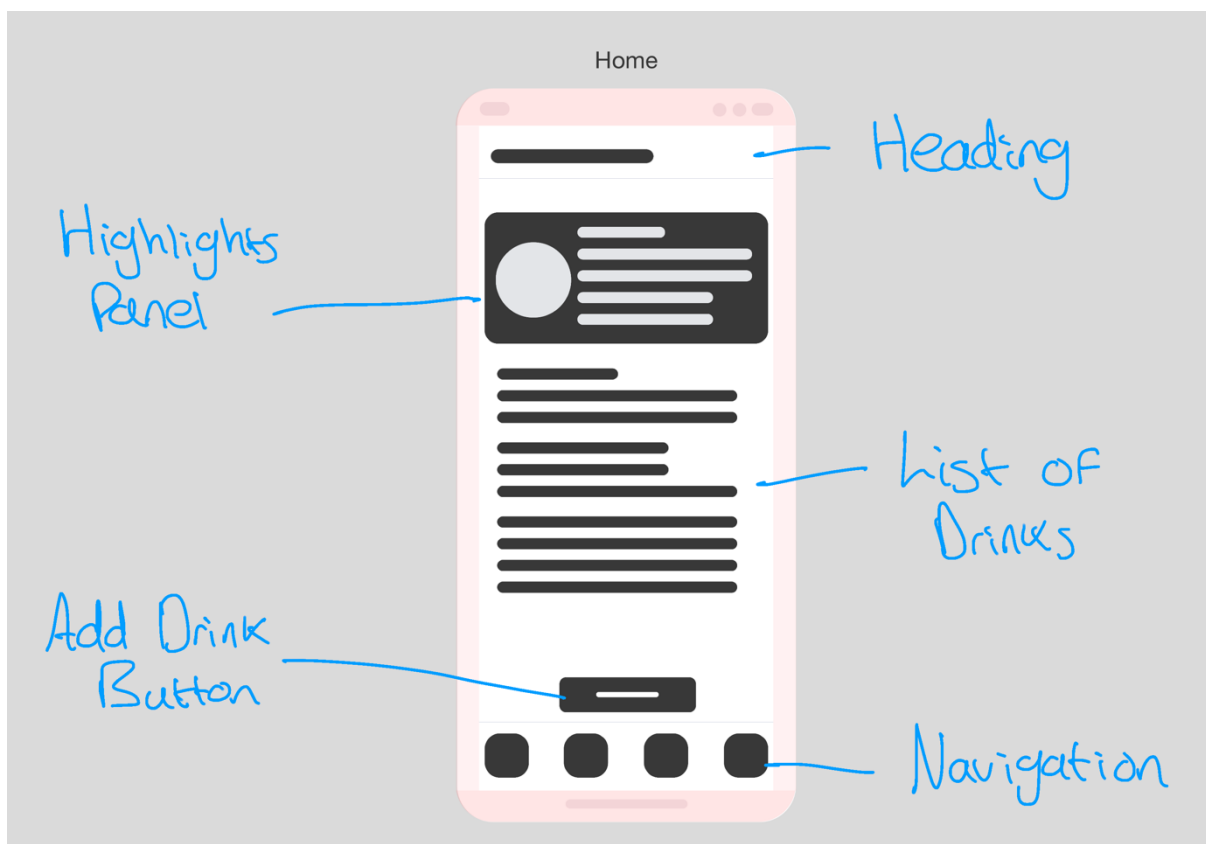


Figure 18: Annotated UI Wireframe for Home Screen

5.4.2.2 Trends Screen

The trends page allows users to understand their caffeine intake over time better. The trends screen will be the second tab available from the navigation bar. A UI Wireframe is shown in Figure 19.

The application will display various graphs highlighting the user's caffeine levels over time, such as the last four weeks. The charts will use the colour scheme detailed above to inform the user on days their caffeine levels were high.

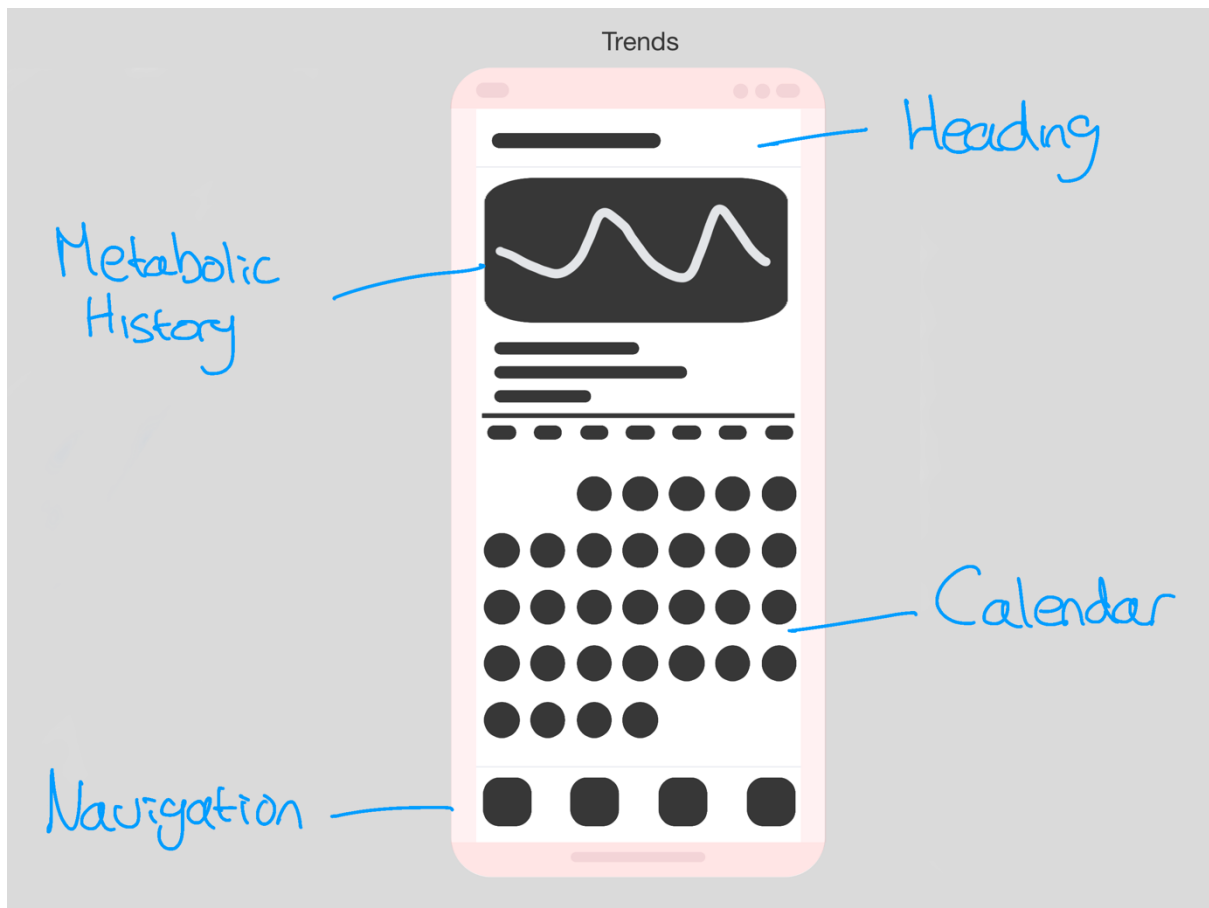


Figure 19: Annotated UI Wireframe for Trends Screen

5.4.4 High-Fidelity Designs

Following the UI wireframes, a series of high-fidelity user interface designs were created. These designs were made with either Procreate for iPad or through XCode. These designs will allow for more detailed styling of the application and show how individual components will be styled.

5.4.4.1 Home Screen Overview

A brief daily overview will be shown on the home screen, which will allow the user to get an insight into their current caffeine levels quickly. Initial designs were made to resemble the Activity section in Apple's Fitness app for iOS (Figure 20), and therefore provided a design that users were potentially already familiar with.

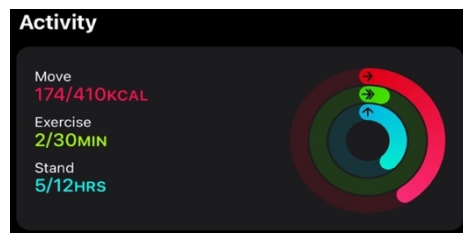


Figure 20: Activity section in Apple's Fitness app

Two different designs were created and sent to alpha testers to gather feedback on each style.

Highlights

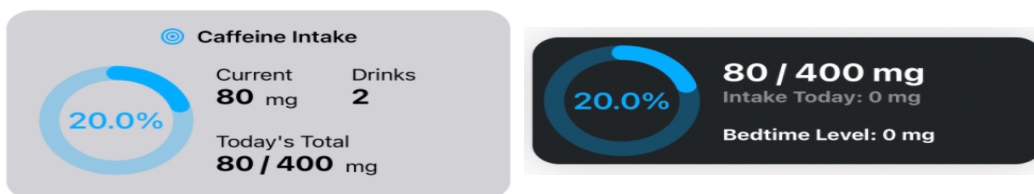


Figure 21: Caffeine Intake Highlight high-fidelity designs

In the first design, a light grey background was chosen, which would change to dark grey upon the user switching to dark mode. It features a main heading with a target icon. A circular bar will display their current caffeine percentage concerning their daily target. The right side details their current caffeine levels, calculated with their caffeine metabolic preferences, which change as the day goes on. It also shows their total amount of drinks consumed and a text indication of their caffeine intake against their limit for the day.

The second design remains a single colour, dark grey, in light and dark modes. This design also features a circular bar which displays their current caffeine percentage concerning their daily target. The right side details small information about their current caffeine levels, their total daily intake, and their caffeine levels at their set bedtime.

Following feedback from alpha testers, the first design was chosen as it better integrates with Apple's Human Interface Guidelines, specifically about being dark mode enabled, and offers more concise information.

5.4.4.2 Home Screen

A high fidelity design was created for the entire home screen following the overview panel detailed above. This is the most critical design since this is the single page that all users will access every time they access the application.

The Highlights section contains the detailed design as outlined above.

In addition to this, a list is shown containing all the drinks that the user has consumed that day. The drink's name is displayed alongside an icon to represent the drink visually. Different icons would be shown for other beverages, such as a can for soft drinks and cups for coffees.

The Add Drink button will be fixed to the bottom of the screen to ensure it constantly remains in one place and can be easily accessed with one hand. The button will be rounded to maintain consistency with the Highlights panel.

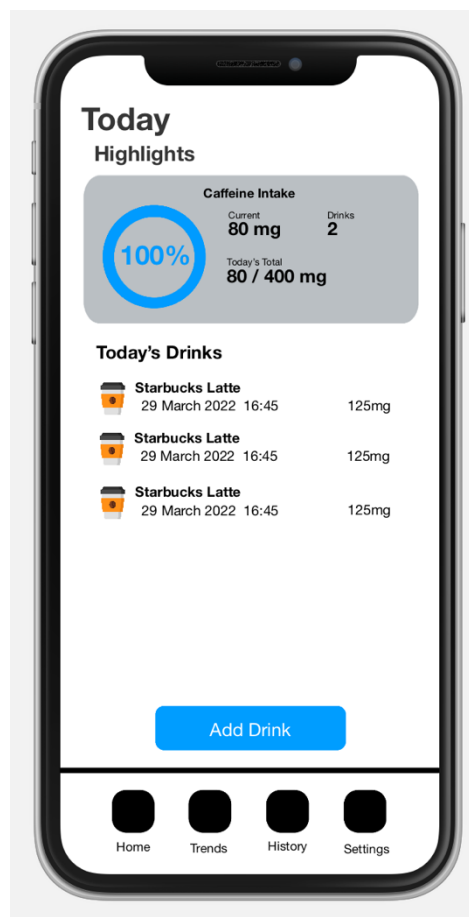


Figure 22: Home Screen High Fidelity Design

5.4.4.3 Trends Screen

This page shows a graphical visualisation of a user's caffeine intake. This page has been changed from the original wireframe design, as feedback suggested that only one page contains a calendar and that this page was the least necessary. The trends screen will be available from the second navigation tab.

The design below shows one chart detailing the user's caffeine intake over the past week. In the finished application, it is expected that there are more comparisons available, such as the past month and four-week average.

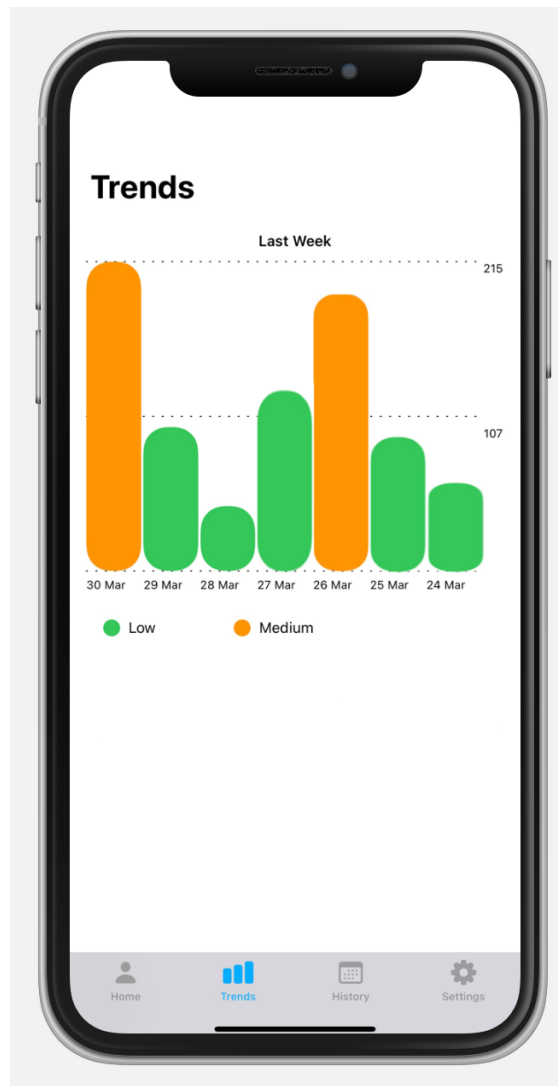


Figure 23: High Fidelity designs for Trends screens

5.4.4.4 History Screen

The history screen is accessed from the third tab in the navigation panel and shows the user their caffeine intake history. It features a calendar view displaying the current month, with controls allowing users to change the month. This idea builds on previously researched applications, showing an essential list requiring users to scroll to access earlier dates.

Following the calendar is a list, which shows the drinks consumed on the day the user has viewed the calendar. The list allows users to swipe right on each row if a drink needs to be deleted.

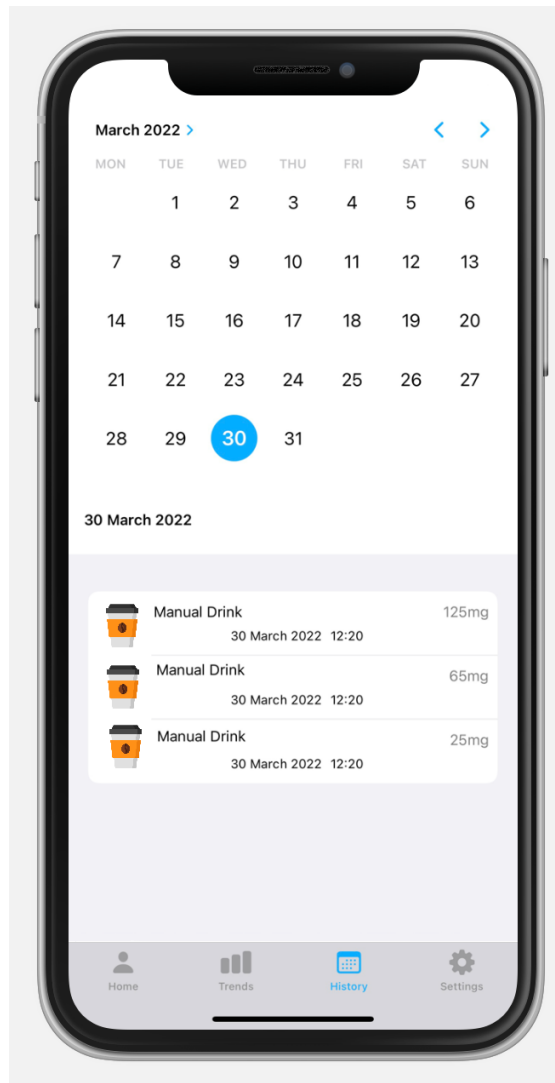


Figure 24: High Fidelity design for History screen

5.4.4.5 Add Drink Screen

The add drink screen will be presented as a modal accessible from the home view. The modals will be presented as 'Sheets', covering the entire page. Apple's Human Interface Guidelines recommend showing these pages as modals and state that presenting modals "Helps people focus on a self-contained task or set of closely related options" (Apple, n.d.-j).

The first modal presented shall display a button to access the second modal, allowing users to create custom drinks. In addition to this, the user can enter a manual caffeine quantity to add to their log should they wish to record their drink quickly.

The second modal allows users to add a custom drink, such as home-brewed coffee, to the application to track this drink in the future. It requires some basic details, such as the name and type of drink and the measurement method for the caffeine. Two types of measurements are available.

- **Fixed:** Allows users to create drinks with fixed caffeine content. This can be useful for drinks where the caffeine levels don't change, such as fixed-sized drinks from coffee shops or tablets.
- **Concentration:** This allows users to create more flexible drinks with caffeine content based on the size of a drink. This is more useful for drinks in various sizes and with caffeine content that changes accordingly. Examples of this include soft drinks and filtered coffees.

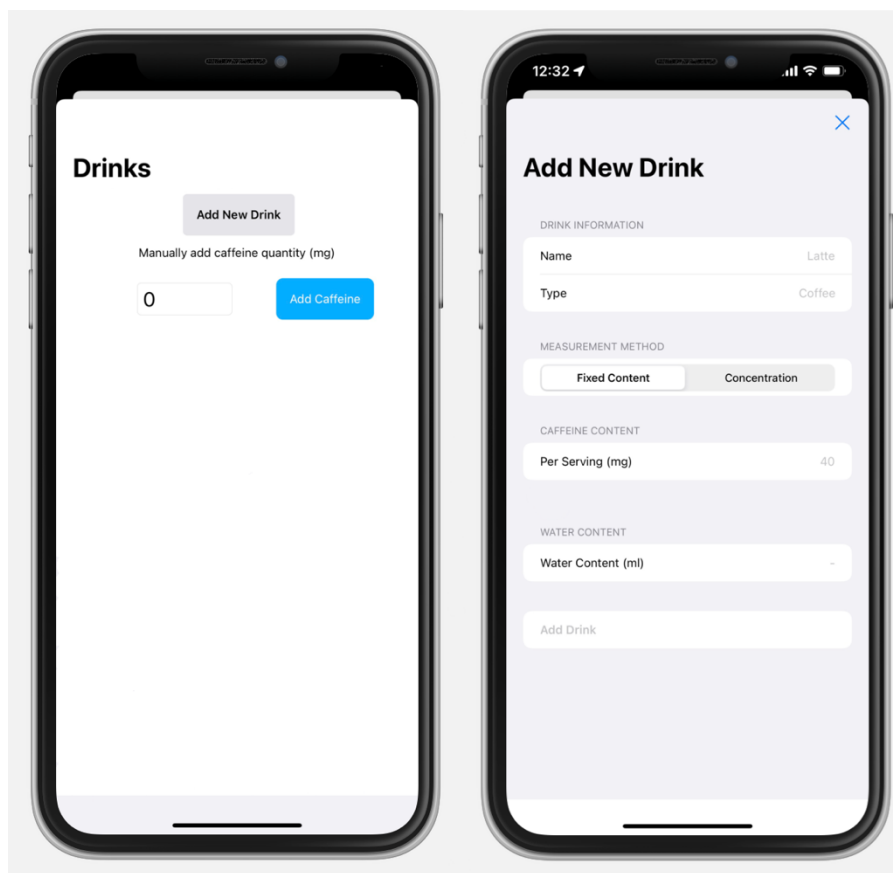


Figure 25: High Fidelity design for Adding Drinks

5.4.5 Testing on High-Fidelity Designs

All the high-fidelity designs created above were sent to the alpha testing group. This was to ensure that no essential points were missing from the designs, and if so, these would be added during the implementation stage. The designs were also analysed against the system requirements to ensure they had been met.

5.4.5.1 Changes based on Testing

Following the feedback on the high-fidelity designs, a few minor changes were made before the implementation stage.

The first change was in the history screen. The feedback retrieved suggested that the designs about the list of drinks would be better designed to exclude the date, as the date is available in the calendar above. They also thought the drink name should be clearer and more focused on the user. The resulting change shown in Figure 27 has a cleaner look but keeps the date within the view. This was done to reaffirm that the drink was consumed on that day but takes the focus away from the date and more towards the drink name, which is now in a bold font.

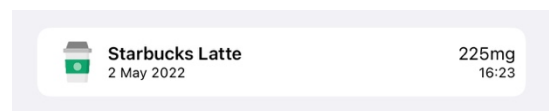


Figure 27: Changes made to the list of drinks in the History screen

The group suggested a second change in the trends screen. The original design was for the charts to take up about 70% of the screen; however, the feedback suggested that the charts were too big and should be made smaller. This change was made and can be seen in Figure 26.

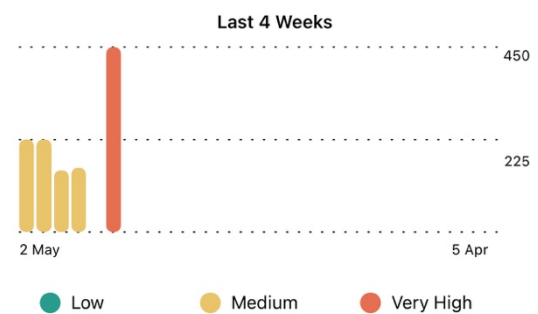


Figure 26: Changes made to the trends charts

5.5 Summary

This chapter used the previous requirements to construct a series of designs for the application. Low-fidelity designs were initially made to get an idea of the initial layout, and following this, more detailed high-fidelity designs were created. These designs were sent for user feedback, and a few changes were made to finalise them.

6. Implementation

This chapter outlines the implementation of the application, including back-end architecture and front-end interfaces.

6.1 Programming Language

As the application will be developed for iOS devices, two supported languages will be chosen from, Swift and Objective-C. Objective-C is the older of the two and is a member of the C programming language family. The Swift programming language was developed as a replacement for Objective-C and allowed for Objective-C, C, and C++ code to be compiled alongside Swift. Due to Swift having more modern features, such as the introduction of SwiftUI, Swift was chosen for this project.

6.2 Programming Resources

Since the author has little to no knowledge of the Swift programming language, further learning was needed to complete the implementation stage. This was necessary as Swift has a vast quantity of external libraries which could be used to enhance the application further; therefore, a better understanding was needed.

The first method used was Codecademy (Codecademy.com, n.d.). This contains an extensive catalogue of lessons for various programming languages, most free. There were three courses on the website for the Swift programming language, as shown in Figure 28. These courses served primarily as an introduction to the language and often would repeat content or explain previously known content.

The second website used was LinkedIn Learning (LinkedIn Learning, n.d.), which offered more advanced and often targeted learning. Experts often create these courses in their fields with a broad skill set. Two courses were chosen and completed before the implementation phase was started to solidify the knowledge of Swift better; these are shown in Figure 29.

These two courses were chosen as they were both recently updated with content from the latest Swift version and included SwiftUI, which will be used for the interface design of this application. After completion, there was more than enough knowledge to complete this project.

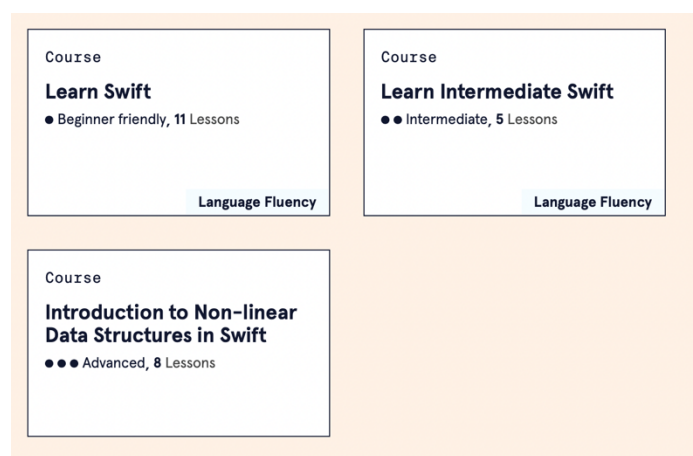


Figure 28: The three courses completed on Codecademy

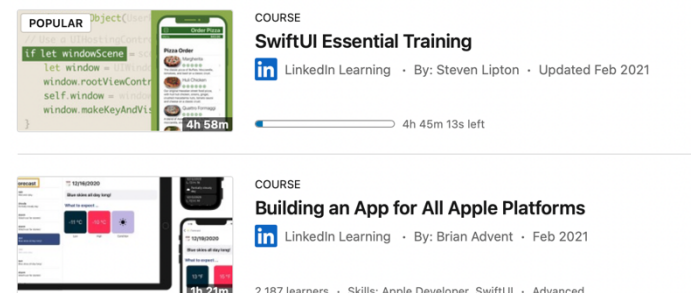


Figure 29: The two courses completed on LinkedIn Learning

6.3 Integrated Development Environment Choice

With the project being coded in Swift and based on iOS applications, it made sense to develop the application with XCode (Apple, n.d.-o). This is the default IDE used to develop iOS applications on Apple's macOS platform. The application is developed by Apple and generates the entire code base needed to begin development. Furthermore, the IDE is optimised to run on macOS systems and is very simple. Alternatives, such as Eclipse (Eclipse, n.d.) or IntelliJ IDEA (JetBrains, n.d.), would have worked for Windows platforms but would have encountered problems on macOS. The decision was made to stick with the Apple ecosystem to ensure the best compatibility.

Using a built-in simulator, which can run any Apple device, was vital as a physical device was not always available. This software helped with the front-end development by allowing a simulator to be shown within the development environment, saving time by removing the obligation to compile and run on a physical device. In the latter stages of development, a physical device was used to understand the user experience better.

6.4 Version Control

The design chapter details that the project will be stored using version control on GitHub to back up the project on a globally accessible platform. The repository was committed at the end of crucial development phases but did not represent any development milestones.

The repository is available here: <https://github.com/ryan-bush/MyCaffeine>

6.5 Common Components

The project will reuse several components to help provide higher code maintainability and consistency. From a development perspective, reusing components will be more efficient by preventing duplicated code and saving time.

6.5.1 Global Styling

Rather than externally storing and referencing the hex values used for colour generation, defining colours in one place is commonplace. This is done within the XCode IDE and iOS projects by utilising the Assets catalogue, which is a single place where all colours and images are stored.

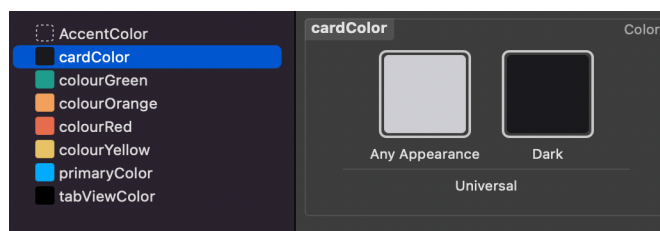


Figure 30: The Asset Catalogue showing colours, including the difference in light and dark modes

Each colour used is semantically named to improve code readability when referenced later. The Assets catalogue also allows the definition of light and dark colours, which can be used when users switch their devices between light and dark mode. This saves additional code within the user interface as colours are swapped automatically. Figure 30 shows the colours used within the project and an example of a single colour definition sharing two colours, one for light and one for dark modes.

6.5.2 Application-wide Icons

Similarly to the styling, the Asset catalogue is also used to store icons and images used throughout the application. Whereas colours can store light and dark mode colours, images stored in the catalogue contain three different styles. These styles are known as 1x, 2x and 3x and refer to the size difference from the original. For example, the image saved as 3x would be three times the size of the image stored as 1x. There are significant advantages when building an application that runs on multiple devices, such as an iPhone 6, compared to an iPhone 13 Pro Max, with vastly different screen sizes. The smaller phone would use the 1x image, whereas the larger would likely use the 3x. Doing so ensures that the image used is of the highest quality to the user.

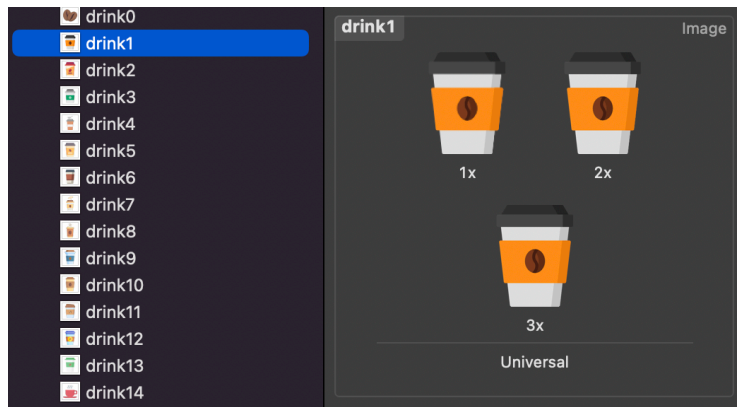


Figure 31: An example of how images are stored in the assets catalogue, including multiple sizes

The images saved to the Asset catalogue include the application logo and the icons defined later in the project. An example of how the images are stored can be seen in Figure 31.

6.5.3 Helper Functions

Helper functions are separate functions typically located in a different file that performs part of the computation of another function. They help make programs easier to read by giving more descriptive names and making it easier to reuse code. The helper functions for this project are in the file `Helpers` and can be found at:

MyCaffeine > MyCaffeine > Base > Helpers

6.5.3.1 Calculate Metabolism

Caffeine metabolism is calculated using the exponential decay formula ($P(t) = P_0e^{-rt}$). Since this is a more complex algorithm than others, it was in a helper function as it will likely be used in several places across the application. The function can be seen below in Figure 32.

```
static func calculateCaffeineRemaining(initial: Int, decay: Double, time: Double) -> Double {
    let rem = ((Double(initial) * pow(2, -(time/60) / decay)) * pow(2, M_E) / pow(2, M_E)
    return rem
}
```

Figure 32: Helper function to calculate caffeine metabolism

6.5.3.2 Retrieve Current Caffeine Level

Another algorithm within a helper function calculated the user's current caffeine level as a percentage. Although a relatively simple calculation, using a helper function helps to reduce the repetition of code. The function can be seen in Figure 33.

```

static func getCurrentCaffeinePercent(currentCaffeineLevel: Int, dailyCaffeineLimit: String) -> Double {
    let currentCaffeine = Double(currentCaffeineLevel)
    let dailyLimit = Double(dailyCaffeineLimit)!
    let percent = currentCaffeine / dailyLimit
    return percent
}

```

Figure 33: Helper function to retrieve the current caffeine percentage

6.5.3.3 Get Bedtime Caffeine Level

The last helper function to be detailed is calculating the user’s bedtime caffeine level. Doing this will allow the user to see how much caffeine will be in their body at their chosen bedtime, which users can change in the settings. The implementation of the function can be seen below in Figure 34.

```

static func getBedtimeCaffeineLevel(currentCaffeineLevel: Int, bedtime: String, decay: Double) -> Double {
    let currentCaffeine = Double(currentCaffeineLevel)
    // Get current Date
    let now = Date()
    // Split bedtime into HH and mm
    let dateSplit = bedtime.components(separatedBy: ":")
    // Apply bedtime to current date to get current date and bedtime
    let date3 = Calendar.current.date(bySettingHour: Int(dateSplit[0]) ?? 23, minute: Int(dateSplit[1]) ?? 30, second: 0, of: Date())!
    // Get difference between current and bedtime
    let difference = date3.timeIntervalSince(now)
    var caffeine = 0.0
    // Calculate caffeine remaining at bedtime
    if(difference > 0 && difference < 28800) {
        caffeine = calculateCaffeineRemaining(initial: Int(currentCaffeine), decay: decay, time: (difference / 60))
    }
    return caffeine
}

```

Figure 34: Helper function to retrieve the user’s bedtime caffeine level

6.6 Core Data

As detailed in chapter 5.1.2, Core Data would be used as the primary storage structure in the application. Using Core Data allowed for simple implementation, with an option to enable this done when first creating the project.

6.7 User Interface

When constructing the user interface, two frameworks exist when developing solely for iOS devices – UIKit (Apple, n.d.-d) and, more recently, SwiftUI (Apple, n.d.-m). These frameworks contain several user interface components and varying methods of implementation.

SwiftUI primarily uses programmatically constructed interfaces using a declarative syntax “so you can simply state what your user interface should do” (Apple, n.d.-m). SwiftUI also allows for changes such as size, font, and colour within the declarative syntax, allowing for quicker and more visual editing of components within the code. SwiftUI also enables an integrated live preview in XCode, allowing visual representation without compiling projects.

Both UIKit and SwiftUI code can be used interchangeably. The SwiftUI framework was chosen for this project as it allows for a faster development speed and enables a built-in live preview. Each user interface was clutter-free and minimalistic to allow greater flexibility.

The primary UI views are shown in the hierarchy below in Figure 35.

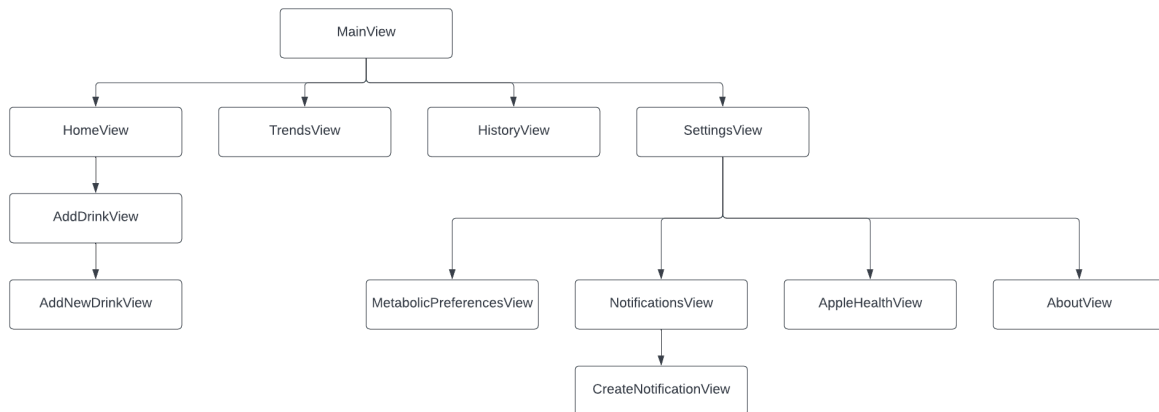


Figure 35: Major UI views of the application

6.7.1 Launch Screen

Since the first screen shown to a user is the launch screen, this was implemented first. This screen is displayed for approximately three seconds before the application loading; however, this could be shorter or longer depending on the device and the application loading time.

The implementation of the launch screen is relatively easy in XCode. A new storyboard was created from the new file tab and set as a launch screen. The background was then changed to the launch screen background defined in the assets catalogue, and the logo was added to the centre of the screen.

Apple requires a launch screen also conforming with its Human Interface Guidelines (Apple, n.d.-j). The implemented launch screen can be seen in Figure 36.



Figure 36: The implemented launch screen

6.7.2 Navigation

A navigation bar was created to assist users with navigating around the main pages of the application. A `TabView` (Apple, n.d.-e) was used as it easily allows for switching between multiple views and allows labels and images to be easily integrated. This was created on the `MainView` struct, which serves as the default screen when opening the application for the first time. By default, the `HomeView` is displayed to allow users to see their day overview immediately.

```
TabView {
  HomeView2(hydration: hydration, level: Data())
    .tabItem{
      Label("Home", systemImage: "person.fill")
    }
  InsightsView()
    .tabItem{
      Label("Trends", systemImage: "chart.bar.fill")
    }
  HistoryView()
    .tabItem{
      Label("History", systemImage: "calendar")
    }
  SettingsView()
    .tabItem{
      Label("Settings", systemImage: "gearshape.fill")
    }
}
.onAppear() {
  UITabBar.appearance().barTintColor = UIColor(Color("tabViewColor"))
  // On appearance, request authorization of HealthKit
  // (App works without authorization)
  health.requestAuthorization { success in
    if !success {
      print("Access not granted!")
    }
  }
}
.accentColor(Color("primaryColor"))
```

Figure 37: Implementation of the `TabView` within `MainView`

Each tab item contains a label and an icon, which have been chosen to allow better representation for the user of what the page does. This is also recommended in Apple's Human Interface Guidelines.

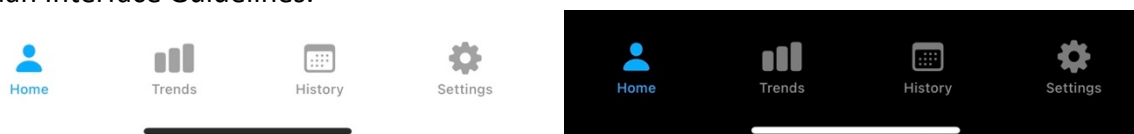


Figure 38: `TabView` Implemented as shown in light and dark mode

A light grey colour is applied to the background of the `TabView`, and the primary colour scheme is used for the label and icon.

6.7.3 Home View

The HomeView will be the application's landing page and, therefore, should be clutter-free so the user doesn't feel overwhelmed. The screen will consist of a highlights panel, which visually shows the user an overview of their current caffeine intake and a list of the user's drinks for the day. The user will be able to add drinks by clicking on an 'Add Drink' button.

6.7.3.1 Highlights Panel

The implementation of the highlights panel, as designed in chapter 5.4.4.1 Home Screen Overview, was the first focus of the home view.

After creating a blank card, with the colour set to vary between light and dark modes, a circle is drawn, representing the user's caffeine intake up to their maximum. By applying a `lineWidth` to the stroke syntax, the circle is turned into a 'ring' shape. The user's current caffeine percentage would then be retrieved, and then the circle is trimmed to match the percentage. The implementation of this can be seen below in Figure 39.

```
ZStack {
  Circle()
    .stroke(lineWidth: 15)
    .opacity(0.3)
    .foregroundColor(Color("primaryColor"))
  if (Helper.getCurrentCaffeinePercent(currentCaffeineLevel: getCurrentCaffeine(), dailyCaffeineLimit: dailyCaffeineLimit) * 100 > 100) {
    Circle()
      .trim(from: 0.0, to: Helper.getCurrentCaffeinePercent(currentCaffeineLevel: getCurrentCaffeine(), dailyCaffeineLimit:
        dailyCaffeineLimit))
      .stroke(style: StrokeStyle(lineWidth: 15.0, lineCap: .round, lineJoin: .round))
      .foregroundColor(Color("colourRed"))
      .rotationEffect(Angle(degrees: 270.0))
  } else {
    Circle()
      .trim(from: 0.0, to: Helper.getCurrentCaffeinePercent(currentCaffeineLevel: getCurrentCaffeine(), dailyCaffeineLimit:
        dailyCaffeineLimit))
      .stroke(style: StrokeStyle(lineWidth: 15.0, lineCap: .round, lineJoin: .round))
      .foregroundColor(Color("primaryColor"))
      .rotationEffect(Angle(degrees: 270.0))
  }
  Text("\(String(format: "%.1f", Helper.getCurrentCaffeinePercent(currentCaffeineLevel: getCurrentCaffeine(), dailyCaffeineLimit:
    dailyCaffeineLimit) * 100))%")
    .font(.system(size: 24))
    .bold()
    .foregroundColor(Color("primaryColor"))
}
.frame(width: 100, height: 100)
.padding()
.padding(.leading)
```

Figure 39: Implementation of the percentage circle on the highlights panel

The rest of the panel involves various text views that would display more information to the user, such as total drinks for the day and current caffeine level. The final implementation of the highlights panel can be seen below in Figure 40.

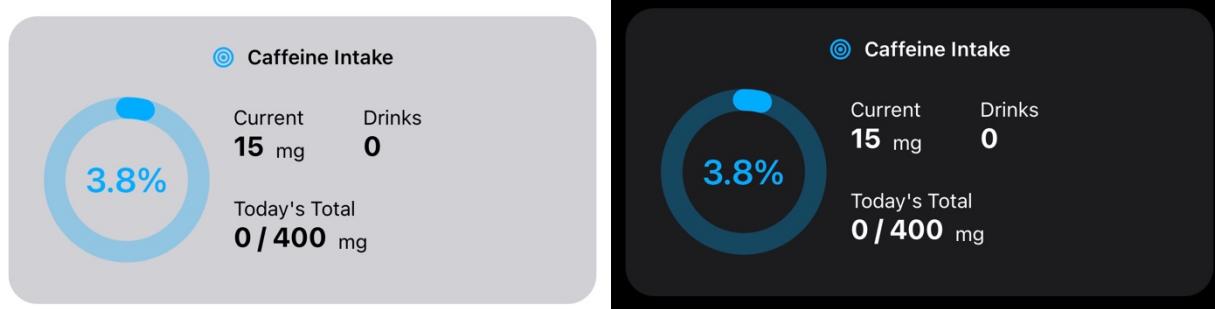


Figure 40: The result of the implementation of the highlights panel is light and dark modes

6.7.3.2 List of Drinks

Below the highlights panel, a list of all drinks the user has consumed today will be shown. The implementation of this was relatively easy using Core Data and List views. First, by checking if the list is empty, a text view can be displayed, alerting the user to add drinks. After this, the data is looped through to ensure the drinks' date matches today's date before showing the drinks in a list view and the icon and caffeine content.

Figure 41 and Figure 42 show how the list was implemented and the result within the application.

```
VStack(alignment: .leading) {
  ForEach(drinks, id:\.self) { drink in
    if Calendar.current.isDateInToday(drink.timestamp ?? Date()) {
      VStack(alignment: .leading) {
        HStack {
          Image(drink.imageID ?? "drink0")
            .resizable()
            .scaledToFit()
            .frame(width: 32, height: 32)
          VStack {
            HStack {
              Text("\(drink.name ?? "Unknown")")
                .font(.headline)
                .frame(maxWidth: .infinity, alignment: .leading)
              Text("\(drink.caffeineContent)mg")
                .frame(alignment: .trailing)
            }
            HStack() {
              Text(drink.timestamp ?? Date(), style: .date)
                .font(.caption)
              Spacer()
              Text(drink.timestamp ?? Date(), style: .time)
                .font(.caption)
                .frame(alignment: .trailing)
            }
          }
        }
      }
    }
  }
  .frame(height: 42)
  Divider()
}
```

Figure 41: Implementation of Today's Drinks in the HomeView

Today's Drinks



	Starbucks Latte 3 May 2022	225mg 12:32
	Pepsi 3 May 2022	39mg 17:29

Figure 42: Result of the implementation of Today's Drinks

6.7.3.3 Add Drink Button

The final component of the Home view was the 'Add Drink' button, which will float at the bottom of the screen. Using SwiftUI's Stack Views (Apple, n.d.-g), components can be layered

on top of each other to create a 'floating' effect. This was applied when creating the button. The implementation of this button can be seen below in Figure 43.

```
ZStack() {
  HStack() {
    Button(action: {
      showAddDrinkView.toggle()
    }, label: {
      Text("Add New Drink")
        .foregroundColor(.white)
        .frame(width: CGFloat(160), height: CGFloat(20), alignment: .center)
        .padding()
        .background(Color("primaryColor"))
        .cornerRadius(50)
    })
    .sheet(isPresented: $showAddDrinkView, content: {
      AddDrinkView(hydration: hydration)
    })
    .frame(maxHeight: .infinity, alignment: .bottom)
    .padding()
  }
}
```

Figure 43: Implementation of the 'Add Drink' button

6.7.4 Add Drink View

The AddDrinkView will serve as the location where users can log their drinks to the application. It will feature a list of all the user's custom added drinks and their icon and caffeine content. The list will be searchable to allow the user to find the drink they are looking for in a large list. There will also be a text field that allows the user to manually enter a caffeine quantity, up to 400mg, if the user wants to add a drink without creating a new drink quickly.

6.7.4.1 Manual Caffeine Entry

The first component implemented on the AddDrinkView will be the ability to add a quantity of caffeine into the application manually, without the need to track a specific drink. This will be used by users who are in a rush to track consumption.

A text field that only allows the users to enter numbers shows the Numpad keyboard on their device. By displaying this keyboard, it reduces the chance of user input error. There is then a quick check to ensure no invalid characters are entered by filtering out only numeric characters.

A button is located alongside the text field, which allows the user to log the quantity to their log. It first ensures the amount entered is above zero before referencing the Core Data object and saving the amount to the object. If the user enables this, the application then synchronises their entered caffeine quantity to their Health app. Finally, the sheet is closed, taking the user back to the HomeView.

The implementation of this component is shown in Figure 44.

```

VStack {
  Text("Manually add caffeine quantity (mg)")
  HStack {
    TextField("quantity", text: $quantity)
      .keyboardType(.numberPad)
      .onReceive(Just(quantity)) { newValue in
        if newValue.count > 3 {
          self.quantity = String(newValue.prefix(3))
        }
        let filtered = newValue.filter { "0123456789".contains($0) }
        if filtered != newValue {
          self.quantity = filtered
        }
      }
      .padding()
      .textFieldStyle(RoundedBorderTextFieldStyle())
      .font(.system(size: 24))
      .fixedSize()
      .frame(maxWidth: 200)
    Button {
      if(Int(quantity) ?? 0 > 0) {
        self.currentCaffeineLevel += Int(quantity) ?? 0

        let drinkLog = DrinkLog(context: managedObjectContext)
        drinkLog.name = "Manual Drink"
        drinkLog.caffeineContent = Int64(quantity) ?? 0
        drinkLog.type = "Manual"
        drinkLog.timestamp = Date()

        PersistenceController.shared.save()

        if(syncCaffeine == true) {
          self.hydration.logIntake(intake: Double(quantity) ?? 0.0, date: Date.now)
        }

        self.presentationMode.wrappedValue.dismiss()
      }
    } label: {
      Text("Add Caffeine")
        .foregroundColor(.white)
        .padding()
        .background(Color("primaryColor"))
        .cornerRadius(8)
    }
  }
}

```

Figure 44: Implementation of manual entry of caffeine

6.7.4.2 List of Custom Drinks

The next component implemented in this view shows the user a list of drinks created by the user. This was a simple component to implement, as it simply referenced the Core Data object and looped through all the drinks that the user had added. They are displayed in a list alongside their icon and caffeine quantity. If the user clicks on a drink, a Half Sheet will be displayed, revealing further information about the drink and displaying a button to add the drink to their log. If the drink created has a caffeine quantity that changes per drink size, then a TextStepper is shown, allowing the user to change the drink size. In addition to this, the list is searchable, which allows the user to find the drink they want if the list starts to grow.

The implementation of these components can be seen in Figure 45 and Figure 46.

```

List {
  ForEach(drinks, id:\.self) { drink in
    Button(action: {
      sheetDrinkName = drink.name ?? ""
      sheetDrinkType = drink.type ?? ""
      sheetDrinkImage = drink.image ?? ""
      sheetDrinkFixedContent = drink.fixedContent
      sheetDrinkCaffeine = Int(drink.caffeineContent)
      sheetDrinkWater = Int(drink.waterContent)
      sheetDrinkTime = drink.timestamp ?? Date()
      isPresented.toggle()
    }) {
      HStack {
        Image(drink.image ?? "drink1")
          .resizable()
          .scaledToFit()
          .frame(width: 32, height: 32)
        Text("\${drink.name ?? "Unknown"}")
          .badge("\${drink.caffeineContent}mg")
      }
    }
  }
}
.onDelete(perform: deleteItem)
}
.searchable(text: $searchText)
.onChange(of: searchText) { newValue in
  drinks.nsPredicate = newValue.isEmpty ? nil : NSPredicate(format: "name CONTAINS %@", newValue)
}
}

```

Figure 45: Implementation of the list showing users created drinks

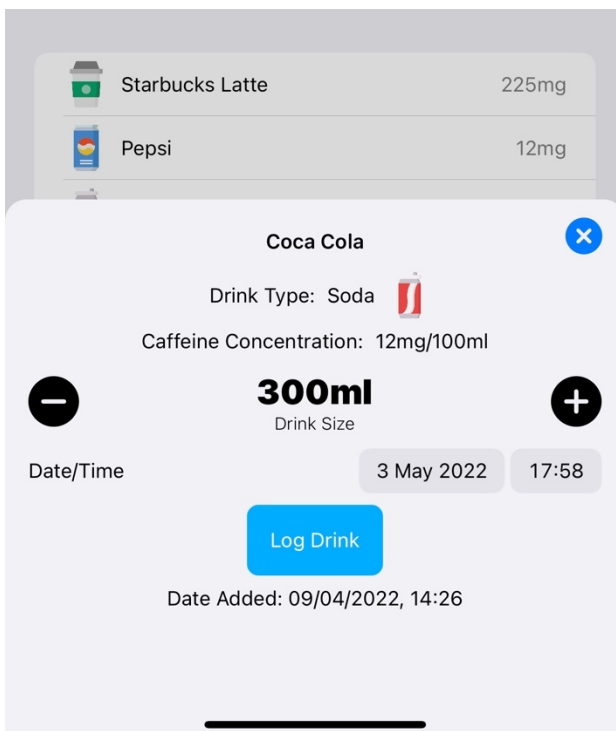


Figure 46: Result of the implementation of the list view with the HalfSheet and TextStepper

6.7.5 Trends View

The trends view will be home to a couple of graphs to help users understand their caffeine consumption over a short period. The charts will use the SwiftUICharts library (mecid, n.d.), as detailed in chapter 6.8.3 SwiftUICharts. Initially, two charts will be available to view, one describing the user's caffeine consumption over the past week and the other for the past month. The charts will use the colour scheme set out in chapter 5.4.1.1 Colour to display to the user the days their caffeine consumption is high.

Figure 47 below shows how the chart was implemented. It starts by looping through the past seven days and retrieving the caffeine content of every drink consumed on that day. The points are then appended to the chart data structure after being assigned a colour based on the user's daily caffeine limit. A similar approach was followed for the monthly chart but looping through the last 28 days.

```
func buildLastWeekChart() {
    let cal = Calendar.current
    var date = cal.startOfDay(for: Date())
    var days = [Int]()
    var points = [Int]()

    for _ in 1 ... 7 {
        var dayTotal = 0
        let day = cal.component(.day, from: date)
        days.append(day)

        for drink in drinks {
            if(Calendar.current.isDate(drink.timestamp ?? Date(), inSameDayAs: date)) {
                dayTotal += Int(drink.caffeineContent)
            }
        }
        points.append(dayTotal)
        if(Int(dayTotal) < (Int(dailyCaffeineLimit) ?? 400)/3) {
            lastWeekChart.append(.init(value: Double(dayTotal), label: "\(date.formatted(.dateTime.month(.abbreviated).day()))", legend: low))
        } else if(Int(dayTotal) > ((Int(dailyCaffeineLimit) ?? 400) / 3) && Int(dayTotal) < ((Int(dailyCaffeineLimit) ?? 400) / 3) * 2) {
            lastWeekChart.append(.init(value: Double(dayTotal), label: "\(date.formatted(.dateTime.month(.abbreviated).day()))", legend: medium))
        }
        else if(Int(dayTotal) > ((Int(dailyCaffeineLimit) ?? 400)/3) * 2 && Int(dayTotal) < Int(dailyCaffeineLimit) ?? 400) {
            lastWeekChart.append(.init(value: Double(dayTotal), label: "\(date.formatted(.dateTime.month(.abbreviated).day()))", legend: high))
        }
        else if(Int(dayTotal) > Int(dailyCaffeineLimit) ?? 400) {
            lastWeekChart.append(.init(value: Double(dayTotal), label: "\(date.formatted(.dateTime.month(.abbreviated).day()))", legend: veryHigh))
        }
        date = cal.date(byAdding: .day, value: -1, to: date)!
    }
}
```

Figure 47: Implementation to build a chart for caffeine consumption over the past week

6.7.6 History View

The history view will allow the user to go back in time and view their specific caffeine consumption on any given day. There will be a calendar which the user can navigate to select the particular day they wish. On choosing the date required, a list is shown below detailing the drinks they consumed on that day.

The list is built similarly to the one displayed on the HomeView, looping through each drink and checking if the calendar date matches the user's selected date. Each drink is shown with an icon and caffeine quantity.

```
List {
  ForEach(drinks, id:\.self) { drink in
    if(Calendar.current.isDate(drink.timestamp ?? Date(), inSameDayAs: date)) {
      VStack(alignment: .leading) {
        HStack {
          Image(drink.imageID ?? "drink0")
            .resizable()
            .scaledToFit()
            .frame(width: 32, height: 32)
          VStack {
            HStack {
              Text("\(drink.name ?? "Unknown")")
                .font(.headline)
                .frame(maxWidth: .infinity, alignment: .leading)
              Text("\(drink.caffeineContent)mg")
                .frame(alignment: .trailing)
            }
            HStack() {
              Text(drink.timestamp ?? Date(), style: .date)
                .font(.caption)
              Spacer()
              Text(drink.timestamp ?? Date(), style: .time)
                .font(.caption)
                .frame(alignment: .trailing)
            }
          }
        }
      }
    }
  }
  .onDelete(perform: deleteItem)
}
```

Figure 48: Implementation of the list shown on the History View

The list on this view also allows users to delete the drink from their log. Deleting items from data structures is typically a difficult feat; however, Core Data makes it easy. The syntax `onDelete` is appended to the List with a function to be performed. The function takes the offset index of the item deleted, loops through it, and removes it.

```
func deleteItem(at offsets: IndexSet) {
  for index in offsets {
    let item = drinks[index]
    PersistenceController.shared.delete(item)
  }
}
```

Figure 49: Implementation required to delete the item from the list

6.7.7 User Notifications

It was deemed necessary from the application's requirements to send notifications to the user when they choose.

The user will be able to customise when they receive notifications from a page located in the Settings View. If the user hasn't visited this page before then, the application will ask the user for permission to send notifications. The user can then create a notification at a time to suit them, and once created, the list is updated. The user can also delete notifications in the same method as deleting drinks in the previous chapter.

The application will use SwiftUI's `UNNotificationManager` (Apple, n.d.-f) to authorise, load, create and send notifications. The implementation can be seen below in Figure 50.

```
final class NotificationManager: ObservableObject {
    @Published private(set) var notifications: [UNNotificationRequest] = []
    @Published private(set) var authStatus: UNAuthorizationStatus?

    func reloadAuthStatus() {
        UNUserNotificationCenter.current().getNotificationSettings { settings in
            DispatchQueue.main.async {
                self.authStatus = settings.authorizationStatus
            }
        }
    }

    func requestAuthorization() {
        UNUserNotificationCenter.current().requestAuthorization(options: [.alert, .badge, .sound]) { isGranted, _ in
            DispatchQueue.main.async {
                self.authStatus = isGranted ? .authorized : .denied
            }
        }
    }

    func reloadLocalNotifications() {
        print("reloadLocal")
        UNUserNotificationCenter.current().getPendingNotificationRequests { notifications in
            DispatchQueue.main.async {
                self.notifications = notifications
            }
        }
    }

    func createLocalNotification(hour: Int, min: Int, completion: @escaping (Error?) -> Void) {
        var dateComponents = DateComponents()
        dateComponents.hour = hour
        dateComponents.minute = min

        let trigger = UNCalendarNotificationTrigger(dateMatching: dateComponents, repeats: true)

        let notificationContent = UNMutableNotificationContent()
        notificationContent.title = "Record your caffeine intake now!"
        notificationContent.sound = .default
        notificationContent.body = "Don't forget to record your caffeine intake and stay on track!"

        let request = UNNotificationRequest(identifier: UUID().uuidString, content: notificationContent, trigger: trigger)

        UNUserNotificationCenter.current().add(request, withCompletionHandler: completion)
    }

    func deleteLocalNotifications(identifiers: [String]) {
        UNUserNotificationCenter.current()
            .removePendingNotificationRequests(withIdentifiers: identifiers)
    }
}
```

Figure 50: Implementation of Notifications

6.7.8 Apple Health Integration

Along with notifications, it was requested in the requirements that the user can synchronise their caffeine intake to Apple's Health application. This is all done using HealthKit, which provides a central repository for health and fitness data on iPhone and Apple Watch (Apple, n.d.-g).

Along with notifications, it was requested in the requirements that the user can synchronise their caffeine intake to Apple's Health application. The user will be asked to provide access to allow the application to write data to their Caffeine and Water data. If the user denies permission, then the application will not write any data through HealthKit, and the user would have to enable this through the Health app in the future. The application will use a HealthKitManager to contain all the functions required to initialise, authorise, and save data to HealthKit.

Initially, an HKHealthStore object is made to request permission to share data, and once granted is used to save new samples to the store. A check is first made to ensure that HealthKit is available on the device, as it is not available on devices that run macOS or iPadOS. The authorisation function asks the user for permission to write caffeine and water data to HealthKit and is called when the application is opened for the first time. The implementation of these functions is shown below in Figure 51.

```
@Published var store:HKHealthStore?

init() {
    if HKHealthStore.isHealthDataAvailable() {
        store = HKHealthStore()
    } else {
        fatalError("HealthKit not available on this platform!")
    }
}

func requestAuthorization(completion: @escaping (Bool) -> Void) {
    let caffeineType = HKQuantityType.quantityType(forIdentifier: HKQuantityTypeIdentifier.dietaryCaffeine)!
    let waterType = HKQuantityType.quantityType(forIdentifier: HKQuantityTypeIdentifier.dietaryWater)!

    let toShare = Set([
        caffeineType, waterType
    ])

    guard let store = self.store else {
        return completion(false)
    }

    store.requestAuthorization(toShare: toShare, read: []) { (success, error) in
        completion(success)
    }
}
```

Figure 51: Implementation of HealthKit functions

Two functions are then created to save the data to HealthKit, once for water and one for caffeine. The implementation of the caffeine function is shown below in Figure 52.

```

func saveCaffeineData(amount: Double, date: Date) {
    let quantityType = HKQuantityType.quantityType(forIdentifier: HKQuantityTypeIdentifier.dietaryCaffeine)

    if let store = self.store {
        let newSample = HKQuantitySample.init(type: quantityType!, quantity: HKQuantity.init(unit: HKUnit.gramUnit(with: .milli), doubleValue: Double(amount)), start: date, end: date)
        store.save(newSample) { (success, error) in
            if (error != nil) {
                print("ERROR WITH SAVING: \(String(describing: error))")
            }
            if success {
                print("SAVED: \(success)")
            }
        }
    }
}

```

Figure 52: Implementation of HealthKit function to save caffeine data

6.8 External Libraries

Several external libraries were used to enhance the application and reduce implementation time. These libraries are detailed below.

6.8.1 SF Symbols

SF Symbols is a library of iconography designed to integrate seamlessly with San Francisco, the system font for Apple platforms (Apple, n.d.-k). The symbols can be customised in font-weight and scale and automatically align with text added to the user interface. These symbols will be used throughout the application, such as TabBar.

6.8.2 Flaticons

In the design phase, icons were used alongside drinks so that users could visually compare different beverages. A series of icons were used from Flaticons (Flaticon, n.d.). These images all conform to the same style to enhance the user interface and include a range of drinks. The user will be able to select which icon they want to associate with specific beverages themselves. Some of the icons used can be seen in Figure 53.



Figure 53: A few of the icons used from Flaticons

6.8.3 SwiftUICharts

To give the user a better representation of their caffeine consumption over a period, the application will use charts to display information. Designing charts from scratch would have taken a large portion of the implementation stage; therefore, it was decided early on that an external library would be used to simplify this process.

SwiftUICharts (mecid, n.d.) was chosen as it offers an easy implementation method and great customisation features. It allowed scaling colours to be used and defined in the design process.

6.8.4 HalfASheet

SwiftUI includes Sheets (Apple, n.d.-h) which can be used to display a view over an existing view. A sheet helps people perform a distinct task that's related to the parent view without taking them away from their current context (Apple, n.d.-l). This method was used when displaying the Add Drink View over the Home View, as it allowed the user to easily back out of adding a drink without reloading the view.

It was deemed necessary in some locations that a full-screen sheet was not required, and instead, a smaller one was ideal. SwiftUI's Sheets can be set to cover half a screen and must support full screen; therefore, it was necessary to find a library with custom sheets.

HalfASheet (franklynw, n.d.) allows for precisely this. It can be set as a whole sheet or resized to fit a proportion of the screen. This method was used when users confirmed what drink they wanted to log in and for help navigating the metabolic preferences within the settings view.

6.8.5 TextFieldStepper

TextFieldStepper (joe-scotto, n.d.) is a component that makes inputting numbers easier than SwiftUI's Stepper (Apple, n.d.-i). It will be used when the user inputs the size of drink they wish to consume, as it allows for quicker changing of larger numbers.

6.9 Summary

This chapter has detailed several of the methods used throughout the implementation stage. It describes the resources and software used to create the project and the methods used to implement the features. Following this chapter will be a testing phase to ensure the application works as expected and that there are no errors.

7. Testing

This chapter will detail the tests undertaken throughout the implementation stage to ensure that the application runs as expected and can provide a quality user experience.

Testing software is vital to ensure quality, safety, and reliability. Different types of tests will be carried out to gauge what parts of the application work best and need improvements.

7.1 Primary Testing Methods

The primary testing method used during the implementation stage was to release builds to Apple's TestFlight system (Apple, n.d.-n). This allowed up to 10,000 testers to get early previews of the application and provide feedback. For this project, it was decided that releasing regular builds of the application to the TestFlight system would be massively beneficial. It removes the pressure on the sole developer to test and instead allows them to focus solely on implementation.

A total of 33 preview builds were uploaded to the TestFlight system for testers to access from October 2021 to May 2022.

A brief set of update notes accompanied each build presented the first time they opened the application after updating. This was used to alert the testers about what had changed and been fixed in the update to allow accurate testing of features. Testers could provide feedback via the iOS system and offer screenshots or recordings of bugs found. By reporting issues, additional information was collected about the tester's device, such as screen size and device, which would further help resolve the problems.

A total of 63 pieces of feedback were collected this way, which proved substantial in resolving bugs.

In addition to this testing method, GitHub issues were used to track bugs found during the implementation stage by the developer. This allowed problems to be tracked and fixed before being released to the TestFlight system. Only two issues were reported this way, with both being marked as 'enhancement' rather than an actual bug.

7.2 Functionality Testing

Functionality testing concentrated on the system's functionality as represented in the functional requirements. Tests were run on simulated devices and a physical iPhone 12 Pro Max. In total, five different simulated devices were used, with a varying screen-sized, with the devices used being:

- iPhone 8
- iPhone 11
- iPhone 11 Pro
- iPhone 13 Pro Max
- iPhone SE

This allowed for most currently used screen sizes to be tested. The results of the functionality testing are shown in Table 4.

ID	Description	Expected Result	Result	Grading
T01	Loading Application	The user is shows the home view	As expected	Pass
T02	Application Navigation	The user will be presented with the corresponding view after clicking on the TabView	As expected	Pass
T03	Application Permissions	Application asks for permissions for notifications and Apple Health	As expected	Pass
T04	Creating Custom Drink	Application allows the user to create a new custom drink, with name, category, icon, measurement method, caffeine and optionally water.	As expected	Pass
T05	Log Custom Drink – Fixed Size	Application allows the user to log a custom drink with a set drink size	As expected	Pass
T06	Log Custom Drink – Fixed Concentration	Application allows the user to select a drink size and adjusts the caffeine quantity	As expected	Pass
T07	Trends Chart	Application automatically updates the Trends Chart after each change to caffeine log	As expected	Pass
T08	Delete Drink	Application allows the user to swipe to delete a drink from the History View	As expected	Pass
T09	Notifications	Application allows users to create notifications at a time that suits them, and the application notifies them at that time.	As expected	Pass
T10	Metabolic Preferences	Application allows the user to change various metabolic preferences to better suit themselves	As expected	Pass
T11	Apple Health Integration	Application allows the user to synchronise data from the application to their Health app	As expected	Pass

Table 4: Results of Functionality Testing

The initial testing stage was a success, with all primary functions passing. This indicates that the application works as expected and is bug-free.

7.3 Responsiveness Testing

As MyCaffeine is an iOS application, it must work on various devices with different shapes and sizes. The user interfaces must be designed to allow the view to work seamlessly on any screen size. Figure 54 shows how the application looks when running on devices with different screen sizes.

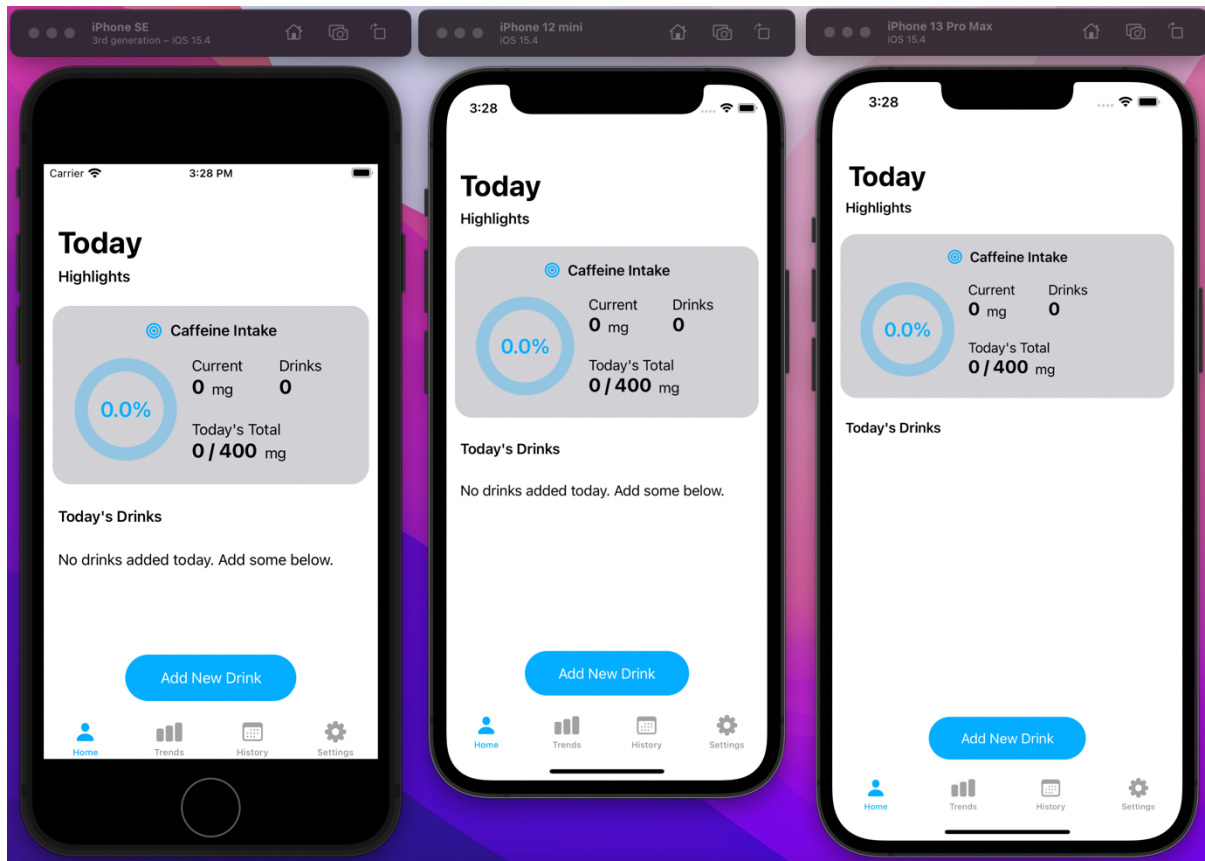


Figure 54: Application running on three different simulated devices

The test shows that the application looks acceptable on all devices but looks better on larger devices. This was somewhat expected as the application was primarily developed using an iPhone 12 Pro Max with a larger screen.

7.4 Summary

This chapter detailed the testing of the application developed during the previous stage. The primary goal was to ensure the application was error-free, with no significant issues. The application was also extensively tested with a group of alpha testers on the TestFlight system to provide excellent user testing. The next stage will evaluate the application, looking back at the initial requirements.

8. Evaluation

This chapter evaluates how successfully the project was implemented and examines the initially planned requirements. This chapter will also reference the original project initial document (PID) (Appendix A) to analyse how close the final artefact resembles the initial plans.

8.1 Evaluation against Requirements

The application was evaluated against the requirements set out in chapters **Error! Reference source not found.** and **Error! Reference source not found.**. The results are shown in Table 5. All the requirements marked with a priority of Must or Should were met, with three marked as Could failing.

Identifier	Priority	Requirement	Requirement Satisfied
NF1	Must	Support a minimum of iOS 14	Pass
NF2	Must	Responsive to all mobile screen sizes	Pass
NF3	Must	Easy to use	Pass
NF4	Should	User-friendly interface	Pass
NF5	Must	Follow Apple's Human Interface Guidelines	Pass
NF6	Should	Low or no internet connection	Pass
NF7	Must	Security	Pass
NF8	Should	Code Documentation	Pass
NF9	Could	Data Backup	Fail
F1	Must	Users must be able to access the system without login	Pass
F2	Must	Ability to track caffeine	Pass
F3	Should	Easy to track caffeine	Pass
F4	Must	Users can change caffeine metabolism preferences	Pass
F5	Must	Users must view privacy policy	Pass
F6	Should	Allow users to receive notifications	Pass
F7	Should	Ability to view insights into caffeine intake	Pass
F8	Could	Onboarding	Fail
F9	Must	Deleting Drinks	Pass
F10	Could	Categorise Drinks	Pass
F11	Could	Ability to record branded drinks	Fail

Table 5: Evaluation against Requirements

8.1.1 Failed Requirements

As shown in the table above (Table 5), three requirements were not successfully implemented into the application. The primary reason these were missed was the timeframe, with more time allocated to the requirements that had a higher priority. In addition, the potential legal requirements needed to add branded drinks into the project simply meant it wasn't feasible for this project once the implementation had started.

Even though three requirements have failed, the project is still a completed application.

8.2 Methodology Evaluation

Before the project started, a methodology was chosen to set up the latter stages. The methodology used has been relatively successful, with changes made during the testing phase to help accommodate the remote development. Alongside this methodology, a GANTT chart was created (Appendix C) to help with time management.

8.3 Time Management Evaluation

Time management was critical with such a short timeframe to complete a project of this scale. Regular meetings were scheduled with their supervisor throughout the project to ensure the project was on track and see if any help was needed. These meetings also allowed the project to remain at the front of the schedule regarding other work completed during the academic year.

Overall, the project progressed slowly in the initial months, heavily influenced by personal reasons and the COVID-19 pandemic. These both affected the ability to visit campus and access additional resources.

The project's timeline (Appendix C) was not kept to, with initial items being pushed back significantly, resulting in a shortened implementation period. In the latter stages, more time was spent throughout the day to ensure development targets were still met to mitigate this.

8.4 Future Work

After reflecting on the finished application and its drawbacks, there is the potential for large amounts of future work to help polish the application more. This being said, the application could quickly be released now without many drawbacks from a user perspective.

The biggest drawback is the lack of branded drinks in the application. This would have made it much easier for users to easily add drinks they commonly consume, especially since they hide their caffeine content. Ultimately, the unknown legal requirements made it impossible to add this during the project's timeframe; however, this would be a priority for the future.

Push notifications are currently fixed in terms of the content they deliver, with the only item customisable to the user being the time they are given. A future change would be allowing users to select some pre-defined options to make the notifications more engaging to the user.

Another potential future addition is supporting other platforms, such as Android. As shown during the literature review, Android and iOS share roughly the entire market for mobile platforms; therefore, porting the application to Android would further expand the application's reach.

Future work could also include the use of monetisation or advertising. Two of the three researched applications have in-app purchases to unlock more advanced features, and this is something which could be explored to achieve some revenue.

8.5 Personal Conclusion

Once the supervisor approved the basic idea of the project, there was an instant belief that the project would be completed with ease and on time. The project required a greater understanding of programming languages which I had not covered before. Although worrying to some, I believed this would help further develop some technical skills needed when developing applications. The application also allowed me to develop new skills when designing the application, something which I had rarely done before.

The minimal knowledge of the programming language used had little impact; however, as the application grew, it started to put more constraints on the application. If the project were to be completed again, far more time would be allocated to learning more about the programming language and the tools used throughout the Apple ecosystem.

Many issues arose throughout the implementation stage, which was solved using internet resources. These issues could be attributed to my lack of knowledge of the programming language, and as more components were developed, fewer issues were found due to poor understanding.

References

- Agarwal, R., & Umphress, D. (2008). Extreme programming for a single person team. *Proceedings of the 46th Annual Southeast Regional Conference: XX*, 82–87. <https://search.ebscohost.com/login.aspx?direct=true&db=edb&AN=83604201&site=eds-live>
- Alsaqqa, S., Sawalha, S., & Abdel-Nabi, H. (2020). Agile Software Development: Methodologies and Trends. *International Journal of Interactive Mobile Technologies*, 14(11), 246–270. <https://doi.org/10.3991/ijim.v14i11.13269>
- Apple. (n.d.-a). *Apple Developer Documentation*. Retrieved May 6, 2022, from <https://developer.apple.com/documentation/coredata>
- Apple. (n.d.-b). *Apple Developer Documentation*. Retrieved May 6, 2022, from <https://developer.apple.com/documentation/cloudkit>
- Apple. (n.d.-c). *Apple Developer Documentation*. Retrieved May 6, 2022, from https://developer.apple.com/documentation/security/keychain_services
- Apple. (n.d.-d). *Apple Developer Documentation*. Retrieved May 3, 2022, from <https://developer.apple.com/documentation/uikit>
- Apple. (n.d.-e). *Apple Developer Documentation*. Retrieved March 29, 2022, from <https://developer.apple.com/documentation/swiftui>
- Apple. (n.d.-f). *Apple Developer Documentation*. Retrieved May 3, 2022, from <https://developer.apple.com/documentation/usernotifications/unusernotificationcenter>
- Apple. (n.d.-g). *Apple Developer Documentation*. Retrieved May 3, 2022, from <https://developer.apple.com/documentation/healthkit>
- Apple. (n.d.-h). *Apple Developer Documentation*. Retrieved May 3, 2022, from [https://developer.apple.com/documentation/SwiftUI/View/sheet\(isPresented:onDismiss:content:\)](https://developer.apple.com/documentation/SwiftUI/View/sheet(isPresented:onDismiss:content:))
- Apple. (n.d.-i). *Apple Developer Documentation*. Retrieved May 3, 2022, from <https://developer.apple.com/documentation/swiftui/stepper>
- Apple. (n.d.-j). *Human Interface Guidelines - Design - Apple Developer*. Retrieved March 29, 2022, from <https://developer.apple.com/design/human-interface-guidelines/>
- Apple. (n.d.-k). *SF Symbols - Apple Developer*. Retrieved May 2, 2022, from <https://developer.apple.com/sf-symbols/>
- Apple. (n.d.-l). *Sheets - Views - iOS - Human Interface Guidelines - Apple Developer*. Retrieved May 3, 2022, from <https://developer.apple.com/design/human-interface-guidelines/ios/views/sheets/>
- Apple. (n.d.-m). *SwiftUI Overview - Xcode - Apple Developer*. Retrieved May 3, 2022, from <https://developer.apple.com/xcode/swiftui/>
- Apple. (n.d.-n). *TestFlight - Apple Developer*. Retrieved May 3, 2022, from <https://developer.apple.com/testflight/>
- Apple. (n.d.-o). *Xcode - Interface Builder - Apple Developer*. Retrieved May 3, 2022, from <https://developer.apple.com/xcode/interface-builder/>
- Barista. (n.d.). *Barista - Caffeine tracker*. Retrieved February 7, 2022, from <https://apps.apple.com/gb/app/barista-caffeine-tracker/id1570223740>
- Biørn-Hansen, A., Rieger, C., Grønli, T.-M., Majchrzak, T. A., & Ghinea, G. (2020). An empirical investigation of performance overhead in cross-platform mobile development

- frameworks. *Empirical Software Engineering*, 25(4), 2997–3040.
<https://doi.org/10.1007/s10664-020-09827-6>
- Codecademy.com. (n.d.). *Swift Courses & Tutorials | Codecademy*. Retrieved May 2, 2022, from <https://www.codecademy.com/catalog/language/swift>
- Daly, J. W., Holmén, J., & Fredholm, B. B. (1998). [Is caffeine addictive? The most widely used psychoactive substance in the world affects same parts of the brain as cocaine]. *Lakartidningen*, 95(51–52), 5878–5883.
<https://search.ebscohost.com/login.aspx?direct=true&db=cmedm&AN=9889511&site=eds-live>
- Delia, L., Galdamez, N., Corbalan, L., Pesado, P., & Thomas, P. (2017). Approaches to mobile application development: Comparative performance analysis. In *2017 Computing Conference, Computing Conference, 2017* (pp. 652–659). IEEE.
<https://doi.org/10.1109/SAI.2017.8252165>
- Dzhurov, Y., Krasteva, I., & Ilieva, S. (2009). *Personal Extreme Programming—An Agile Process for Autonomous Developers*.
- Eclipse. (n.d.). *Eclipse desktop & web IDEs | The Eclipse Foundation*. Retrieved May 6, 2022, from <https://www.eclipse.org/ide/>
- EFSA Panel on Dietetic Products, N. and A. (NDA). (2015). Scientific Opinion on the safety of caffeine. *EFSA Journal*, 13(5), 4102.
<https://doi.org/https://doi.org/10.2903/j.efsa.2015.4102>
- European Food Safety Authority. (2017). *EFSA explains risk assessment : caffeine*. European Food Safety Authority. <https://doi.org/doi/10.2805/618813>
- Fitt, E., Pell, D., & Cole, D. (2013). Assessing caffeine intake in the United Kingdom diet. *Food Chemistry*, 140(3), 421–426.
<https://doi.org/https://doi.org/10.1016/j.foodchem.2012.07.092>
- FlatIcon. (n.d.). *Free icons designed by Freepik*. Retrieved May 2, 2022, from <https://www.flaticon.com/authors/freepik>
- franklynw. (n.d.). *GitHub - franklynw/HalfASheet: A SwiftUI pseudo-modal partial screen sheet, with height customisation*. GitHub Repository. Retrieved May 3, 2022, from <https://github.com/franklynw/HalfASheet>
- Google. (n.d.). *Cloud Firestore | Firebase Documentation*. Retrieved May 6, 2022, from <https://developer.apple.com/documentation/cloudkit>
- Grønli, T., Hansen, J., Ghinea, G., & Younas, M. (2014). Mobile Application Platform Heterogeneity: Android vs Windows Phone vs iOS vs Firefox OS. *2014 IEEE 28th International Conference on Advanced Information Networking and Applications*, 635–641. <https://doi.org/10.1109/AINA.2014.78>
- Hamed, A. M. M., & Abushama, H. (2013). Popular agile approaches in software development: Review and analysis. In *2013 INTERNATIONAL CONFERENCE ON COMPUTING, ELECTRICAL AND ELECTRONIC ENGINEERING (ICCEEE), Computing, Electrical and Electronics Engineering (ICCEEE), 2013 International Conference on* (pp. 160–166). IEEE.
<https://doi.org/10.1109/ICCEEE.2013.6633925>
- Hatton, S. (2008). Choosing the Right Prioritisation Method. In *19th Australian Conference on Software Engineering (aswec 2008), Software Engineering, 2008. ASWEC 2008. 19th Australian Conference on* (pp. 517–526). IEEE.
<https://doi.org/10.1109/ASWEC.2008.4483241>

- HiCoffee. (n.d.). *HiCoffee - Caffeine Tracker*. Retrieved February 7, 2022, from <https://apps.apple.com/us/app/hicoffee-caffeine-tracker/id1507361706#?platform=iphone>
- JetBrains. (n.d.). *IntelliJ IDEA: The Capable & Ergonomic Java IDE by JetBrains*. Retrieved May 6, 2022, from <https://www.jetbrains.com/idea/>
- joe-scotto. (n.d.). *GitHub - joe-scotto/TextFieldStepper: A SwiftUI component to make inputting numbers easier than the native stepper component*. GitHub Repository. Retrieved May 3, 2022, from <https://github.com/joe-scotto/TextFieldStepper>
- Lamhaddab, K., Lachgar, M., & Elbaamrani, K. (2019). Porting Mobile Apps from iOS to Android: A Practical Experience. *Mobile Information Systems, 2019*, 4324871. <https://doi.org/10.1155/2019/4324871>
- Lindstrom, L., & Jeffries, R. (2004). Extreme Programming and Agile Software Development Methodologies. *Information Systems Management, 21*(3), 41–52. <https://doi.org/10.1201/1078/44432.21.3.20040601/82476.7>
- LinkedIn Learning. (n.d.). *LinkedIn Learning with Lynda: Online Training Courses for Creative, Technology, Business Skills*. Retrieved May 2, 2022, from <https://www.linkedin.com/learning/>
- mecid. (n.d.). *GitHub - mecid/SwiftUICharts: A simple line and bar charting library that supports accessibility written using SwiftUI*. GitHub Repository. Retrieved May 3, 2022, from <https://github.com/mecid/SwiftUICharts>
- Nehlig, A. (1999). Are we dependent upon coffee and caffeine? A review on human and animal data. *Neuroscience & Biobehavioral Reviews, 23*(4), 563–576. [https://doi.org/https://doi.org/10.1016/S0149-7634\(98\)00050-5](https://doi.org/https://doi.org/10.1016/S0149-7634(98)00050-5)
- Public Health England, & F. S. A. (2014, May 14). *Quantity of soft drinks consumed per day in the United Kingdom (UK) from 2008 to 2012, by age (in grams)*. Statista. <https://www.statista.com/statistics/437106/quantity-soft-drinks-consumed-in-the-united-kingdom/>
- React Native · Learn once, write anywhere*. (n.d.). Retrieved November 1, 2021, from <https://reactnative.dev>
- Shah, K., Sinha, H., & Mishra, P. (2019). Analysis of Cross-Platform Mobile App Development Tools. In *2019 IEEE 5th International Conference for Convergence in Technology (I2CT), Convergence in Technology (I2CT), 2019 IEEE 5th International Conference for* (pp. 1–7). IEEE. <https://doi.org/10.1109/I2CT45611.2019.9033872>
- Shaydulin, R., & Sybrandt, J. (2017). *To Agile, or not to Agile: A Comparison of Software Development Methodologies*. <https://search.ebscohost.com/login.aspx?direct=true&db=edsarx&AN=edsarx.1704.07469&site=eds-live>
- StatCounter. (2021a, June 29). *Market share held by mobile operating systems in the United Kingdom (UK) from December 2011 to December 2020*. Statista. <https://www.statista.com/statistics/262179/market-share-held-by-mobile-operating-systems-in-the-united-kingdom/>
- StatCounter. (2021b, June 29). *Mobile operating systems' market share worldwide from January 2012 to June 2021*. Statista. <https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/>
- Statista. (2021, May 21). *Forecast of the number of smartphone users in the United Kingdom from 2010 to 2025 (in millions)*. Statista.

<https://www.statista.com/forecasts/1143841/smartphone-users-in-the-united-kingdom>

Stine, M. M. (1), O'Connor, R. J. (1), Klein 3), L. C. (1, Yanko, B. R. (2), & Grunberg, N. E. (2). (n.d.). Evidence for a relationship between daily caffeine consumption and accuracy of time estimation. *Human Psychopharmacology*, 17(7), 361–367. <https://doi.org/10.1002/hup.423>

WaterMinder. (n.d.). *WaterMinder® - track your daily water intake, hydrate, feel better!* Retrieved February 7, 2022, from <https://waterminder.com>



**School of Computing
Project Initiation Document**

Ryan Bush

**MyCaffeine: Caffeine Tracker for iOS
Engineering Project**

1. Basic details

Student name:	Ryan Bush
Draft project title:	MyCaffeine: Caffeine Tracker for iOS
Course:	BSc Computer Science
Project supervisor:	Steven Ossont
Client organisation:	N/A
Client contact name:	N/A

2. Degree suitability

This project satisfies the criteria for the BSc Computer Science course by meeting several of the learning objects on modules taken throughout the degree. It builds on knowledge learned from programming modules in the first year, such as Introduction to Programming, where both Python and Java were learnt, and software engineering modules in the second year, where experience was gained from learning about the software development cycle and putting that knowledge into use by creating a full-scale application. It also involves developing an application for a real-life problem.

3. Outline of the project environment and problem to be solved

The problem my project will aim to solve is helping people track and understand their caffeine consumption. There are currently applications dedicated to tracking the consumption of caffeine, however these often lack depth and are hard to understand. Users would benefit from an application that makes it easy to track their usage.

The target audience for this application is people who currently, or wish to, monitor their caffeine intake either to aid in removing an addiction or to better understand their consumption. Caffeine addiction can cause adverse effects on people's everyday lives.

4. Project aim and objectives

The overall aim of this project is to produce an application that will allow users to keep track of their caffeine intake in a quick and easy manner. It will also aim to assist users monitor and reduce their caffeine intake over time.

Aims of the project:

- Allow users to manually input a quantity of caffeine (measured in milligrams [mg]).
- Allow users to select predefined drinks, such as those offered at coffee shops or brought from supermarkets.
- Allow users to track their caffeine levels throughout the day and see predictions on how their caffeine levels will be for the future, such as bedtime.

Stretch aims of the project:

- Allow users to synchronise their caffeine intake to Apple Health.
- Allow users to display widgets on their homescreen to easily view information.
- Allow the application to be opened on Apple iPad and Watch devices.
- Allow the user to synchronise heart related data, to see the effects caffeine has on heart rate.

5. Project deliverables

MyCaffeine will be delivered as an iOS application via the Apple App Store; therefore, it will be accessible by anyone who wishes to use the application from their own devices, providing they own or have access to an Apple mobile device. It will use online synchronisation features, such as cloud saving, but will not be dependent on it, so the user can still use the application without internet connection.

Alternatively, due to the need for approval to publish the application to the App Store, if approval is not received in time for the final demonstration, it will be accessible via the Apple TestFlight platform.

A final report will be written describing the processes behind the creation of the project, including the stages of development through the software life cycle. In addition to this, a section of the app will contain details on how to use the system, including a frequently asked questions section and an introduction page for setting up the application.

6. Project constraints

MyCaffeine will be built using the Swift programming language using XCode. The programming language is constantly updated as new technologies arise; however, it is unlikely for any update to happen during the development of this project.

The biggest constraint is time. The application must be completed by early May in order to demonstrate the application. A project plan will be completed to ensure progress does not fall behind schedule.

There are no further constraints for this project at this time; however, this may change in the future as the project progresses.

7. Project approach

The application will be developed using an agile methodology; this will be running the entire span of the project. The documentation period for this project, which primarily consists of the write up for the final report, will be conducted throughout the entire length of the project, to ensure enough time is given to the report.

The application will be developed using the Swift programming language. Some previous knowledge of this language is already known; however, more research will be required to enhance understanding of the language further. Furthermore, the application will make use of various SDKs available from Apple, such as HealthKit and SwiftUI.

8. Literature review plan

Using university library resources and other academic journal resources, I will research how the consumption of caffeine can affect someone's daily life, and how these effects can be reduced. Existing applications will also be researched to see what features are generally included and determine what kind of features would be necessary that the existing applications are currently lacking.

The research for this project will investigate existing systems for tracking levels of caffeine and how effective they are at engaging users and helping potential addiction. Research will also be conducted on how caffeine can affect someone.

The existing systems that will be researched are:

- WaterMinder [waterminder.com] - track your daily water intake, hydrate, feel better!
- HiCoffee
[<https://apps.apple.com/us/app/hicoffee-caffeine-tracker/id1507361706>] – Caffeine Tracker

The existing reports initially used for research are:

- Bae, E. J., Kim, E. B., Choi, B. R., Won, S. H., Kim, J. H., Kim, S. M., Yoo, H. J., Bae, S. M., & Lim, M. H. (2019). The Relationships between Addiction to Highly Caffeinated Drinks, Burnout, and Attention-Deficit/ Hyperactivity Disorder. *Soa--ch'ongsonyon chongsin uihak = Journal of child & adolescent psychiatry*, 30(4), 153–160. <https://doi.org/10.5765/jkacap.190015>

9. Facilities and resources

The development of this application will require an Apple Developer account, priced at £79 a year, which is already owned. This allows for the application to be published to the Apple App Store should the application reach all its aims. In addition to this, an Apple computer will be required to be able to develop the application, as well as Apple mobile devices, such as an iPhone, to effectively test the application. Xcode comes with various simulators, so the application can be effectively simulated on various devices, without the need to purchase several.

No other facilities or resources will be required.

10. Log of risks

The table below shows a range of risks that could occur during the project. The risks will be re-evaluated during each development stage of the project, and if any problems arise then the table will be updated. This will ensure that the project stays on target with deadlines, and any concerns regarding risks are addressed before they become more severe.

Description	Impact	Likelihood	Mitigation	First indicator
COVID-19 outbreak forces closure of the campus	Medium	Very High	Ensure that all tasks can be completed in a home working environment.	University informs that lab closure is likely
Lack of productivity during initial project stages	High	Low	Creating and sticking to milestones for each stage to ensure they are completed on target.	If a milestone is not met without any valid reason.
COVID-19 outbreak within the household	High	High	Maintain strict social distancing, personal hygiene and cleaning standards.	Myself or a member of the household developed COVID-19 symptoms.
Inadequate risk management	High	Low	Review the log of risks regularly to ensure it is updated with new risks.	New risks are found later in the development process.
Data loss	Very High	Medium	Regularly back up work to cloud and GitHub.	Hardware and/or software problems
Deadlines or other modules	High	Medium	Create and follow a project plan dedicating certain hours of the week to the project.	Focusing on other modules and falling behind on project plan

11. Project plan

The development of MyCaffeine will be broken down into several different sections, following the waterfall methodology. This will allow for an appropriate amount of time for each stage during the academic full year, as time is a major constraint. The different stages are outlined below. A Gantt chart follows the list which shows when each stage will take

place, and a rough deadline for each stage. This project plan will be updated throughout the year to allow for problems that could arise.

Literature Review

The literature review will investigate existing systems for caffeine tracking, which are similar to the system being developed.

Specification

This stage will involve the creation of system requirements that the project will aim to achieve. Requirements will be gathered by researching existing and previous systems and prioritising features in order of importance, as well as a questionnaire which will be sent to potential users.

Design

The design stage will include all the main systems within the project, including the main areas of the user interface that the end-user will interact with, as well as all data storage designs. Designs may be changed later in the development stage if required. A couple of designs will be made for each front-end feature, to analyse best which design will work from a user point of view.

Implementation

This stage will involve the main development of the platform. During this stage, each feature will be split up into minor milestones to meet, to ensure that the project can be completed in time. Each milestone will be tested once completed to ensure it works before starting the next milestone.

Testing

The testing stage will include closed tests of the software which will consist of a series of specified actions that users will be performing. Alpha and Beta software will be released to small numbers of users to allow for more intense testing. All problems that arise during testing will be documented with any issues found being corrected before the platform is completed. The testing phase will be conducted towards the end of the implementation phase to ensure time is allocated to fix any issues.

Documentation

The documentation stage of the project will be completed once the implementation has completed. This will cover how well the project has met the system and user requirements that were initially detailed. It will also cover how the implementation and testing stage has affected any changes that were made to the project.

The final documentation will be provided via a frequently asked question (FAQ) and tutorials section within the application.

Write Up

During the entire project, the final report will be created and maintained to document the entire development process. There will be small breaks from the write up to ensure time is spent on the implementation stage.

PROJECT PLAN		2021												2022																				
Task Name	Duration	November				December				January				February				March				April				May								
Week Commencing		17	24	31	7	14	21	28	5	12	19	26	2	9	16	23	30	6	13	20	27	6	13	20	27	3	10	17	24	1	8	15	22	29
Project Research	2 wks	[Gantt bar]																																
Literature Review	8-12 wks	[Gantt bar]																																
Specification	4-8 wks	[Gantt bar]																																
Design	3-5 wks	[Gantt bar]																																
Implementation	20-22 wks	[Gantt bar]																																
Testing	7-8 wks	[Gantt bar]																																
Documentation	6 wks	[Gantt bar]																																
Report Write Up	24-27 wks	[Gantt bar]																																
Project Demo	2 wks	[Gantt bar]																																

12. Legal, ethical, professional, social issues (mandatory)

An ethics form will be completed, which will be reviewed by my supervisor, to ensure that there are no ethical breaches in this project. The database used to store data will be protected and every precaution will be taken to ensure its security is maintained. In the event there is a compromise of data, there will be no link between the data stored and the individual it belongs to.

The application will also be developed in accordance with regulations from the United Kingdom, such as the Data Protection Act and the General Data Protection Regulation. This will involve making sure that sensitive data, such as personal information, are not mishandled and stored in a safe way. I will also avoid the use of copyright or patented products in the project, as I will be producing the work on my own and will only use products which are copyright and licence free.

Appendix B: Ethics Review



Certificate of Ethics Review

Project title: MyCaffeine: Caffeine Tracker for iOS

Name:	Ryan Bush	User ID:	904935	Application date:	18/11/2021 10:36:07	ER Number:	TETHIC-2021-102113
--------------	-----------	-----------------	--------	--------------------------	------------------------	-------------------	--------------------

You must download your referral certificate, print a copy and keep it as a record of this review.

The FEC representative(s) for the **School of Computing** is/are [Philip Scott, Matthew Dennis](#)

It is your responsibility to follow the University Code of Practice on Ethical Standards and any Department/School or professional guidelines in the conduct of your study including relevant guidelines regarding health and safety of researchers including the following:

- [University Policy](#)
- [Safety on Geological Fieldwork](#)

It is also your responsibility to follow University guidance on Data Protection Policy:

- [General guidance for all data protection issues](#)
- [University Data Protection Policy](#)

Which school/department do you belong to?: **School of Computing**

What is your primary role at the University?: **Undergraduate Student**

What is the name of the member of staff who is responsible for supervising your project?: **Steven Ossont**

Is the study likely to involve human subjects (observation) or participants?: Yes

Will peoples' involvement be limited to just responding to questionnaires or surveys, or providing structured feedback during software prototyping?: Yes

Will the study involve National Health Service patients or staff?: No

Do human participants/subjects take part in studies without their knowledge/consent at the time, or will deception of any sort be involved? (e.g. covert observation of people, especially if in a non-public place): No

Will you collect or analyse personally identifiable information about anyone or monitor their communications or on-line activities without their explicit consent?: No

Does the study involve participants who are unable to give informed consent or in are in a dependent position (e.g. children, people with learning disabilities, unconscious patients, Portsmouth University students)?: No

Are drugs, placebos or other substances (e.g. food substances, vitamins) to be administered to the study participants?: No

Will blood or tissue samples be obtained from participants?: No

Is pain or more than mild discomfort likely to result from the study?: No

Could the study induce psychological stress or anxiety in participants or third parties?: No

Will the study involve prolonged or repetitive testing?: No

Will financial inducements (other than reasonable expenses and compensation for time) be offered to participants?: No

Are there risks of significant damage to physical and/or ecological environmental features?: No

Are there risks of significant damage to features of historical or cultural heritage (e.g. impacts of study techniques, taking of samples)?: No

Does the project involve animals in any way?: No

Could the research outputs potentially be harmful to third parties?: No

Could your research/artefact be adapted and be misused?: No

Does your project or project deliverable have any security implications?: No

I confirm that I have considered the implications for data collection and use, taking into consideration legal requirements (UK GDPR, Data Protection Act 2018 etc)

I confirm that I have considered the impact of this work and and taken any reasonable action to mitigate potential misuse of the project outputs

I confirm that I will act ethically and honestly throughout this project

Supervisor Review

As supervisor, I will ensure that this work will be conducted in an ethical manner in line with the University Ethics Policy.

Supervisor's signature:

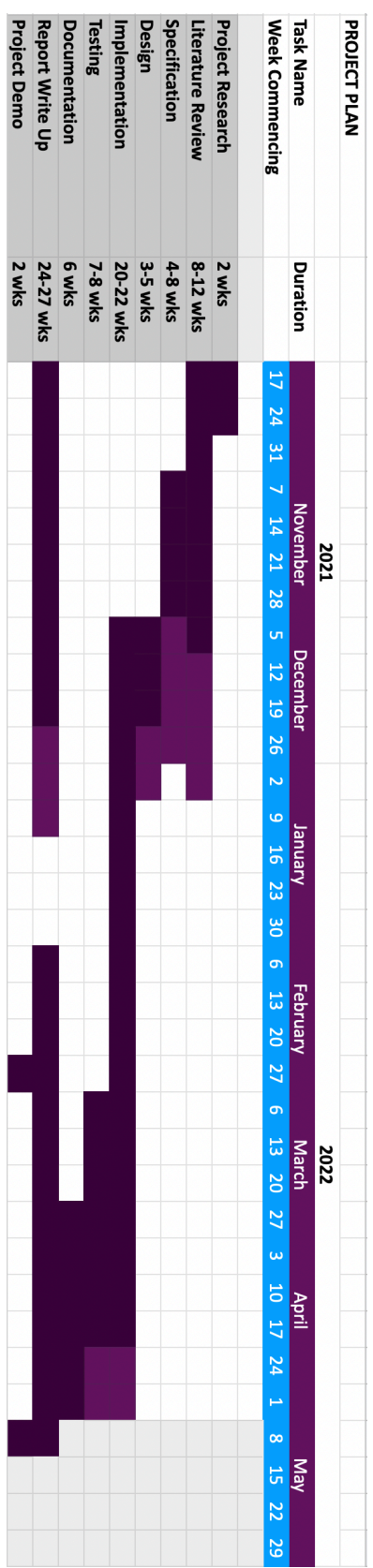
Date:

Faculty Ethics Committee Review

Faculty Ethics Committee Member's signature:

Date:

Appendix C: GANTT Chart



Appendix D: Questionnaire

Caffeine Consumption Questionnaire

The aim of this questionnaire is to gather user insights on how caffeine is consumed and tracked, and what key features of mobile applications are most important.

This questionnaire should take no more than 10 minutes to complete.

Your participation is completely voluntary and you will be able to withdraw at any time without justification.

All survey responses will be strictly confidential and the data will be stored on the University of Portsmouth secure network. If you have any issues or concerns with this survey or its procedures, you may contact me at up904935@myport.ac.uk.

*Required

1. Do you wish to continue with this questionnaire? *

Mark only one oval.

- Yes
 No *Skip to section 4 (Submit)*

Personal Information

2. What is your age bracket? *

Mark only one oval.

- 18-25
 26-35
 36-45
 46-55
 55+

3. What is your gender? *

Mark only one oval.

- Male
- Female
- Prefer not to say
- Other: _____

4. What is your employment status? *

Mark only one oval.

- Full-time Employed
- Part-time Employed
- Student
- Unemployed
- Other: _____

Caffeine Consumption

5. Do you consume caffeinated drinks? *

Mark only one oval.

- Yes
- No

6. What type of caffeinated beverages do you usually consume?

Tick all that apply.

- Coffee & Speciality Coffees (Hot, Cold, Iced)
- Tea & Speciality Teas (Hot, Cold, Flavoured)
- Soft Drinks
- Energy Drinks
- Other: _____

7. Roughly how many caffeinated drinks do you consume a day? *

Mark only one oval.

	0	1	2	3	4	5	6	7	8	9	10	
0 Drinks	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	10 Drinks or more

8. For what purpose would you consume caffeinated drinks?

Tick all that apply.

- To stay up late
- To feel more awake
- To help focus
- To help productivity
- To be more alert
- To help physical performance
- No specific purpose
- Other: _____

9. Are you aware of how much caffeine is in the drinks you consume? *

Mark only one oval.

- Yes
- No

10. Do you currently track your caffeine usage? *

Mark only one oval.

- Yes
- No

11. How do you track your caffeine usage?

Tick all that apply.

- Manually
- Mobile Application
- Desktop Application
- Other: _____

12. Would you say you are currently addicted to caffeinated drinks? *

Mark only one oval.

- Yes
- No
- Prefer not to say

13. Do you feel it would be hard to cut down or stop drinking caffeinated drinks completely? *

Mark only one oval.

- Yes
- No

14. Would having more information about your caffeine intake be likely to change the amount of caffeine you consume?

Mark only one oval.

- Yes
- No
- Maybe

15. Would you find a system that make it easy to track your caffeine usage useful? *

Mark only one oval.

- Yes
- No
- Maybe

Submit

Thank you for your time to complete this questionnaire. To save the answer please click the Submit button below

This content is neither created nor endorsed by Google.

Google Forms

Appendix E: Questionnaire Results

Caffeine Consumption Questionnaire

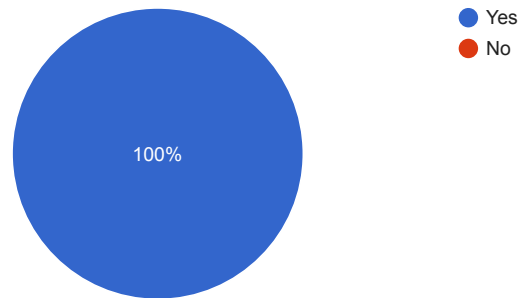
11 responses

[Publish analytics](#)

Do you wish to continue with this questionnaire?

 [Copy](#)

11 responses

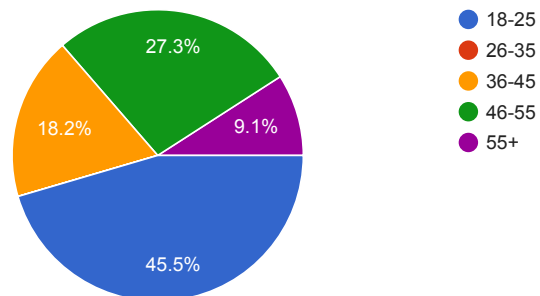


Personal Information

What is your age bracket?

 [Copy](#)

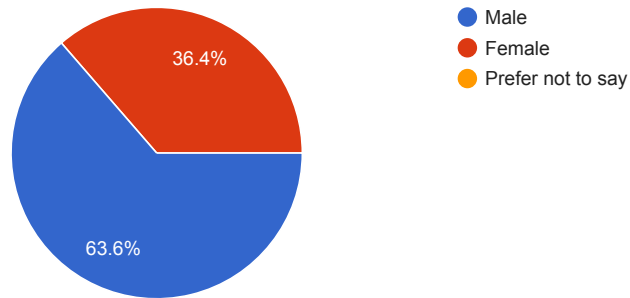
11 responses



What is your gender?

 Copy

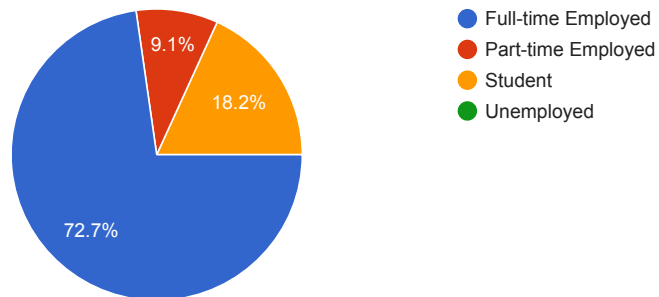
11 responses



What is your employment status?

 Copy

11 responses

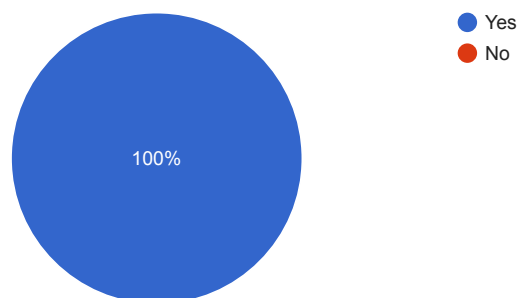


Caffeine Consumption

Do you consume caffeinated drinks?

 Copy

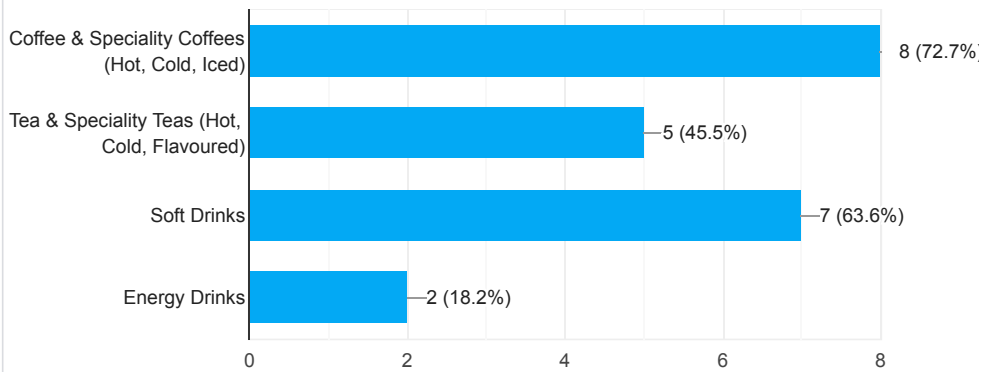
11 responses



What type of caffeinated beverages do you usually consume?



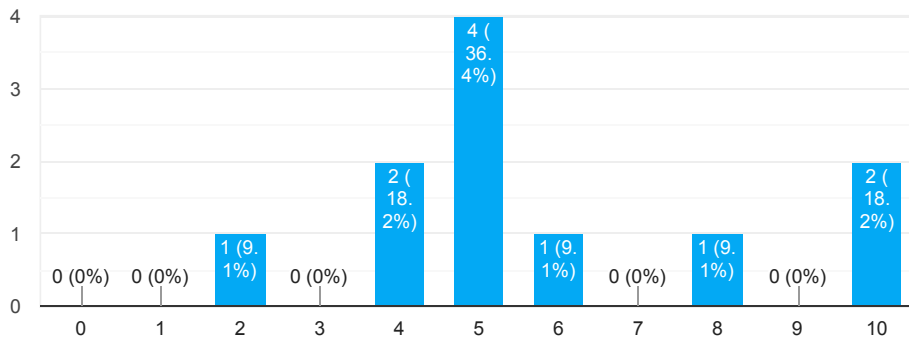
11 responses



Roughly how many caffeinated drinks do you consume a day?



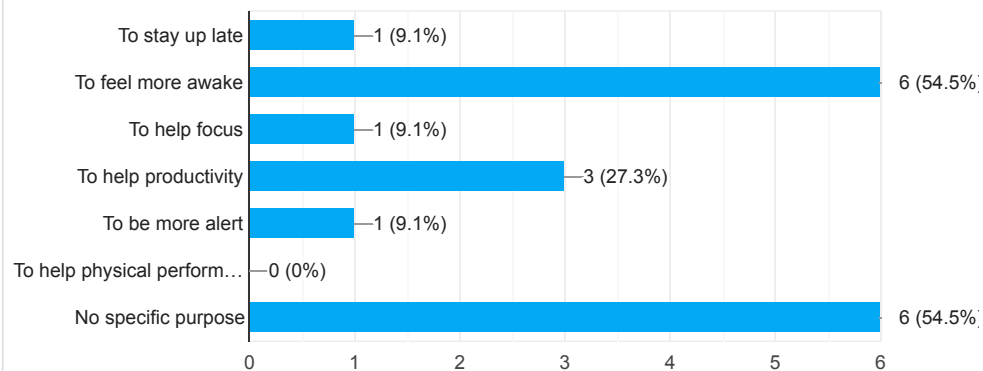
11 responses



For what purpose would you consume caffeinated drinks?



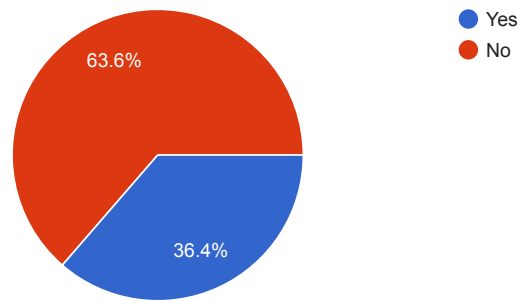
11 responses



Are you aware of how much caffeine is in the drinks you consume?

 Copy

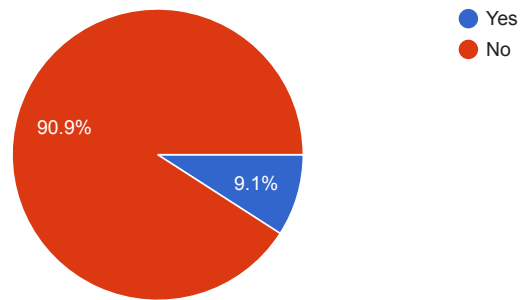
11 responses



Do you currently track your caffeine usage?

 Copy

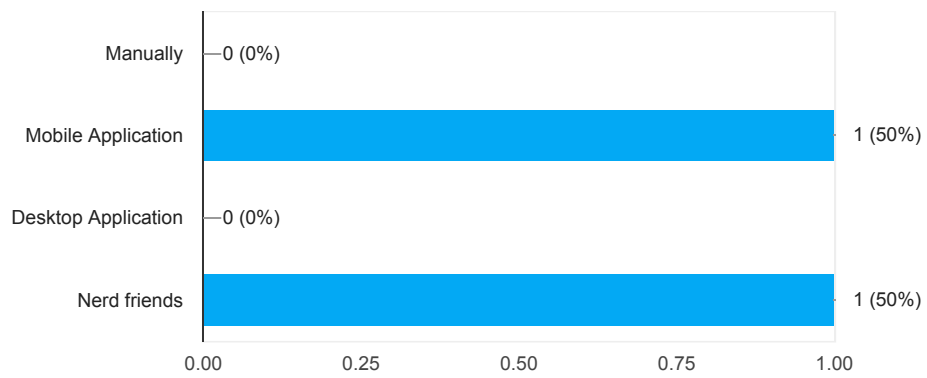
11 responses



How do you track your caffeine usage?

 Copy

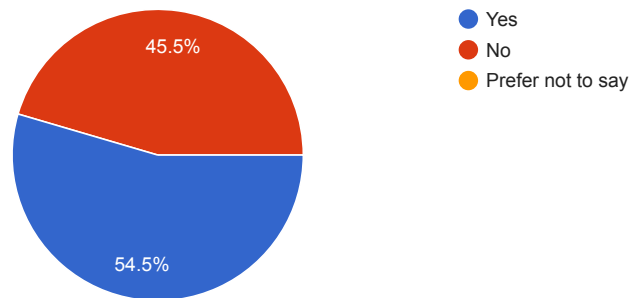
2 responses



Would you say you are currently addicted to caffeinated drinks?

 [Copy](#)

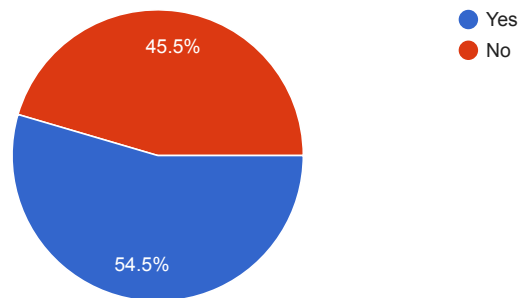
11 responses



Do you feel it would be hard to cut down or stop drinking caffeinated drinks completely?

 [Copy](#)

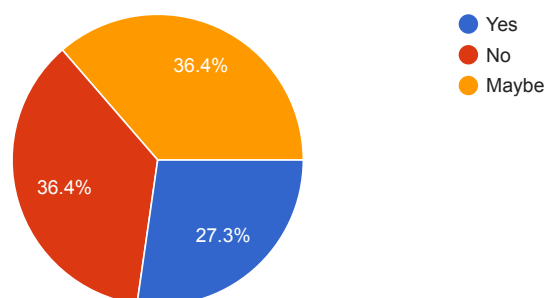
11 responses



Would having more information about your caffeine intake be likely to change the amount of caffeine you consume?

 [Copy](#)

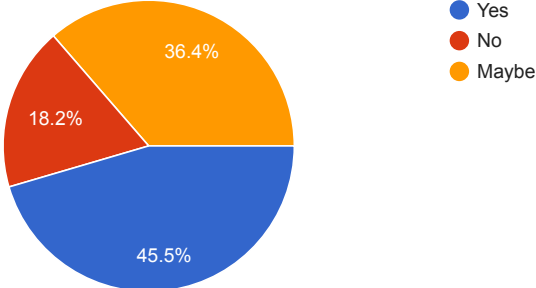
11 responses



Would you find a system that make it easy to track your caffeine usage useful?

 Copy

11 responses



Submit

This content is neither created nor endorsed by Google. [Report Abuse](#) - [Terms of Service](#) - [Privacy Policy](#)

Google Forms