

CM10313

Tempo – A Personal Informatics Application

Group Member	Username	Course
H. Kim	hk621	MComp (hons) Computer Science and Artificial Intelligence (yr. 1)
C. Davies	crd37	BSc (hons) Computer Science (yr. 1)
G. Simonetti	gs929	BSc (hons) Computer Science and Artificial Intelligence (yr. 1)
W. Jones	wij21	BSc (hons) Computer Science and Artificial Intelligence (yr. 1)
A. Barrell	awb50	BSc (hons) Computer Science with Year long work placement (yr. 1)
J. Rabjohns	jor37	BSc(hons) Computer Science (yr. 1)
S. Sujit	ss3601	BSc (hons) Computer Science and Artificial Intelligence (yr. 1)
I. Mocanu	irm37	MComp (hons) Computer Science (yr. 1)

Tempo – A Personal Informatics Application

Hail Kim, Connor Davies, Giovanni Simonetti, William Jones, Archey Barrell, Jay Rabjohns, Shikha Sujit, Ioana Mocanu

Abstract

Personal Informatics is a field of research that focuses on helping people to collect personally relevant information about themselves. The goal of personal informatics systems is to help people collect and reflect data about their own personal behaviours to improve the quality of their lives (Li, Dey and Forlizzi, 2010). University students are expected to regularly study and retain information related to their chosen field, as well as dealing with other aspects of their lives. This can often prove challenge for many students, therefore many of them choose to use applications which help them have a productive day, by helping them manage their time efficiently. According to Kaser (n.d), students intending to monitor their actions over a period of time need to clearly identify the current problem or situation; choose a desired behaviour; set learning and performance goals; and identify consequences for meeting or failing to meet an outcome. The Tempo application was designed to help university students manage their study and exercise habits better. The app provides functionalities which allow the user to use a timer to monitor the length of a session, create, modify, and analyse personal goals and receive motivational messages, which inspire the user to finish a chosen goal. The data collected is stored in a database which can be reviewed and deleted later by the user.

Table of Contents

Tempo – A Personal Informatics Application..... 2

Abstract..... 2

Introduction 4

Agile Software Process and Management 6

Software Requirements Specification..... 9

Design..... 14

 Login and Register UML 14

 Main Form UML 15

 Timer UML..... 16

 Login and Register Sequence diagram 17

 Main Form Sequence diagram 18

Software Testing 19

 Unit & Integration Testing 19

 Acceptance Tests 20

Reflection and Conclusion..... 22

 Reflection 22

 Conclusion..... 25

References 26

Appendices..... 27

 Appendix A (Group Contribution Form)..... 27

 Appendix B (Acceptance Testing) 28

 Appendix C (Unit & Integration Testing)..... 47

 Appendix D (Initial GUI) 58

 Appendix E (Reworked GUI)..... 58

 Appendix F (Survey results) 59

 Appendix G (Password Hashes) 61

Introduction

For this project, we were tasked with creating a Personal Informatics (PI) system with current university students as the target audience. This brought to question what goal the PI system would have, that would specifically have a positive impact on current university students.

Students at any level of education are expected to regularly study and retain information throughout their student lives. University students will also be expected to maintain their studies alongside all other aspects of their daily lives. In the new setting they are thrown into, we felt that maintaining mental and physical wellbeing as well as keeping up with their studies can often be difficult. As a result, we decided to focus on the development of a PI system focused on the educational aspect of being a student.

Research (Atma, Handarini, and Atmoko, 2019) has shown that both self-motivation and time-management methods significantly reduce academic procrastination by up to 40%. This suggested a need for the PI system to have both a time-management and motivational aspect. There are many variations of time-management available to students including calendars, diaries, to-do lists. Despite this, not all students may find it easy to regularly update and keep up with using the previously mentioned methods of time management and therefore avoid them. On the other hand, timers are readily accessible on every phone or computer and can be used as a way of managing time. Pre-existing research on the Pomodoro technique (Cirillo, 2006) has confidently shown how effective timers can be for improving productivity. The issue with using timers is that there is no visual representation of data history; whereas days on a calendar or to-do lists can be viewed with ease after completion, the readily accessible timers do not store data conveniently for viewing. Recording previous instances of study is an integral part of self-monitoring progress which has been shown to improve academic performance (Kaser). As a result, the need to store and view past instances of timer sessions was apparent and would need to be incorporated into our PI system. (Kaser) also explains that the setting of goals as well as consequences of failure are important to self-monitoring progress. However, like keeping up with other forms of time-management, consistently setting and working to surpass goals can be difficult. To counter-act this, (Karanam, Filko, Kaser, Alotaibi, Makhsoom, and Stephen Void, 2014) suggests that the gamification of goals would improve the commitment towards goals.

Though education is important, there were also many other aspects of the student life we wanted to support. Initially, we considered sleep and exercise as potential areas for further PI development. Though sleep has a definite effect on studying (Walker, 2008), it would be more difficult to measure as we would rely on user input regarding quality of sleep. On the other hand, exercise would be easier to record, especially with the use of the timer, and has also shown to have an impact on mental health¹ which is very important in keeping healthy habits including studying (Roig, Nordbrandt, Geertsen and Nielsen, 2013).

To ensure the best choice of our PI system's secondary focus, we collected and analysed data from a questionnaire completed by 60 current university students. When asked if they efficiently managed their time on a daily basis, only a small 28.3% responded positively. Furthermore, a large 60% felt that a self-tracking app would help them feel more in control in aspects of their life. While 26.7% felt neutral, only a small 13.3% felt a tracking app would not help them feel more in control. These responses suggested that a PI system based on time-management would be beneficial to most users. We went on to ask which areas users would want the app to implement out of "Education", "Sleep" and "Exercise". Users could select more than one option. 49 users wanted "Education", 35 users wanted "Sleep" and 32 users wanted "Exercise". As the total number of votes (116) surpassed the number of voters (60) which suggested that users wanted more than one implementation. "Education" was the most desired area of implementation which supported our initial plan to focus on the studious aspect of being a student. "Sleep" and "Exercise" were relatively similar though a few more users supported the former. However, when the students were asked if they felt they were getting enough sleep, though 35% said maybe, only 23.3% said no. When asked how many hours of sleep students got

¹ <https://web-b-ebsscohost-com.ezproxy1.bath.ac.uk/ehost/detail/detail?vid=0&sid=2e262be0-3498-4ba9-909e-4bdbf38ac4cb%40pdc-v-sessmgr01&bdata=JnNpdGU9ZWwhvc3QtbGl2ZQ%3d%3d#db=s3h&AN=147569057>

on average, 78.3% said they slept more than 6 hours. For the age range of university students (Singh, 2021), this suggested that a large majority of students were getting enough sleep. Alongside the difficulty of measuring sleep accurately, the survey suggested that sleep was not an issue for most students. As such, we decided to focus on exercise as the implementation secondary to education. When students were asked about what main functions they would want implemented in an exercise app, 80% of responses said, “activity recording”. Consequently, we decided that it would be efficient and practical to build the app around recording and storing physical activity sessions as well as study sessions.

Though the main features and areas we wanted focus our PI system on were clearer, there were many pre-existing concerns with PI systems, their functionality, and their effectivity. Though studies demonstrate the efficacy of PI systems in developing healthy habits (article from spec pdf), several studies (Potapov, Lee, Vasalou, and Marshall, 2019) (Gulotta, Forlizzi, Yang, and Newman, 2016) also indicate concerns in PI system abandonment. In a study (Potapov, Lee, Vasalou, and Marshall, 2019) focusing on the way youth reacted with PI systems, it was shown that youth felt the tracking of their data was necessary when the activities being tracked worked towards the image of who they want to be. On the other hand, it was also shown that if activities youth deemed useful were not tracked, it could lead to the rejection of PI systems. These findings suggested that effective PI systems for youth would be able to set goals whilst retaining a sense of individuality. Research done by (Gulotta, Forlizzi, Yang, and Newman, 2016) also supported this idea by finding that abandonment was an issue when people’s goals or identity did not align with the PI system. (Gulotta, Forlizzi, Yang, and Newman, 2016) also went on to find that burdensome routine use requirements or lack of social support connections could also lead to abandonment.

For our PI system to meet the concerns surround PI systems, we would need to ensure that the individuality of users’ goals is kept while still allowing each user to review their own unique progress. Incorporating some form of gamification could also improve the motivation of users to achieve their goals. Furthermore, we would need to ensure the PI system was simple and easy to use with little effort required when recording and reviewing user data. Finally, exploring and implementing users with social support would help build a community to further motivate users.

Following the research on PI systems, we adjusted and finalised our PI system to be an organisation tool for students to record, compare and analyse time data to track goal progress and see potential trends. Students would be able to use an inbuilt timer to create study/exercise sessions with specific active and rest times. Making use of this pomodoro-like intermittent activity will help students effectively use their time whilst the option to adjust the session times will prevent the limitations of a pre-set time. Students would also be able to view the metadata of the study and exercise sessions in the form of graphs, to compare and discover if there is a productivity trend, or lack of, between activities or on specific days. Furthermore, the app will allow students to create, track and complete time-related goals. By allowing the creation of goals with unique names and descriptions, students can set personal goals, therefore allowing them to maintain individuality and discourage PI system abandonment. Finally, to regularly encourage students in achieving their goals and maintaining activity, motivational messages are presented to the user throughout the app. For studying and exercise sessions, specific motivational messages targeting the respective activity should be presented whereas general motivational messages would be displayed on the home screen.

Agile Software Process and Management

When we talk about the Agile Software Process and Management the first thing that comes up in our mind is what does agile mean? Being agile is being able to create and respond to any change that happens. It is thinking about what is going on in the environment in the present condition, identifying the uncertainties and then learning to adapt it and moving forward. In our project, we have used the Agile Process to carry out all our tasks in an organised manner to be able to handle any changes that could happen during the project.

Agile Software Development is a framework of different methods like Scrum, Extreme Programming, Test-Driven Development (TDD) or Feature-Driven Development (FDD). We have used the Scrum and Test-Driven development approach to make our personal informatics system. Agile methodologies take an iterative approach to the development of the software. Agile projects consist of smaller cycles also known as sprints. Within a sprint, there are smaller scrum meetings. Scrum works well for long-term, complex projects that require stakeholder feedback, which may greatly affect project requirements. So, when the exact amount of work cannot be estimated, and the release date is not fixed, Scrum may be the best choice.

We created a project backlog (Fig.2) of all the requirements that were required to complete our PI System on an online issue tracker called Trello. Using Trello, we were able to maintain regular changes and adapt to changing priorities. As Trello is focused on doing small pieces of work as they come up, we were able to divide the entire project into three sprints on Trello whilst keeping track of minute changes within each sprint. Each sprint lasted for two weeks and had regular scrum meetings which were kept on alternate days. All our meetings were held on Microsoft teams and all the ideas given by each student were noted in the OneNote app to keep track of the progress made in each meeting. At the start of each sprint, we created a sprint backlog page in Trello which consisted of three boards: a to-do list, a doing list, and a done list. We took a feature that we needed to implement, the requirements needed to implement that feature (from the project backlog) and added it to our to-do list. A few tasks were then assigned to each team member according to individual preferences and interest. At the end of every sprint, the team partook in a sprint review where we displayed the progress made in the prior sprint to a supervisor, using the completed Trello tasks as a structure for the meeting, before discussing how we would like to improve as a team to optimise our use of the agile process in the next sprint.

Before starting our first sprint, we had team meetings regarding what our PI system was going to be and the features that were required to implement it. To gather consumer information to tailor our app requirements, we created an online survey form and sent it to students, asking for their opinions on what type of PI systems and features they would prefer to use in their daily life. We also discussed the different platforms that could be used for the app. In the end we chose to use Java for the back end, Java's Graphical User Interface (GUI) for front-end development and SQLite as the backend database for storing and viewing all the data from the system. A specific requirement we wanted to fulfil was for the data will be GDPR compliant as it would give the app commercial viability. After reviewing the online survey forms as a team, we decided that we would go with the theme of an education and exercise PI System.

Our main objective for our first sprint (Fig.3) was to create a register and login system. Different tasks were assigned among our team members and an attempt was made to distribute the workload as evenly as possible. The tasks included creating software for front-end and back-end development, creating the UML diagram for register and login, basic UI design and Use Case diagram, creating a third-normal-form entity structure for the database, writing the introduction and the requirements table for our final report.

During the scrum meetings, (Fig.1) we discussed the progress of the work and any hurdles faced during the implementation process. We also discussed adding an extra feature, which was implementing the goals page to improve the user engagement whilst using the application. We discussed the requirements needed to create this new feature and added these requirements to our project backlog to implement it in our next sprint. Several J-Unit tests were run for that feature to record and rectify all the errors. During our Sprint review, we went through all the tasks mentioned above and we were able to complete most of them. We then moved on to the planning and execution of our next sprint where any unfinished tasks were read.

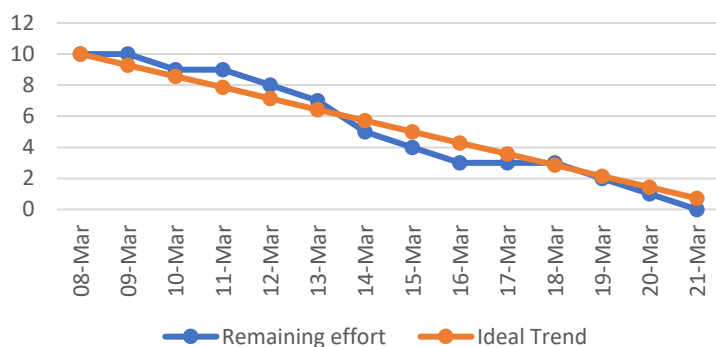
In our second sprint (Fig.4) we decided to implement the timer, motivational messages, and goals components; create the database table for storing personal goals; create the home page; create the editing user account details form; design the UML for Timer and Sequence diagrams; implement password encryption; setup the database handler; set up a basic exercise timer and all components required to make the timer, and create a UI design for how each page would look like. All these tasks were placed in the sprint backlog. Like in the first sprint, we had regular scrum meetings to discuss the progress we had achieved over regular intervals. By the end of sprint two, we had completed the basic study timer page and the back end for the motivational message. The sequence diagrams for login and registration and Timer UML were completed. Some tasks could not be completed, so we decided to move them to the next sprint backlog. During one of our scrums meetings, we came up with a name for our PI system, which was “Tempo”. Several J-Unit tests for the timer were run to test the working of the system.

In our third sprint (Fig.5) we decided that we would organise all the features which we had created to provide a complete Personal informatics system. All the tasks were related to improving the existing system features, implementing any incomplete features, writing the main report, and creating the final video. At the end of the sprint, we were able to follow the same review procedure as previous sprints and reflected on our use of SCRUM.

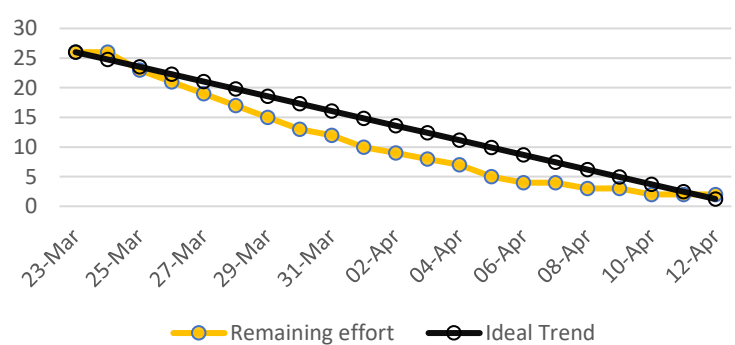
From all these sprint planning and scrum meetings, we were able to create a complete PI system that could be used by students to effectively manage their time, as we set out to achieve. Teamwork was an essential component of our success, and through utilising and incrementally improving them, we have accomplished our goal.

Below are the burndown charts of our three sprints. The x-axis represents the days within a sprint and y-axis represents the estimated effort-hours taken. The ideal trend line intersects the horizontal axis close to the end of the sprint duration, we can conclude that we have completed all the tasks “on time”. The remaining effort line is our work during each sprint.

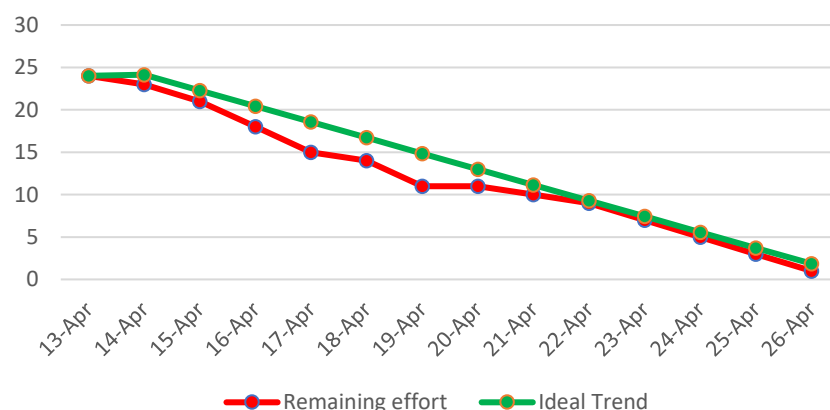
Sprint 1 Burndown Chart



Sprint 2 Burndown Chart



Sprint 3 Burndown Chart



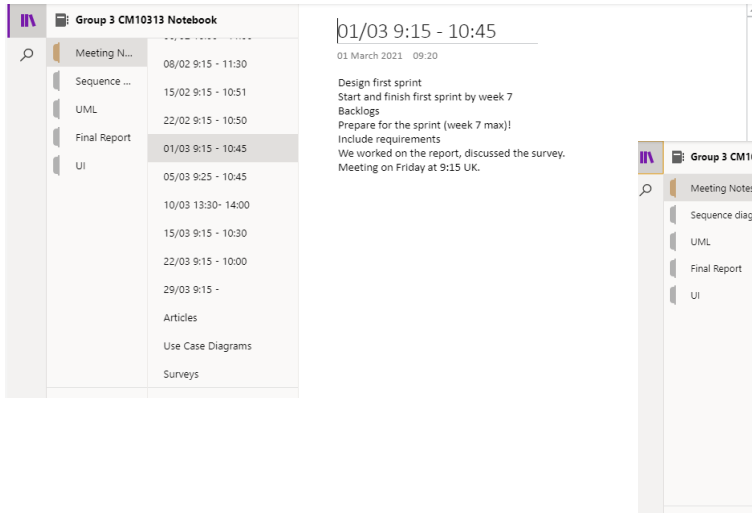


Fig 1: SCRUM MEETINGS

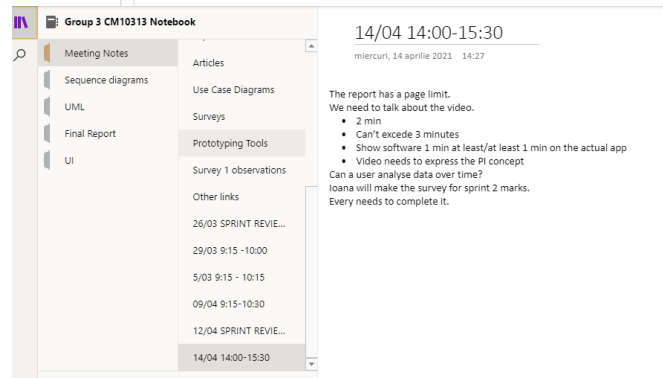


Fig 2: Project Backlog

Fig 3: Sprint 1

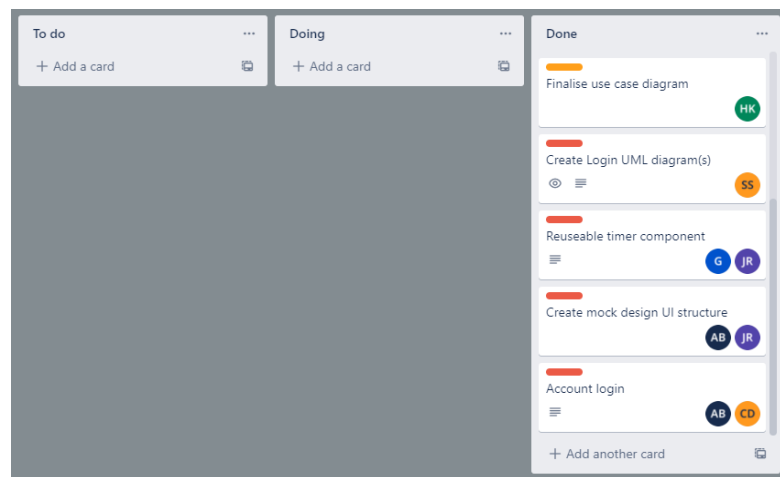
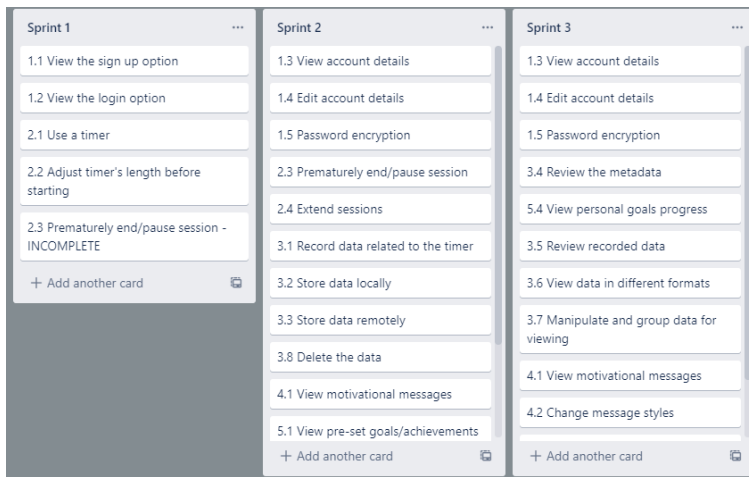
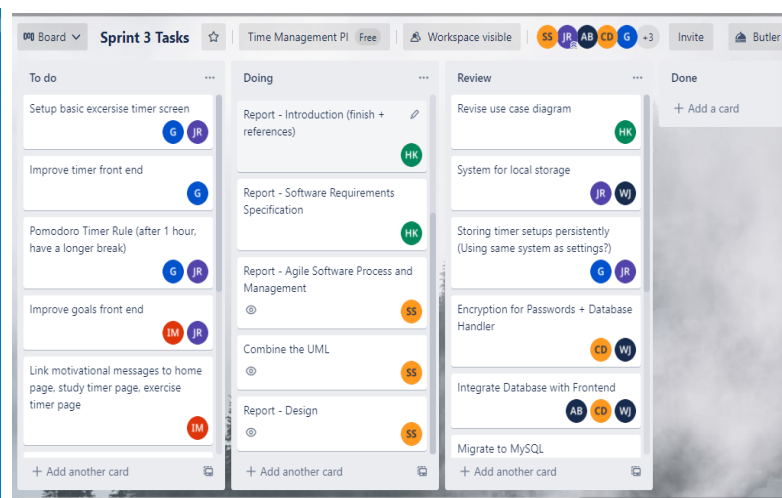
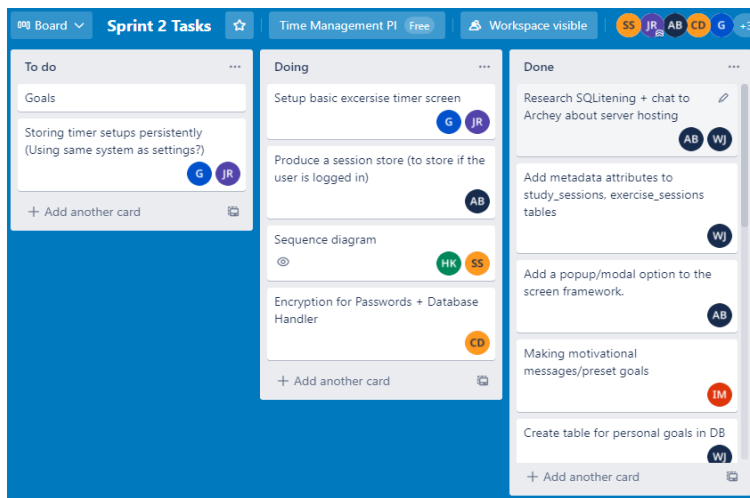


Fig 4: Sprint 2

Fig 5: Sprint 3



Software Requirements Specification

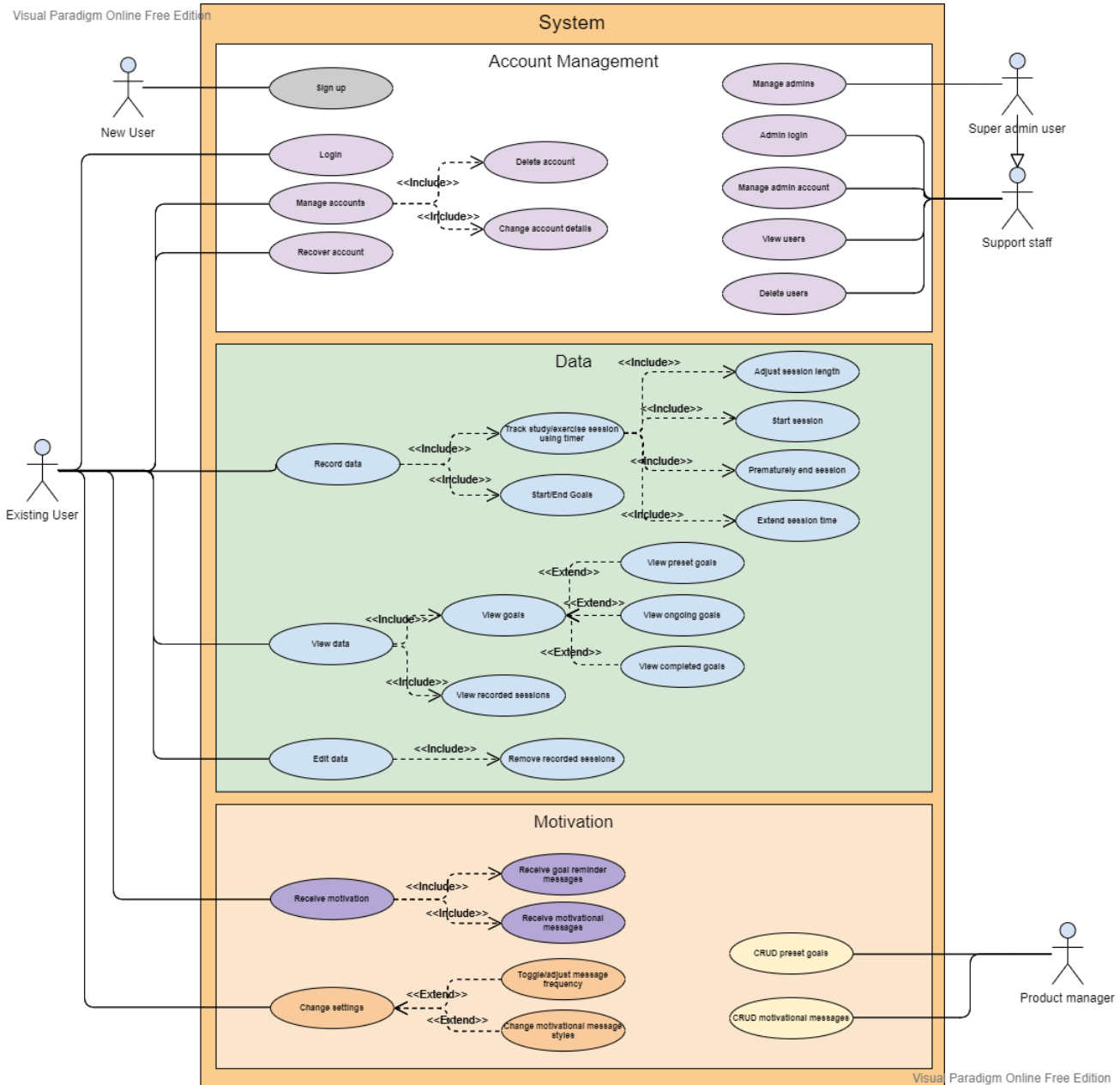
A software requirements specification represents the foundation of each project as it provides a structure for the development of the project. It helps members keep aware of the project as a whole and makes it easier to make and complete targets. At the beginning of the project before our first sprint, we were able to meet and collect ideas around a PI system involving education, sleep, exercise or mood. Through research on existing PI systems and issues surround PI systems, alongside further discussions leading up to the first sprint, we were able to establish a more precise idea of app. We decided that we wanted a PI system using user-input data with goals and motivational messages to encourage the users. However, we were still unable to choose a specific area of interest to centre our PI system around. During the first few days of the first sprint, we were able to send a survey to 60 current university students and use the data to confirm on “education” and “exercise” as the main areas of focus. Following this, we were able to clarify our ideas and updated the final requirements for the project.

Moving on, we developed requirements for the basic functionalities of the app including a sign-up/login page for the user, a timer, and a database that would be used to store the data collected. Afterwards, we provided the user with the option to view their account details, to extend their study session, to view motivational messages and set personal goals. We also expanded the functionalities of the database to be able to view and modify data.

We wanted our PI system to have a positive impact on current university students. We were aware that being a university student was challenging. Students are expected to organise their time effectively, to keep up with their studies, maintain a healthy mental and physical wellbeing, participate in extracurriculars, and deal with other unexpected events of their lives. As a result, we decided to create a system that could help students stay motivated, manage their schedules efficiently, to reduce procrastination. We decided to focus on the education habits that the students have, since education is the most important aspect of being a student, and the exercise habits, because exercise helps an individual maintain a healthy mental and physical wellbeing, and it is easy for any user to record their exercise sessions. We wanted our system to be accessible to everyone, so we created it without having certain age or ethnicity categories in our minds. The app can be used by any student, regardless of course or university.

As part of our report, we have also included a use case diagram. “The purpose of use case diagram is to capture the dynamic aspect of a system.” (Waykar, 2015). Use case diagrams are used to specify the events of a system and their flows, gather system requirements, show how actors interact with the system and identify factors which influence the system.

The use case diagram below, presented as a sequence of simple steps, shows how the potential users can perform tasks on our app. It reflects the app from the users and staff’s point of view. A new user can only sign up. Existing users can login and manage their accounts (as part of Account Management). They can begin a new study/exercise session, extend it, or prematurely end it. They can also record data related to the length of their study/exercise sessions and remove it. They can view their goals (pre-set/ongoing/completed), receive goal reminders and motivational messages. The super admin user is able to manage the admins. The support staff can login as an admin, manage their account and view or delete users. The product manager is in charge of introducing in the system, pre-set goals and motivational messages.



FUNCTIONAL REQUIREMENTS

1. REQUIREMENTS RELATED TO USERS ACCESSING THEIR ACCOUNTS

1.1	Requirement Name: <i>View the signup option.</i>	Priority: HIGH
	Description: The system should include a signup option, where unregistered users can create an account.	Dependencies: N/A
		Source: N/A
1.2	Requirement Name: <i>View the login option.</i>	Priority: HIGH
	Description: The system should include a login option, where registered users can access their accounts through this option.	Dependencies: N/A
		Source: N/A
1.3	Requirement Name: <i>View account details.</i>	Priority: LOW
	Description: The user should be able to view their account details.	Dependencies: 1.2
		Source: N/A (data stored?)
1.4	Requirement Name: <i>Edit account details.</i>	Priority: LOW
		Dependencies: 1.2, 1.3

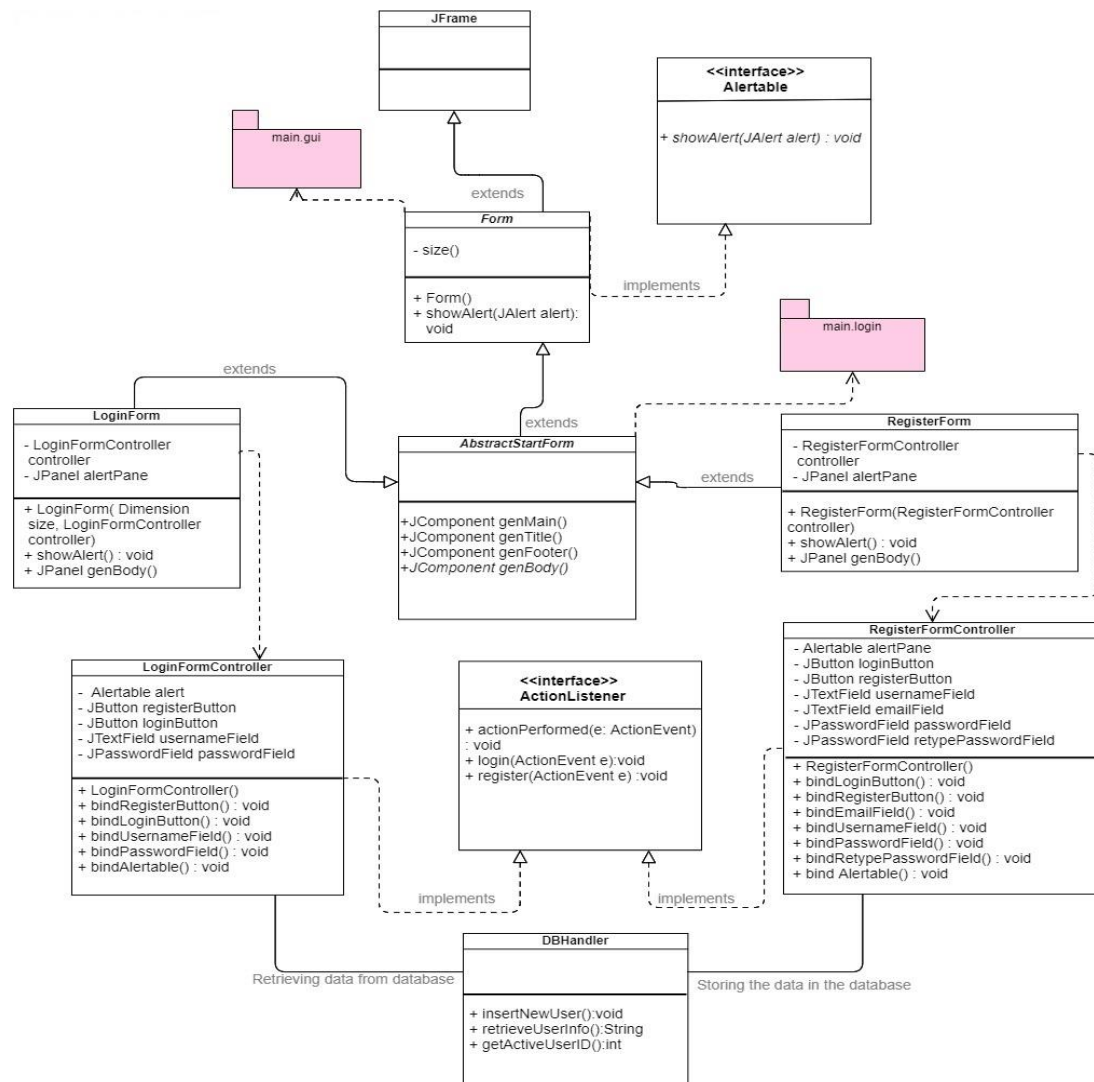
	Description: The user should be able to edit their account details.	Source: N/A
1.5	Requirement Name: <i>Password Encryption</i>	Priority: MEDIUM
	Description: A user's password should be encrypted by using some form of hashing algorithm	Dependencies
		Source: 3.9
2. REQUIREMENTS RELATED TO TIMING		
2.1	Requirement Name: <i>Use a timer.</i>	Priority: HIGH
	Description: The system should include a timer, where users can measure their study/exercise sessions. The base time is 25 minutes work with 5 minutes break.	Dependencies: N/A
		Source: Cirillo, F., 2006. <i>The Pomodoro Technique (The Pomodoro)</i> .
2.2	Requirement Name: <i>Adjust timer's length before starting</i>	Priority: MEDIUM
	Description: The user should be able to adjust the length of the timer before starting the study/exercise session.	Dependencies: 2.1
		Source: Personal Informatics Survey, February 2021
2.3	Requirement Name: <i>Prematurely end/ pause session</i>	Priority: MEDIUM
	Description: The system should include an option which allows the user to prematurely end/pause their session.	Dependencies: 2.1
		Source: N/A
2.4	Requirement Name: <i>Extend sessions</i>	Priority: MEDIUM
	Description: The user should be able to extend their current session.	Dependencies: 2.1
		Source: Personal Informatics Survey, February 2021
3. REQUIREMENTS RELATED TO STORING DATA		
3.1	Requirement Name: <i>Record data and meta data related to the timer</i>	Priority: LOW
	Description: The system should record the timer's data.	Dependencies: 2.1
		Source: N/A
3.2	Requirement Name: <i>Store data locally</i>	Priority: HIGH
	Description: The system should store the data locally.	Dependencies: 3.3
		Source: N/A
3.3	Requirement Name: <i>Store data remotely</i>	Priority: HIGH
	Description: The system should store the data remotely.	Dependencies: 3.2
		Source: N/A
3.4	Requirement Name: <i>Review the metadata</i>	Priority: LOW
	Description: Admins and users can review time of data input.	Dependencies: 3.1, 3.2, 3.3, 3.5, 3.6
		Source: N/A
3.5	Requirement Name: <i>Review recorded data.</i>	Priority: HIGH
	Description: The admins and users should be able to review the data collected.	Dependencies: 3.1, 3.2, 3.3, 3.4, 3.6
		Source: N/A
3.6	Requirement Name: <i>View data in different formats</i>	Priority: MEDIUM
	Description: Both the users and admins should be able to visualise the stored data in different formats.	Dependencies: 3.1, 3.2, 3.3, 3.4, 3.5
		Source: N/A

3.7	Requirement Name: Manipulate and group data for viewing	Priority: HIGH
	Description: The system should be able to manipulate and group the stored data in order to make it easier for the users and admins to view and analyse it.	Dependencies: 3.1, 3.2, 3.3, 3.4 Source: N/A
3.8	Requirement Name: <i>Delete the data</i>	Priority: MEDIUM
	Description: Data should be deleted after a set period of time, after account termination, or by request of deletion.	Dependencies: 3.1, 3.2, 3.3 Source: N/A
4. REQUIREMENTS RELATED TO MOTIVATIONAL MESSAGES		
4.1	Requirement Name: <i>View motivational messages</i>	Priority: LOW
	Description: The system should feature an option to send notifications to the user in the form of motivational messages.	Dependencies: N/A Source: N/A
4.2	Requirement Name: <i>Change message styles</i>	Priority: LOW
	Description: The user can select between different message styles.	Dependencies: 4.1 Source: N/A
5. REQUIREMENTS RELATED TO PERSONAL GOALS/ACHIEVEMENTS		
5.1	Requirement Name: <i>View pre-set goals/achievements</i>	Priority: LOW
	Description: The user can view their chosen personal goals/achievements.	Dependencies: 5.2 Source: N/A
5.2	Requirement Name: <i>Set personal goals</i>	Priority: MEDIUM
	Description: The system should feature an option for the user to set personal goals.	Dependencies: 5.1 Source: Personal Informatics Survey, February 2021
5.3	Requirement Name: <i>Adjust personal goals</i>	Priority: MEDIUM
	Description: Any user should be able to make changes to their personal goals.	Dependencies: 5.2, 5.1 Source: N/A
5.4	Requirement Name: <i>View personal goal progress</i>	Priority: MEDIUM
	Description: The system should highlight the progress towards a user's personal goals.	Dependencies: 5.2, 5.1, 5.3 Source: N/A
5.5	Requirement Name: <i>View goal reminders</i>	Priority: MEDIUM
	Description: The system should reminders to the user, regarding their personal goals.	Dependencies: 5.4, 5.2, 5.1, 5.3 Source: N/A
5.6	Requirement Name: <i>Divide goals into subtasks</i>	Priority: LOW
	Description: The user should have an option to divide goals into subtasks.	Dependencies: 5.1, 5.2, 5.3, 5.4 Source: N/A
NON – FUNCTIONAL REQUIREMENTS		
6.1	Requirement Name: <i>Scrum methodology</i>	Priority: HIGH
	Description: The project must follow Scrum methodology in software process.	Dependencies: 6.2 Source: Lectures, Coursework requirements
6.2	Requirement Name: <i>Three sprints</i>	Priority: HIGH
	Description: The software development cycle must be made up of at least three sprints. Each sprint should last between 1 and 3 weeks.	Dependencies: 6.1 Source: Coursework requirements

6.3	Requirement Name: <i>Review functional requirements</i>	Priority: MEDIUM
	Description: Team members must regularly review the functional requirements associated with the envisioned system features.	Dependencies: Functional Requirements, 6.1, 6.2
		Source: Coursework requirements, lectures
6.4	Requirement Name: <i>Expand upon the initial requirements</i>	Priority: MEDIUM
	Description: The team must expand upon all the initial requirements, based on their PI research.	Dependencies: Functional Requirements, 6.1, 6.2, 6.3, 6.5
		Source: Coursework requirements, PI research (including articles, interviews, surveys)
6.5	Requirement Name: <i>Expand upon the initial functional requirements</i>	Priority: MEDIUM
	Description: Must expand upon the initial functional requirements and add additional functionality to the system to deliver features the team has chosen to offer. Additional requirements should be established using appropriate requirements gathering techniques.	Dependencies: Functional requirements, 6.3, 6.2, 6.4
		Source: Coursework requirements, PI research (including articles, interviews, surveys)
6.6	Requirement Name: <i>GDPR compliance</i>	Priority: LOW
	Description: The system should be GDPR compliant.	Dependencies: 1.5, 3.1, 3.2, 3.3, 3.8
		Source: N/A
6.7	Requirement Name: Come up with a name	Priority: HIGH
	Description: Come up with a name better than "Group 3"	Dependencies: All
		Source: Our ideas
6.8	Requirement Name: <i>Read and cite articles about Personal Informatics</i>	Priority: HIGH
	Description: Must read and cite at least three articles in the area of Personal Informatics, at least one of which must be drawn from the reference section of this coursework document.	Dependencies: 6.1, 6.3, 6.4
		Source: Coursework requirements, articles related to Personal Informatics
6.9	Requirement Name: <i>Read and cite articles</i>	Priority: HIGH
	Description: The team members should read and cite at least six articles of any kind.	Dependencies: 6.1, 6.3, 6.4, 6.8
		Source: Coursework requirements, articles found while doing research
6.10	Requirement Name: <i>Test-driven development approach</i>	Priority: MEDIUM
	Description: The project must adopt a test-driven development approach.	Dependencies: N/A
		Source: Coursework requirements, the code
6.11	Requirement Name: <i>Analyse interviews and questionnaires results.</i>	Priority: HIGH
	Description: Get feedback from interviews and questionnaires.	Dependencies: N/A
		Source: Coursework requirements: PI research (including interviews, surveys)

Design

This section consists of the UML and sequence diagrams that have been created. There are three UML diagrams in total for the login and registration; the main form; and the timer. There is a total of two sequence diagrams for the login/registration and main forms, each of which explains the sequence of using different features in the app.

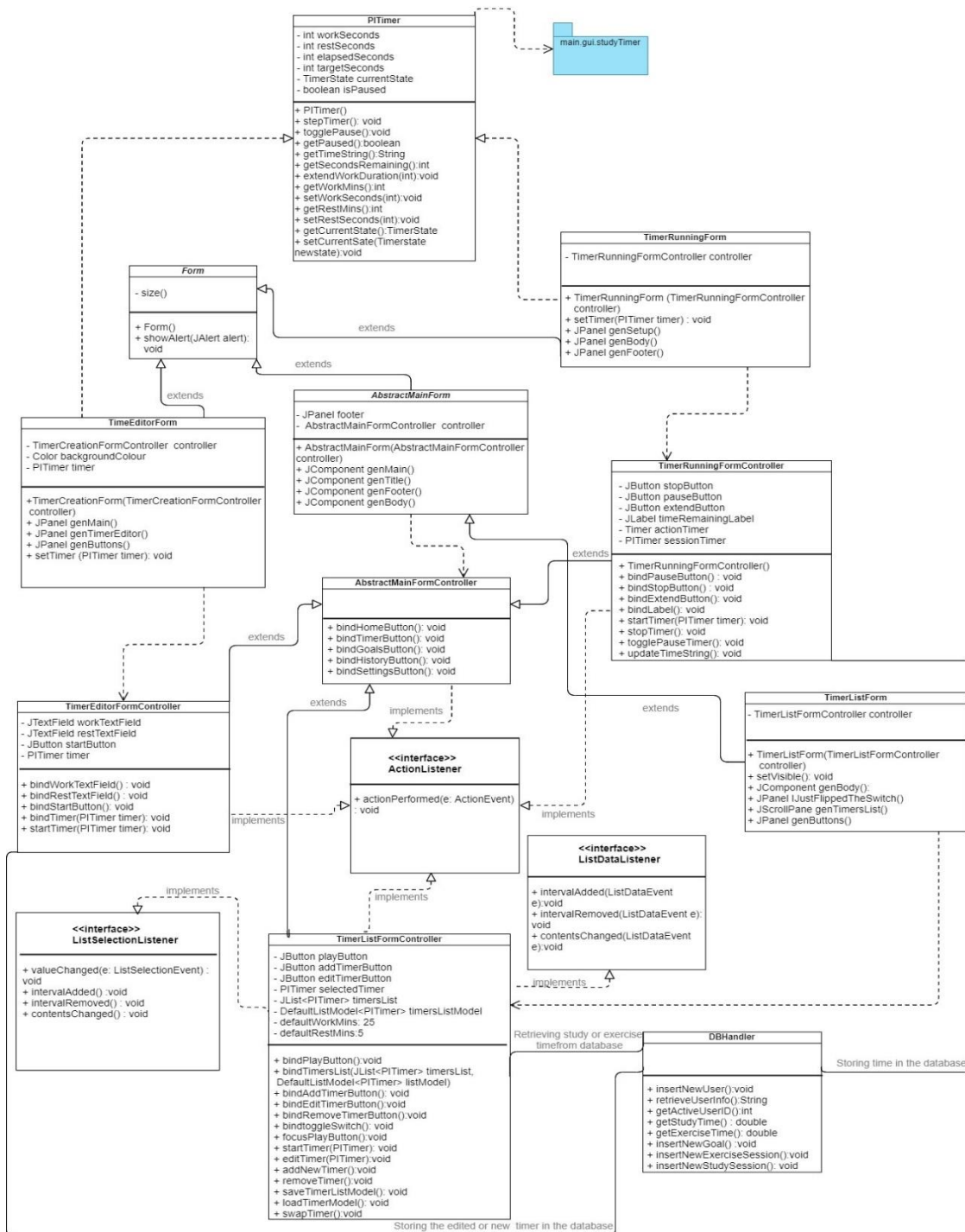


Login and Register UML

Form class under the package `main.gui` extends the `AbstractStartForm` class. The `AbstractStartForm` class, under the package `main.login`, gives the general layout of the page which can be used both for the login and register pages. The `AbstractStartForm` extends the `LoginForm` and `RegisterForm` classes which shows the concept of inheritance. Both the `LoginForm` and `RegisterForm` classes explain how the layout of the form should look. Class `RegisterFormController` takes data from the user (Username, Email, Password and Re-type password) and stores all the information in the database after verification. `LoginFormController` retrieves the data from the database when the user enters a username and password to log in. Both these classes implement an interface `ActionListener` for receiving all the action events. All the data can be stored and retrieved from the class `DBHandler`.

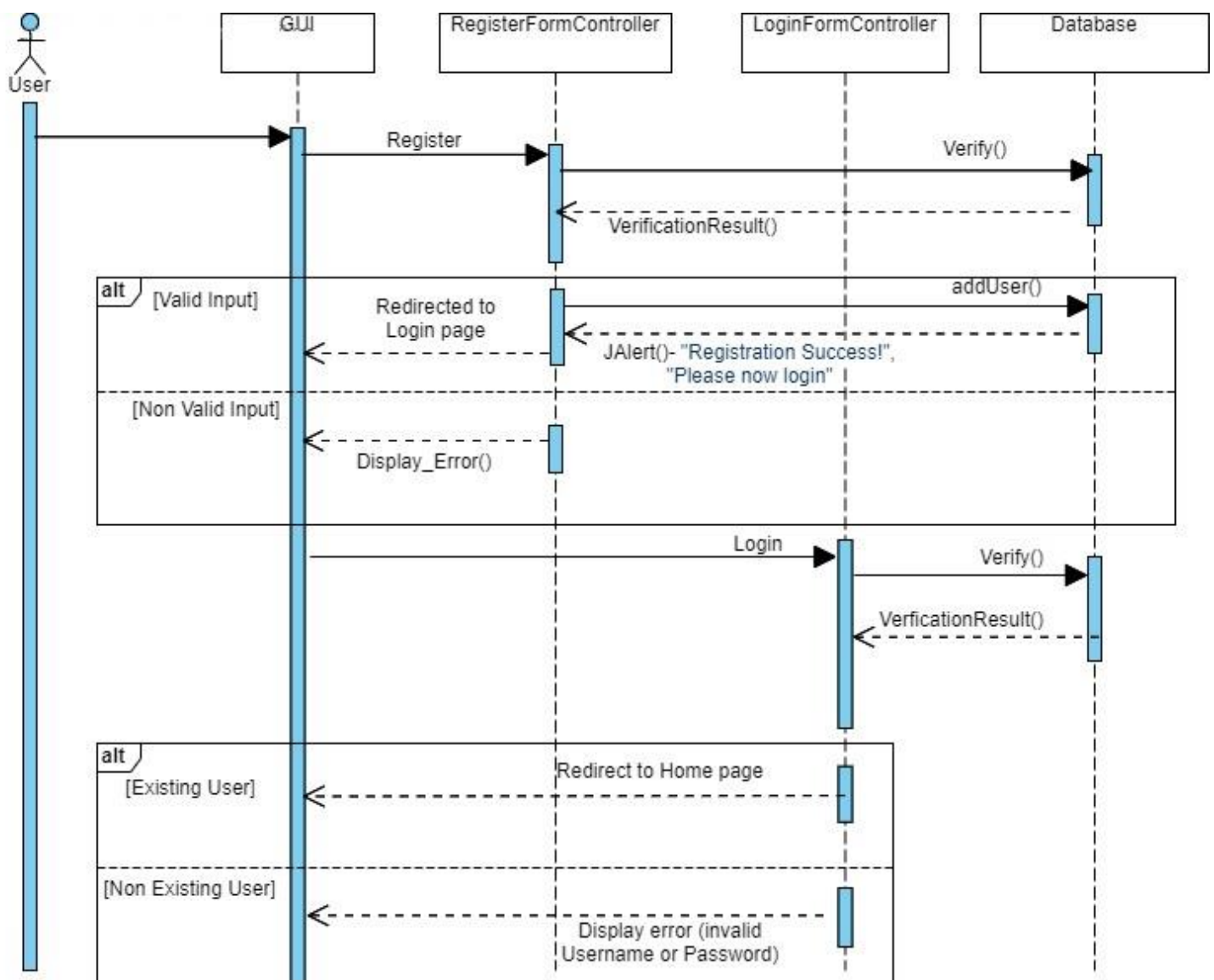
The Main Form UML is divided into two parts with one part explaining all the classes involved in the making of the timer (study and exercise) and the other part consists of all other features.

Form class under the package main.gui extends the AbstractMainForm class. The AbstractMainForm class under the package main.gui gives the general layout of the page and the buttons for traversing around the app. The AbstractStartForm has a dependency on AbstractStartFormController class which is used to bind all the buttons in the main form. Classes HomeForm, GoalsEditingForm, GoalsChoosingForm, GoalsViewForm, SettingsForm, HistoryAnalysisForm, TimerEditorForm, TimerRunningForm and TimerListForm extends the AbstractMainForm class. All these classes explain the position, styling, size, and methods required for layout creation of all user controls in the respective screens. Each of HomeFormController, GoalsCreationFormController, GoalsViewFormController, GoalsChoosingFormController, SettingsFormController, HistoryAnalysisFormController, have AbstractStartFormController as a dependency. This class implements an interface ActionListener for receiving all the action events and ListSelectionListener. All the data can be stored and retrieved from the class DBHandler. GoalsViewFormController class shows all the goals which have been stored in the database, GoalsChooseFormController class can be used to choose any pre-existing goals, GoalsCreationFormController class is used to edit any goals, SettingsFormController class is used to logout from the app, HistoryAnalysisFormController is used to display all the data and it consists of four inner classes for four buttons namely Past Week, Past Month, Past Year and Overall, which are used to display the data stored in the database when respective button is pressed, HomeFormController class is used to show the summary of the data and it also displays different motivational messages.



Timer UML

Form class under the package main.gui extends the AbstractMainForm class. There is a main PITimer class under package gui.studyTimer which is used to get and set values for each of the timer classes. The AbstractMainForm class under the package main.gui extends the classes TimerEditorForm, TimerRunningForm and TimerListForm. All these classes explain the position, styling, size, and methods required for layout creation of all user controls in the Timer screens. Each of these classes has dependency class namely TimerEditorFormController, TimerRunningFormController and TimerListFormController, which implements the class AbstractStartFormController. All the data can be stored and retrieved from the class DBHandler. TimerListFormController class is used for creating and editing timers, starting timers, or removing any timer. TimerEditorFormController class shows the screen for editing, adding, or removing the timer. TimerRunningFormController class is used to show the timer running. The only difference between the study and exercise timer is its layouts, the exercise timer is set in seconds as opposed to minutes for the study timer. Other than that, all the functionalities of both timers are similar.

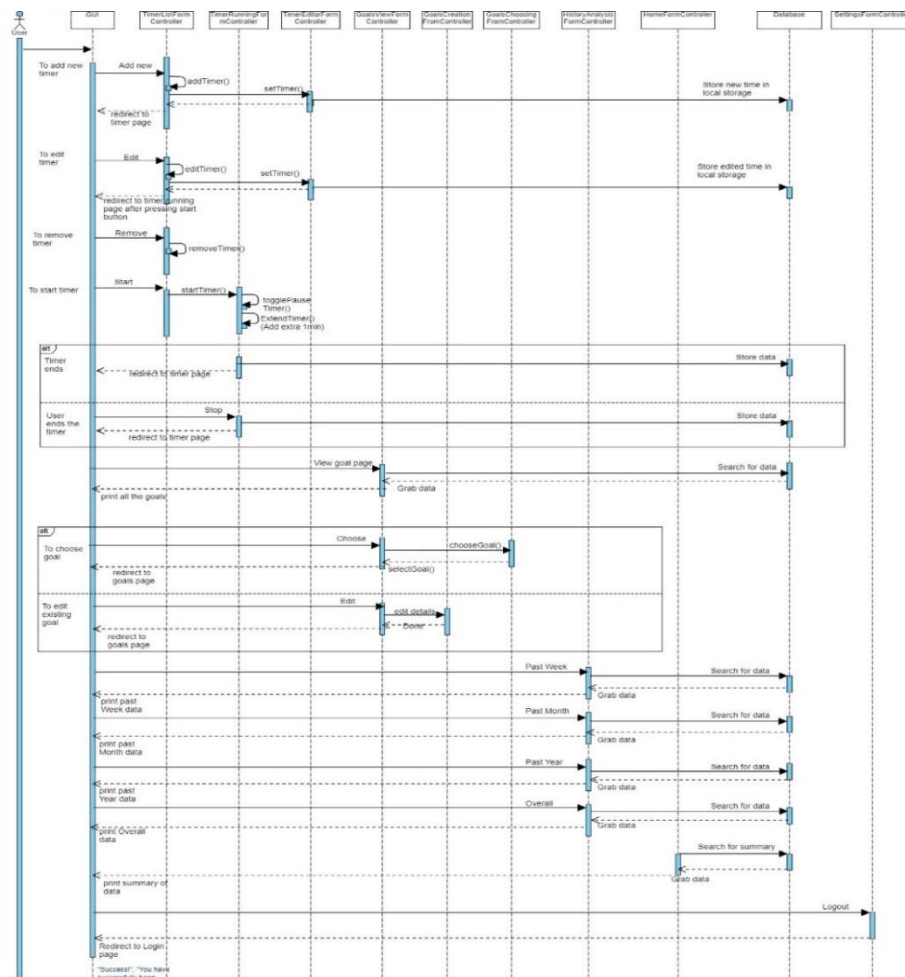


Login and Register Sequence diagram

This sequence diagram shows the lifelines: User, GUI, RegisterFormController, LoginFormController and Database. It shows a User lifeline and four standard lifelines, Call messages, reply messages and two alt fragments (for validating input and checking user existence) with two alternative options.

GUI class consists of different classes explaining the layout of the app. When the user opens the app, they are directed to the register page, where they can register by giving the details. If all the details are entered correctly, they are stored in the database, the user is registered and is redirected to the login page. If the user is an existing user, they can press the login button and enter their username and password to log in to the app, before being redirected to the home page. If the login details are entered incorrectly or the user is not an existing user, it will display an error message of an invalid username or password.

Through this, we have completed the functional requirements which consist of the user accessing their account. (i.e., 1.1-1.5), timer functionality (i.e., 2.1-2.4), storing data (i.e., 3.1-3.8), motivational messages, which are displayed in home page (i.e., 4.1) and personal goals (i.e., 5.1-5.4). We were not able to complete some of the functional requirements, so we decided to include them in our future sprints. All the non-functional requirements were done during the process of implementation of our PI system, which is not explicitly shown in the UML and sequence diagrams, but it is explained in the other parts of the report.



Main Form Sequence diagram

This sequence diagram shows the lifelines: User, GUI, TimerListFormController, TimerEditorFormController, TimerRunningFormController, GoalsViewFormController, GoalsCreationFormController, GoalsChoosingFormController, HistoryAnalysisFormController, HomeFormController, SettingsFormController and Database. It shows a User lifeline and ten standard lifelines, Call messages, reply messages and two alt fragments (one ending the timer and one for goals) with two alternative options.

GUI class consists of different classes explaining the layout of the app. When the user presses the timer button in the app, they are directed to the timer page, where there is a toggle switch to select whether to use the study or exercise timer. A user can add new, edit, remove, and start a timer. When the user presses the *Add new* button, they will be directed to TimerListForm, where they will be able to add a new time and this time will be set in the TimerEditorForm. The time will be stored in local storage. After that, the user will be redirected to the main timer page. If the user presses the edit button, they will be directed to a list of timers from which they can select and edit it, and that time will be stored in the local storage. If the user presses the remove button, they can remove any timer. If the user presses the *Start* button, the timer will start for the given time or default time (25 mins work and 5 mins break). The timer, for the study timer and for the exercise timer, will be displayed in the form: minutes and seconds. The user will be able to pause or extend the timer for 1min. When the set timer ends automatically, that time will be stored in the database, or if the user presses the stop button, the timer stops, and that time will be stored in the database. When the user presses the *Goals* button, they are directed to the goals page and search for any goals from the database and if any goals are present, they will be displayed on that page. The user can choose or edit existing goals. There will be a list of pre-defined goals from which the user can choose. When the user presses the *History* button, they will be directed to the historical analysis page where there will be four buttons (Past Week, Past Month, Past Year and Overall). The user can press any of the buttons to display that specific data which will be taken from the database. When the user presses the home button, the average sum of the data will be taken from the database and motivational messages will be

displayed. When the user presses the Settings button, the user will be directed to the Settings Page where there is a Logout button. From there the user will be able to Logout from the app and will be redirected to the Login Page.

Software Testing

Testing a software system is a pivotal part of ensuring the success of the software development process. It is a system that ensures what is known in the field of engineering as Quality Assurance, where you assure that the product you have made meets the requirements of the customer/end-user. These can be requirements specifically defined in the requirements specification; however, this can also include those that are assumed, i.e., the system must run without crashing. When testing a system, there are many different strategies that you can use, including how many people are involved with testing, and in how many ways the system is tested. However, these strategies can be different based on how “good enough” we need the system to be. (Pressman, 2019) As our system was only a prototype, and will not be put into the real world, there were certain quality assurance methods that we could forgo. It was more important to meet the set requirement specifications, rather than make a reliable program that was robust.

Unit & Integration Testing

We used unit and integration testing throughout our project to ensure that the different submodules worked and interfaced with each other correctly. For the components that were at the bottom of the architecture, we used unit testing to test the functionality of them. However, for modules higher up in the architectural design, we combined unit and integration testing. This is because a lot of the functionality of these higher modules interact with one or more other modules, so it was difficult to test them without also testing how they interact with the other modules as well.

To speed development up, we allowed the people who created the modules to write their own tests, as they are the people who best know the system, and as such we would not have to use time into explaining any small intricacies that may need to be tested. Also, tests of single modules often do not relate to specific requirement points, it is only a combination of the modules that then allows for the requirement specification to be met. If this was the case, then we ensured the testing plan was reviewed by at least one other person.

To automate integration testing, we created “mock” modules that simulated the functionality of the modules around the specific module being tested so we could simulate different scenarios. However, not everybody had enough time to do it, so instead they either got covered by the acceptance tests; as if the overall system works, then it implies that the submodules must also work. If we were aiming for high quality, then this would not be a suitable testing method, and we would instead ensure every module had been covered by unit and integration tests.

As well as testing when a module is first created development, these submodules were continuously tested, even if they were not changed (a person may change something by accident, for example, somebody may incorrectly merge two git branches), to ensure that the submodules still worked individually and together. However, due to time constraints, there were some modules that were not tested automatically. This is partly because some modules are very difficult to automate, for example, a module that provides the layout of a form.

Below is an example of unit testing for the Password Encryptor along with the test data it was given. We have tried to use a range of passwords, including a long password, and a range of salts in order to ensure that two hashes of the same password but with different salts are not the same. On the next page, there is a screenshot of the report which shows that the automated tests passed. An extended version of our acceptance testing can be found in Appendix C.2.

Reference	Salt
A	EhriHSffB8Xrgxi8LRGClb8tt
B	EhriHSffB8Xrgxi8LRGClb8tq
C	K5s7HJkhkbaHGK678HKJghsdp

Unencrypted	Hash
1234567890	\$1\$EhriHSFB8XrgI8LRCgIBcltZ25m8YxxApJ3HNDGcDAJyrogGGU/AKrcNu5UV12k=
"\$1\$1\$1"	\$1\$EhriHSFB8XrgI8LRCgIBcltZ5euklqHeWhe3w7L1N5L7oqQIUHQUBN1tRAAm=
"\$1\$1\$1\$1\$1"	\$1\$EhriHSFB8XrgI8LRCgIBcltbnN1E1CkYOnUqWd6510s9nZ73oQA0Gt9rA=
"\$1\$1\$1\$1\$1\$1\$1\$1"	\$1\$EhriHSFB8XrgI8LRCgIBcltMd6e3Zk-Lz4Bz2RfHKqfO0DwWf0CvJwXllJ19=
"this is a really long password that you would never use because you would likely forget it"	\$1\$EhriHSFB8XrgI8LRCgIBt9tK5r49dAXpITtA5eQw8r2tNCp0Z26k+mQgRe=
"\$!\$%GHE87"	\$1\$EhriHSFB8XrgI8LRCgIBclt7Va5zdr9LMyb9yuih5Gp1YrJvaZ45OTkeYkQ=

Below is an example of an acceptance test. Here, these are the tests needed to fulfil requirement 1.2, which says that a user must be able to login. It is also associated with requirement 6.6, which is GDPR, as people shouldn't be able to access accounts they don't really have access to. An extended version of our acceptance testing can be found in Appendix C.1.

Test Scenario	Test Case	Requirement	Pre-Conditions	Test Steps	Test Data	Expected Result	Actual Result	Pass/Fail
(C) A user trying to login	(1) A user logging in with the correct details	1.2	Test B.2 has been completed.	Fill in the test data Press Login	Username: user Password: abcd1234	The app should behave in a way that signifies they have logged in successfully (<u>i.e.</u> a success alert or directing them to the home page).	The app redirects the user to the home page.	
	(2) A user logging in with an incorrect username	1.2, 6.6	No users exist on the system	Fill in the test data Press Login	Username: user Password: abcd1234	The app should display a message saying "Your username/email or password is incorrect"	A red alert box pops up saying "Your username/passw ord is incorrect."	
	(3) A user logging in with an incorrect password	1.2, 6.6	Test B.2 has been completed	Fill in the test data Press Login	Username: user Password: 1234abcd	The app should display a message saying "Your username/email or password is incorrect"	A red alert box pops up saying "Your username/passw ord is incorrect."	

Reflection and Conclusion

Reflection

Reflection, as well as project improvements, are fundamental aspects of the agile methodology. A group, which is using the agile methodology, should regularly reflect on the changes that can be made to improve the project and should adapt their behaviour accordingly. During the reflection process, the team should examine their accomplishments related to the project.

The agile software processes refer to a group of methodologies that can be used to develop software, whereas requirements and solutions evolve using an agile process through the collaboration and organisation of small teams. The agile process helped us encourage teamwork, organisation, adaptation, and leadership within our group. It also taught us how to adapt to new ideas and how to improve the existing ones. In our project, we used the agile process to carry out our tasks in an organised manner and to be able to handle any changes. In addition, the agile development enabled us to quickly deliver a product of greater quality. We were required to use the scrum process, which is a subset of agile. The Scrum process is widely known to help teams increase the quality of their products, cope better with change and be more in control of the project, advantages which were also felt by our team members.

We used a project backlog to keep track of all the requirements and tasks, that each team member needed to complete to make our PI System. For that, we used an online platform called Trello. Using Trello, we were able to do small releases and adapt to changing priorities, because Trello, unlike Scrum, does not feature sprints with their predefined goals. The app helped us focus on small pieces of work. In the beginning, we did not know how to take advantage of all features Trello had to offer, but over time, we had learned how to use the application better, and consequently, had improved our organisation of the tasks. Our meetings were held on Microsoft Teams, due to the ongoing pandemic and the fact that some team members were in different countries, and all the ideas, that were brainstormed during each meeting, were noted in OneNote. As we were advancing with the project, we organised more frequent meetings during the 3 sprints. At the start of each sprint, we created a sprint backlog page in Trello which consisted of three boards: a to-do list, a doing list, and a done list. We tried to assign tasks to each team member according to their interests and preferences. During the sprint reviews, we would discuss all the tasks which were done, as well as the tasks which needed to be done in our next sprint and create a sprint backlog.

In order to obtain more information, that could help us decide the subject of our PI system, and to get to know our target audience better, we choose to create a survey, which was shared with students across the University of Bath. The survey was filled in by 60 students, who were asked questions about their sleep, exercise and study habits. From the survey results, it could be seen that only a small amount of students knew how to manage their time effectively, most of the students were getting enough sleep and that people felt a self-tracking would help them feel more in control over their study and exercise habits, along with other activities. Our decision to create a PI system related to education and exercise was widely influenced by an overwhelming amount of responses from individuals who greatly wanted to improve their study and exercise patterns, as well as the fact that the majority of people were getting enough sleep. And since most of them said that they would like to see in an exercise app an “activity recording” option, we decided to implement that feature and we made it available also to those who wanted to record their study sessions. Overall, we were satisfied with the fact that we managed to get people from different departments to fill in our survey because we were able to analyse the topics from various perspectives. We observed different study, sleep and exercise patterns among students from different departments. In the future, we would like to add open ended questions to our survey.

We created the Software Requirements Specification during our first sprint. This was the first major step in creating our application because the quality of the overall product would depend on the Software Requirements Specification. Every team member came up with ideas, and we selected the most realistic ones and the most likely to be accomplished. When we assigned the priorities in the requirements table, we assigned the **HIGH** value to the tasks we wanted to start doing at once or to tasks which represented the base of the app. We assigned the **MEDIUM** priority to the tasks we were planning to begin in the second sprint, and the **LOW** priority to requirements we were planning

to satisfy in the third sprint. This prioritisation helped us manage our tasks better and organise the team effectively. We were able to satisfy all the medium and high priority requirements and had only several **LOW** priorities remaining. For example, we did not manage to get our system GDPR compliant, but it is not completely non-compliant, we still have the password encryption feature. The existence of the Software Requirements Specification forced us to organise frequent evaluations of the project's status. As a result, we had to confront and resolve issues regularly which helped us progress towards the desired outcome.

While making the project, it was challenging for us to come up with a completed and well detailed design for the app. We tried to come up with a basic design during the first sprint, because we considered it essential for the team's coordination and a successful completion of our project. Most of the details in our design, however, had emerged through coding. We allowed the coding team to explore different ideas and to challenge themselves, especially when they produced the front end. The code was refactored to keep consistency between old and new design ideas. When designing the front end, we approached the design from two angles: the user experience design and the visual design. Both aspects were important for our application because they enhanced the user experience, increased the customer satisfaction and are usually essential for preventing PI system abandonment. GDPR compliance is a feature we would like to improve in future sprints.

When designing the GUI for the app, we initially started with a very simple home-screen to provide a general idea for navigation buttons and structure. However, during the meeting at the end of the first sprint, we reviewed and determined that the basic home-screen we had designed was insufficient in providing a concrete structural idea for the independent team members to consider when working on the backend. This meant that there was difficulty exactly determining what functionalities and extra implementations may be required. As a result, in the second sprint we improved the design of the GUI significantly by laying out the specific pages and the features we would like to implement so that each member could have a clearer idea of what different aspects were required of the app, and how they would interact. By following the SCRUM process, the effective communication in our first sprint review helped us identify the issue and adapt accordingly. Having set the new GUI layout as the target design, the team was progress more fluidly throughout the SCRUM process with clearer tasks and improved discussions in meetings. Despite having overcome the problem, we realised how laying out clear goals and tasks in initial sprints could improve communications and outcomes of future sprints and will certainly implement it in the future SCRUM projects.

To enable the app to have the interactivity that comes from users being able to track their app usage over time, we decided to include a history analysis screen which would include various metrics about the user's data. In our first sprint, it was decided that we wanted to focus on the core components of the application and build up the backend before implementing this screen as it required the use of many components. However, although we did not provide an implementation in our first sprint, we began the planning process for this feature during our SCRUM meetings which helped us to implement it much more efficiently in a later sprint, thus showing the importance of requirements engineering. When we came to sprint three, we had everything in place to create the history analysis screen and were able to create a robust prototype due to having the components planned out in prior sprints, including a design of the screen. This sprint allowed us to create a summary of the data over multiple different time frames which could be chosen by the user. In a future sprint we aim to create a graphical display of the information in order to further improve the interactivity and engagement for the user.

The timer is the main functionality of the app, it allows the user to record study and exercise sessions. Following the pomodoro methodology, after a set period of work has passed, a period of rest will follow. In the old design, after traversing to the timer page you were prompted to define exact durations of work and rest before starting the timer. It would then take you to a page which starts in the work state, counts down the duration set for work then switch to the rest state and count down the duration set for rest. In the new design we changed the page you are confronted with when first entering the timer page. Inspired from other timer apps, you can now you can create multiple timer setups which are stored in a list you can later select and edit them from. This was later improved further by creating a separation between study and exercise timers. This is done by having a toggle button at the top of the screen to switch between the two modes. These two modes are functionally very similar, however there are slight differences. For instance, they have different background colours to help differentiate them, as well as exercise timers being displayed

in seconds while study timers are in minutes. When a work session is completed, or when it has prematurely ended, the session time is stored in the database. These times can then be used by the other parts of the app such as the history analysis. One idea for a future implementation is the ability to string timers together. An example in which this would be useful would be when practicing multiple types of exercise in a single session, you may not want uniform work durations for each exercise. To implement this, instead of storing single timers it would be storing lists of timers.

We also chose to include motivational messages in our project because they are a constant part of our lives. Nowadays, you cannot access social media without seeing posts about motivational messages. People read motivational messages because the messages have wisdom in them, people find them inspiring, and the quotes help individuals feel motivated in order to achieve their best. Life is tough and it can be comforting to know that we are not alone, that other people have experienced our issues and there is hope for situations to improve. Sometimes, a little bit of motivation and encouragement are the only things we need to finish a task, so we wanted to bring this kind of positivity into our project through the quotes we included. We understand the powerful human desire for accomplishment, so we hope our project can help individuals achieve excellent results in their lives with a well-planned schedule and our product. Initially, we were planning on including motivational messages related to education and exercise. But because we were struggling to get the basic functionalities done and the process took us longer than expected, we decided not to include them in our project for now, however, this is a feature we would like to implement in the future. Another feature we would like to implement in future sprints, is related to motivational messages which appear as notifications. We want the user to receive a motivational message whenever they accomplish a goal, or they finish a task and even during the study/exercise sessions.

Personal goals incentivise the user to regularly use the app, encouraging studying and or exercise. Each user has a list of goals which keeps track of *in progress* and *completed goals*. Functionally, they are just a data class holding things like title, end date, progress, etc, with some mutator methods for interacting with these. You can add, edit, and remove goals from your current list in the goals page of the app. At first the user was able to create open ended goals with no indication for how these would be completed from a technical standpoint. In other words, the app would never know when to update the goals. To fix this we changed the design of the goals page, replacing the ability to *create* goals with the ability to *choose* goals. The difference between the two being that the user must now choose from a predefined set of goals. The user cannot choose duplicate goals in their list. This allows for the app to recognise when it should be updating the goals internally. The goals along with their progress are saved to the database. However, due to time constraints the feature of loading the goals back on start-up, along with updating them after timed sessions was not implemented. Our original design was to make the app query the database for current goals for the current user on start-up. For updating goal progress, at the end of each session (or after a relevant event is fired such as a step being taken) any goals related to that event would be updated for the current user.

Testing of the software system was a pivotal part in ensuring the success of the software development process, as well as the final product. When testing the code, we had taken two major approaches: Unit and Integration Testing, and Acceptance Testing. It was necessary to ensure successful completion of the main software requirements in the form of acceptance tests, to raise the overall quality and to make sure the expectations were met. However, as our system was only a prototype, we did not consider it a priority to produce production-ready software, so we did not spend too much time on unit and integration tests.

The database went through many radical changes during the development of the app. Though there were many initial critiques with the system upon proposal, it is using a MySQL server hosted on Amazon server space. It first started out as a local SQLite server, but when that proved to be more trouble than it was worth, it was moved to a MySQL server hosted on a personal server. The personal server proved to be unstable at times, so it was therefore moved to an Amazon based hosting service, and this has improved stability considerably. As for the design of the database itself, nothing was removed from the original implementation of the database. The original tables and record names remained the same, and the only alterations to the code that needed to be made because of this were changes to the connection statements, and some of the query statements when retrieving data. Aside from these changes, the database remained consistent for the entirety of the project's length. This demonstrated the stability and quality of the database that was originally designed, and showed that through the Agile system, that we were able to roll out

new changes to satisfy any needs. As a result, no changes to the database were necessary as the database was designed with the scope for expansion.

When the database was corrected, the next step was to create a system so that the app could interact with the database fluidly and without any direct user input. This led to the creation of a universal handler which handles all the database connections, queries and related requests, the changes that have been made to the handler have already been discussed within the previous paragraph. An issue still exists with retrieving certain information where a list of results needs to be returned, an upper limit of 999 has been placed, as lists do not normally work within Java. The database handler also had issues regarding the connection statements inside of it, namely improper driver statements and slightly different URLs. But given the Agile system of work, we were able to quickly spot it and fix the issue. If anything was to be changed with the database handler, it would be implementations of list, and an attempt of an implementation of a skeletonised framework where new functions can be made almost immediately, but apart from that, there are no discernible flaws with the handler. In a future sprint, new setters and getters need to be added as well as skeletonising the system to make it faster in general.

The database and the handler also depended on a password encryptor not only for security services, but also to make it GDPR compliant. The password encryptor exists as its own entity and creates a unique hash for each password by concatenating the hashed password and the salt, then when verifying and checking the passwords, the salt and the password are then separated to make comparisons easier between passwords. Originally, it was using a global hash for each password, however this was an issue as if two people had the same password, they would have the same hashed value, which isn't as secure as having a unique hash for each user. The main problems with the encryptor come from the fact that it is a separate class rather than an integrated function into the database handler. The Agile process did make an issue evident with how the salt values are constructed to be solved rather easily, but apart from that, there are no flaws with it. The solution was that the salt values need to be constructed before the fact, and stored globally to ensure that the salt values remain consistent across every password. The sprints allowed us to separate each of these into their own distinct sprints and make it much easier to focus on each specific thing, which let us make sure that they were made as well as they can be. In future sprints, minimising the code within the encryptor is a necessity to make it into a function within the handler, as opposed to its own class.

Conclusion

When developing our PI system aimed at the current university students, we followed the SCRUM structure. During the Agile process, we learnt that a lack of clarity can make it difficult for group members to develop on it further. As such, we quickly solidified the main idea in the next sprint review. Similarly, we realised the importance of how clear our backlog and requirements were. By having specific project goals alongside visible up-to-date progress, it made it easier for our team to notice and target problems. When we implemented this in the second sprint, our communication was greatly improved, and our team was able to progress more smoothly.









Overall, there were many targets that were met, but also some that we were unable to achieve. Initially, our lack of experience in using SCRUM evidently made it difficult for us to maintain the pace of progress we aimed to have. However, as the project went on, we were able to improve on our use of SCRUM and greatly improve the efficiency and accuracy of our group work. By actively using SCRUM in a project, we are much more confident in applying it, and are confident in using it in the future.

References

- Atma, H.W., Handarini, D.M., Atmoko, A., 2019. Correlations Analysis of Self-Motivation and Time Management on Academic Procrastination in Students of Public Junior High School of Malang. *ACM* [Online]. Available from: <https://dl-acm-org.ezproxy1.bath.ac.uk/doi/10.1145/3345120.3345122> [Accessed 28 April 2021]
- Cirillo, F., 2006. *The Pomodoro Technique* (The Pomodoro). Morrisville, North Carolina, United States: Lulu.com
- Gulotta, R., Forlizzi, J., Yang, R., Newman, M.W., 2016. Fostering Engagement with Personal Informatics Systems. *ACM* [Online]. Available from: <https://dl-acm-org.ezproxy1.bath.ac.uk/doi/10.1145/2901790.2901803> [Accessed 28 April 2021]
- Kaser, C.H.. Teaching Students to Self-Monitor Their Academic & Behavioural Performance. Available from: <https://www.odu.edu/content/dam/odu/col-dept/cdse/docs/5-self-monitoring.pdf> [Accessed 28 April 2021]
- Karanam, Y., Filko, L., Kaser, L., Alotaibi, H., Makhsoom, E., Volda, S., 2014. Motivational affordances and personality types in personal informatics. *ACM* [Online]. Available from: <https://dl-acm-org.ezproxy1.bath.ac.uk/doi/10.1145/2638728.2638800> [Accessed 28 April 2021]
- Li, I., Dey, A., and Forlizzi, J. 2010. A staged-based model of personal informatics systems. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* [Online]. Available from: https://dl.acm.org/doi/abs/10.1145/1753326.1753409?casa_token=pXw3xkjaGAcAAAAA%3ARGx7ad8R9GAieZRx8rUe_S3OaJM-duhKSwgYm70HZCRQoweW4loYd71Xm-eZXlDXmYr7Z-Y_OiCF [Accessed 28 April 2021]
- Mule, S.S., Waykar, Y., 2015. Role of use case diagram in software development. *International Journal of Management and Economics* [Online]. Available from: https://www.researchgate.net/publication/322991847_role_of_use_case_diagram_in_software_development [Accessed 26 April 2021].
- Potapov, K., Lee, V.R., Vasalou, A., Marshall, P., 2019. Youth Concerns and Responses to Self-Tracking Tools and Personal Informatics Systems. *ACM* [Online] Available from: <https://dl-acm-org.ezproxy1.bath.ac.uk/doi/pdf/10.1145/3290607.3312886> [Accessed 28 April 2021]
- Pressman, R., Maxim, B. 2020. *Software Engineering: A Practitioner's Approach* [Online]. 9th Edition. New York City, United States: McGraw Hill. Available from: <https://www.mheducation.com/highered/product/software-engineering-practitioner-s-approach-pressman-maxim/M9781259872976.html> [Accessed 28 April 2021]
- Roig, M., Nordbrandt, S., Geertsen, S.S., Nielsen, J.B., 2013. The effects of cardiovascular exercise on human memory: A review with meta-analysis. *ScienceDirect* [Online]. Available from: <https://www.sciencedirect.com/science/article/abs/pii/S014976341300167X> [Accessed 28 April 2021]
- Singh, A., 2021. How Much Sleep Do We Really Need?. *Sleep Foundation* [Online]. Available from: <https://www.sleepfoundation.org/how-sleep-works/how-much-sleep-do-we-really-need> [Accessed 28 April 2021]
- Walker, M.P., 2008. Cognitive consequences of sleep and sleep loss. *ScienceDirect* [Online]. Available from: <https://www.sciencedirect.com/science/article/abs/pii/S1389945708700145> [Accessed 28 April 2021]

Appendices

Appendix A (Group Contribution Form)

GROUP MEMBER	FIRST SPRINT	SECOND SPRINT	THIRD SPRINT	FINAL CONTRIBUTION	SIGNATURE
H. KIM	8.57	8.85	9.14	9	
C. DAVIES	6 (was 6.84)	7.71	8.5 (was 6.42)	7	
G. SIMONETTI	8.14	8.57	8.42	8	
W. JONES	8.14	8.42	8.28	8	
A. BARRELL	8.57	9.14	9.57	9	
J. RABJOHNS	8.42	8.71	8.85	9	
S. SUJIT	8.14	8.42	8.71	8	
I. MOCANU	8.14	8.42	8.42 (was 8.28)	8	

Appendix B

06/02 10:00 - 11:00

06 February 2021

09:12

Target audience is students

1. Exercise
2. Food - <https://developer.edamam.com/food-database-api>
3. Sleep
4. Productivity tracker
5. Mental health - How was your day?

Data structure?

1. Checklists
2. Correlate data together, e.g. sleep / percentage of items completed

Exercise & Mood

WHO advice on exercise n stuff

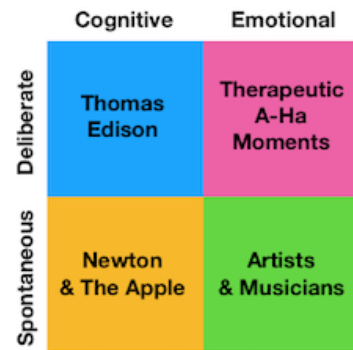
<https://www.who.int/news-room/fact-sheets/detail/physical-activity>

08/02 9:15 - 11:30

08 February 2021

09:23

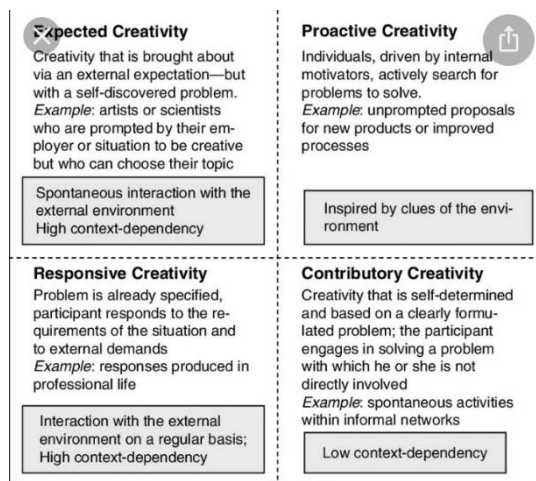
Q. When you tackle problems that require a programmed solution, you are being creative. How would you describe this kind of creativity?



Deliberate and cognitive creativity in initial stages

Proactive creativity

Contributory creativity



Expected creativity

Proactive creativity

Responsive creativity – responding to requirements given
 Contributory creativity

Q. Do people tend to be more or less creative when they work with others?

Overall more likely to be more creative:

- You are able to bounce ideas off of each other
- Even if you may have a more introverted personality, being presented with various ideas could help you think more broadly in your time and space

Q. Think about the programming tasks you have done before, how would you describe the steps you take to solve them?

Checking and understanding the requirements of the task

Outlining potential steps required by the task

Problem decomposition and creating a general structure where implementable

Start programming

Test cycles and code refinement

Q. How do you think these steps compare with the way your classmates tackle programming tasks?

There may be differences in the process regarding the frequency and timing of tests:

Iterative approach vs Test driven approach

- Education and Sleep
- Education and Mood
- ~~Fitness and Sleep~~
- ~~Fitness and Mood~~
- ~~Sleep and Mood~~
- Daily goals (Education?) and Mood
- Daily goals (Education?) and Sleep
- ~~Fitness and Sleep and Mood~~
- ~~Food and Education~~
- ~~Food and Sleep~~

Ways to implement education:

- To do list - % completion
- Study timer - minutes completed
- User input - minutes completed

Ways to implement sleep:

- User input - how tired are you feeling?
- User input - how did you feel your sleep quality was?
- Pre-set to do list - how much of the activities did you complete?
- Panopto API - <https://support.panopto.com/s/article/api-0>

Ways to implement mood:

- User input - how was your day today?

Education with sleep

Education with exercise

15/02 9:15 - 10:51

15 February 2021

09:45

Sleep data

How was your sleep last evening?

Microphone access + Sound analysis?

Poor sleep might be caused by health conditions? (Tip: go see a doctor)

Exercise data

User input

Step measuring app?

Let's start with Education, then we decide if we go all in!!

Time management, time awareness

Produce a list of requirements

Required

1. ~~Permit a user to compare their data over time (e.g. to see if they have walked 5% more each day for the last n days, or if their diet is regularly less vegetable-intensive on a Monday).~~
2. ~~Permit a user to compare different kinds of PI data within a fixed period of time (e.g. to find out if changes in their mood have followed changes in their diet over the last 14 days)~~
3. ~~Viewing and Collecting Tracking Data Store data on user activity relevant to the chosen PI concept (see Section 3 'Tracking Data' for more detail).~~
4. ~~Allow the user to access tracking data that is stored.~~
5. ~~Permit the user to manually enter any tracking data which can not be obtained automatically from a tracking device or service (e.g. allowing the user to enter the number of cups of coffee consumed each day, or to enter a rating that represents their mood for the day)~~
6. ~~Permit a user to manage targets (goals) for a tracked activity.~~
7. ~~Permit a user to set a daily or weekly goal value for a particular data variable (e.g. Steps Goal \geq 10000, Productivity Goal \geq 60%, Cups of Coffee Goal \leq 2).~~
8. ~~Permit a user to update or change a goal.~~
9. ~~Incorporate a feature to motivate the user to achieve their goals (e.g. scoring points, receiving trophies/badges, competing with other users)~~

Viewing and Collecting Tracking Data

Store data on user activity relevant to the chosen PI concept (see Section 3 'Tracking Data' for more detail).

Permit the user to manually enter any tracking data which cannot be obtained automatically from a tracking device or service (e.g. allowing the user to enter the number of cups of coffee consumed each day, or to enter a rating that represents their mood for the day)

Initial Requirements Backlog

Research

Survey

UI\User Experience

Sign Up/Login

Goals/Deadlines

Informatics

Data input

- Timer

Data access

- Who can access what data

- Limitations of access?

Data viewing

- Data graphics

- How many hours studied today so far

- How many hours this week

- How many hours this month etc

- Being able to track your results for each module

Data analysis

- How does the data compare to other data

- Any specific patterns in data?

Intervention

- Timer

- Base time - 25/5

- Adjustable times

- Can extend timer duration at the end of chosen time

- Can submit data \leq the chosen time

Motivation mechanism

- Stops timer if certain actions are done

- Messages - "You better be doing your work", "Good job" + motivational quotes?

Goals/Targets

- User can adjust goal

- Measured in hours/minutes studied?

- Gameify- Using points from studying to customise

Persistent Data Storage

- Remote Server or Local Storage? (Probably loosely coupled to offer choice/flexibility)

22/02 9:15 - 10:50

Monday, 22 February 2021

11:27

Functional requirements

View signup page

View login page

Access through login

View account details

Edit account details

Ability to record exclusive time data

Ability to locally store data

Ability to remote store data

Ability to review the metadata

Ability to review exclusive recorded data

Ability to view data in different formats

Ability to manipulate and group data for viewing

Ability to delete data retroactively

Ability to reduce time data

GDPR compliance

Ability to use timer - base time 25/5

Ability to adjust timer's time before starting

Ability to prematurely end study session

Ability to extend study sessions

Ability to view motivational messages

Ability to view goal reminders/summary

Ability to change message styles

Ability to view Preset goals (achievements)

Ability to view achievements

Ability to set personal goals (time studied by date?)

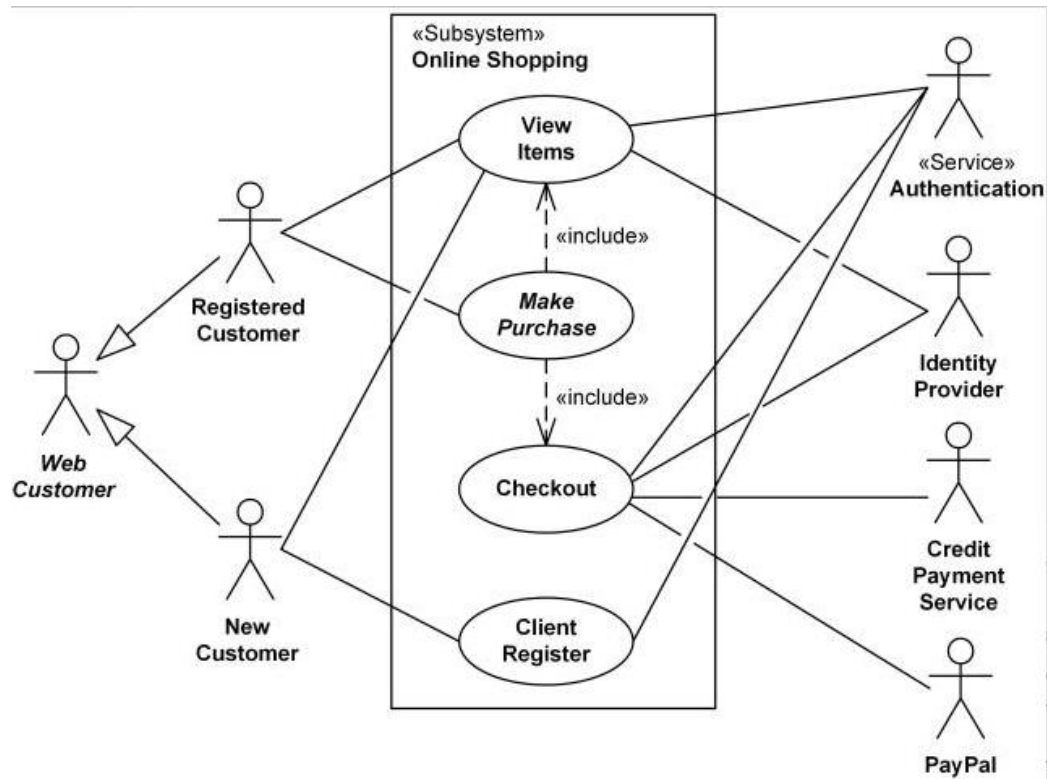
- Ability to adjust personal goals
- Ability to view personal goal progress
- Ability to view goal reminders/summary
- Ability to divide goals into subtasks

Non-functional requirements

- Follow Scrum methodology in software process
- Software development made up of at least three sprints
 - Each sprint should last between 1 and 3 weeks
- Must regularly review the functional requirements associated with envisioned system features.
- Must expand upon the all initial requirements, based on your PI research
- Must expand upon the initial Functional requirements and add additional functionality to the system to deliver features you have chosen to offer
- Additional requirements should be established using appropriate requirements gathering techniques
- Must read and cite at least three articles in the area of Personal Informatics, at least one of which must be drawn from the reference section of this coursework document.
- Should read and cite at least six articles of any kind
- Must adopt a test-driven development approach, including
- Must provide evidence of testing (e.g. JUnit output)
- GDPR compliance
- Get feedback from interview
- Get feedback from preliminary questionnaire
- Get feedback from later questionnaire

Use Case Tool

<https://online.visual-paradigm.com/diagrams/solutions/free-use-case-diagram-tool/>



Actors

New users
Basic user
Advanced user
Leaving users

Admin user
Database

01/03 9:15 - 10:45

01 March 2021

09:20

Design first sprint

Start and finish first sprint by week 7

Backlogs

Prepare for the sprint (week 7 max)!

Include requirements

We worked on the report, discussed the survey.

Meeting on Friday at 9:15 UK.

05/03 9:25 - 10:45

Friday, 5 March 2021

09:27

- We worked on the table with the functional/non-functional requirements.
- And tasks table.
- Gio, Archey, Will - front-end

Database structure – UML diagram

Timer – Start, End after certain time, pause, extend time

Account - sign up, log in

Research – Analyse initial questionnaire data

Finalise relevant articles

Write up the introduction

Finalise use case diagram

Finalise and fill requirements table

10/03 13:30- 14:00

10 March 2021

13:40

	Start	End	Spring meeting
First sprint	8/03	22/03	22/03
Second sprint	23/03	12/04	12/04
Third sprint	13/04	26/04	26/04

15/03 9:15 - 10:30

15 March 2021

09:30

Make testing plan

Review requirements

Clarify what's going on with frontend

22/03 9:15 - 10:00

22 March 2021

09:23

26/03 SPRINT REVIEW 1 14:15-

Friday, 26 March 2021

14:17

The marks from the survey:

Archev - 8,57

Connor - 6,85

Gio - 8,14

Hail - 8,57

Ioana - 8,14

Jay - 8,42

Shikha - 8,14

Will - 8,14

Change the requirements, regarding the system.

We had a look at the UML diagrams.

Remove from the backlog anything not related to the system (i.e requirements; UML diagram; write introduction)

Separate sprint backlog for requirements

We had a look at the test table

Record unit tests - pass/fail

29/03 9:15 -10:00

Monday, 29 March 2021

09:34

We had a look at the requirements table.

We've decided which requirements we want to do for the 2nd sprint

5/03 9:15 - 10:15

Monday, 5 April 2021

09:26

We talked about Easter

We talked about the timer

Do we want very specific motivational messages? How do we adjust the rest of the code?

Go ahead with switching 2 pages

09/04 9:15-10:30

Friday, 9 April 2021

09:32

We had a look at the UI

Some of the daily goals will be generic

Create/remove goal

Show goals from the past 7 days

We debugged the motivational messages

12/04 SPRINT REVIEW 2 9:15-

Monday, 12 April 2021

09:38

We had a look at the UL diagram.

We presented the project to our tutor.

The timer, motivational messages, UI, UML, database

We divided the tasks for sprint 3.

Including the report - Ioana abstract, testing, reflection + conclusion

Hail- reflection, design, software requirements

Shikha-agile, design

Software testing- the coders
Motivational messages: General messages on home
Exercise and study ones on the respective timers

14/04 14:00-15:30

Wednesday, 14 April 2021

14:27

The report has a page limit.
We need to talk about the video.

- 2 min
- Can't exceed 3 minutes
- Show software 1 min at least/at least 1 min on the actual app
- Video needs to express the PI concept

Can a user analyse data over time?

Ioana will make the survey for sprint 2 marks.

Every needs to complete it.

19/04 9:15-

Monday, 19 April 2021

09:27

We asked questions about the report to the tutors.

Appendix C.1 (Acceptance Testing)

Test Scenario	Test Case	Requirement	Pre-Conditions	Test Steps	Test Data	Expected Result	Actual Result	Pass/Fail
(A) Tests related to the timer	Timer setup screen	2.1, 2.2	N/A	Enter work duration in text box	Work duration (amount of time)	Same number appears on timer in running timer screen	Same number appears on timer in running timer screen	
		2.1, 2.2	The preliminary work duration is already set.	Enter rest duration in text box	Rest duration	Same number appears on timer in break screen.	Same number appears on timer in break screen.	
			All durations are set.	Press start button	N/A	Transitions to running timer screen, starts timer	Transitions to running timer screen, starts timer	
	Running timer screen	2.1, 2.3	The timer is already running	Press pause button	The values of the timer	Pauses timer, transitions to paused timer screen	Timer does not pause	
				Press stop button		Stops session, transitions to setup timer screen	Stops session, transitions to setup timer screen	
				Timer reaches 0		Records session, transitions to break screen	Records session, transitions to break screen	
	Paused screen	2.1, 2.3	The timer is paused	Press resume button	N/A	Transitions back to timer screen, resumes timer	Transitions back to timer screen, resumes timer	

				Press stop button		Stops session, transitions to setup screen	Stops session, transitions to setup screen	
	Break screen	2.1, 2.3	The timer stops running	Timer reaches 0	The state of the timer	Transitions to running timer screen, starts timer	Transitions to running timer screen, starts timer	
				Press stop button		Stops session, transitions to setup screen	Stops session, transitions to setup screen	
(B) A user creating an account	(1) Access the registration form.	1.1	N/A	Press Register Button	N/A	The app switches to a page with a registration form on it	The app switches to the registration page.	
	(2) Create an account	1.1	The app is on the registration form. No other users are registered on the system.	Fill in the Test Data Press Submit	Forename: user Surname: surname Username: user Email: u@example.com (Retype)Password: abcd1234	The app switches to the login form, with a success alert.	On registering, the app switches to the login page with a visible success alert.	
	(3) Create a duplicate account with the same username	1.1	Test A.2 has been completed.	Press Register Fill in the Test Data Press Submit	Forename: user Surname: surname Username: user Email: a@example.com	The app should display an error alert saying a user with that username already exists	The app displays an alert saying the username already exists.	

					(Retype)Password: abcd1234			
	(4) Create a duplicate account with the same email	1.1	Test A.2 has been completed, and the app is back on the registration form.	Fill in the Test Data Press Register	Forename: user Surname: surname Username: user2 Email: u@example.com (Retype)Password: abcd1234	The app should display an error alert saying a user with that email already exists	The app displays an alert saying the email already exists.	
	(5) Create an account with different password/re type password values	1.1	The app is on the registration form.	Fill in the test data Press Register	Forename: user Surname: surname Username: user Email: u@example.com Password: abcd1234 Retype Password: 1234abcd	The app should display an error message saying their passwords don't match.	The app displays an alert saying the password's don't match.	
	(6) Invalid email address	1.1	The app is on the registration form.	Fill in the test data Press register	Forename: user Surname: surname Username: user Email:	The app should display an error message saying their email address is invalid	The app displays an alert saying their email address is invalid.	

					u.example.com (Retype)Password: abcd1234			
(C) A user trying to login	(1) A user logging in with the correct details	1.2	Test B.2 has been completed.	Fill in the test data Press Login	Username: user Password: abcd1234	The app should behave in a way that signifies they have logged in successfully (i.e. a success alert or directing them to the home page).	The app redirects the user to the home page.	
	(2) A user logging in with an incorrect username	1.2, 6.6	No users exist on the system	Fill in the test data Press Login	Username: user Password: abcd1234	The app should display a message saying "Your username/email or password is incorrect"	A red alert box pops up saying "Your username/passw ord is incorrect."	
	(3) A user logging in with an incorrect password	1.2, 6.6	Test B.2 has been completed	Fill in the test data Press Login	Username: user Password: 1234abcd	The app should display a message saying "Your username/email or password is incorrect"	A red alert box pops up saying "Your username/passw ord is incorrect."	

Appendix C.2 (Unit & Integration Testing)

Module	Test Case	Requirement	Pre-Conditions	Test Steps	Test Data	Expected Result	Actual Result	Pass/Fail
(A) Motivational messages	(1) A user can receive motivational messages.	4.1, 4.2	There is an additional .txt document containing motivational messages.	Request motivational messages?	The document containing the motivational messages	The app should display a quote or more.	The user receives a motivational message in the home page and other messages in the exercise/study page.	Pass
	(2) A user can receive different types of motivational messages.		There is an additional .txt document containing motivational messages. The motivational messages are divided in different categories.	Request a certain type of motivational message.			The user receives only the general motivational messages.	Fail
(B) General Database Handler Testing	(1) The program can create new information within the database	1.3, 1.4, 3	Database set up to receive information	Call the relevant function to create a new record	The database containing all the information, and various user inputs pertaining to each different test type	The database should create a new record of relevant data		Pass

	(2) The program can retrieve relevant information		Database set up with information to retrieve	Call the relevant function to retrieve information from the database		The program should retrieve relevant information from the database	Some methods are limited to returning 999 records. For prototyping, this is not an issue as there won't be that much data.	
	(3) The program can delete relevant information		Database set up with information to delete	Call the relevant function to delete a certain record from the database		The program should delete relevant information from the database		
	(4) The program can update relevant information		Database set up with information to update	Call the relevant function to update a certain record from the database		The program should update relevant information in the database		
	(5) Have the user input inputs synonymous with SQL injection		Database set up to receive information	Have the user input the relevant data, and have that data be put through the relevant SQL queries		The program should not be vulnerable to SQL injection, and shouldn't return a value	Most methods are protected, with the exception of getUserId, however inputs of this method are only given from data retrieved from the database, so is not too much of an issue.	
	(6) Test if the program cannot		N/A	Call the relevant connection functions		An error message should pop-up in the console, showing the relevant error		

	successfully reach the database							
Password Encryptor	(1) Encrypting the password	1.5		Using the defined salt with reference A, hash the passwords	See Appendix G	The resulting hashes should match the hashes in Appendix C.	All tests passed	
	(2) If two users have the same password, then their hashes should be different.	1.5		Hash the passwords for each of the given salts.	Appendix G	For each iteration of hashing for each password, the hashes should not match.	All tests passed	
	(3) Comparing a hashed password with an unencrypted one.	1.5		For each of the hashes, try verifying it with each of the unencrypted passwords.	Appendix G	It should only return true if the hash is verified with the unencrypted password that the hash was originally generated with.	All tests passed	
Login Form Controller	(1) Logging in with valid credentials	1.1	(1) An instantiated Login Form Controller with a Mock Authenticator, and mock user inputs (2) A Mock Form registered with the Screen as "main"	Fill in the username/password mock inputs and click the mock login button.	Username: User Password: Pass attemptLogin returns True	(1) Call mock authenticator attemptLogin with "User" and "Pass" (2) Call mock authenticator login with "User" (3) Show main form	(1) The correct username/password was passed (2) The correct username was passed to the login method (3) The main form was shown	

	(2) Logging in with invalid credentials	1.1	(1) An instantiated Login Form Controller with a Mock Authenticator, and mock user inputs (2) A Mock Form registered with the Screen as "main"	Fill in the username/password mock inputs and click the mock login button.	Username: User Password: Pass attemptLogin returns False	(1) Call mock authenticator attemptLogin with "User" and "Pass" (2) Add error to alert pane	(1) The correct username/passwor d was passed (2) An alert was added to the alert pane	
	(3) Logging in but the Server fails to connect	1.1	(1) An instantiated Login Form Controller with a Mock Authenticator, and mock user inputs (2) A Mock Form registered with the Screen as "main"	Fill in the username/password mock inputs and click the mock login button.	Username: User Password: Pass attemptLogin throws Server Connection Exception	(1) Call mock authenticator attemptLogin (2) Pass error to frontend	(1) The correct username/passwor d was passed (2) An alert was added to the alert pane	
	(4) Logging in with Invalid User Input	1.1	(1) An instantiated Login Form Controller with a Mock Authenticator, and mock user inputs (2) A Mock Form registered with the Screen as "main"	Fill in the username/password mock inputs and click the mock login button.	Username: User Password: Pass attemptLogin throws invalid user input exception.	(1) Call mock authenticator attemptLogin with "User" and "Pass" (2) Add error to alert pane	(1) The correct username/passwor d was passed (2) An alert was added to the alert pane	

	(5) Register Button Trigger	1.2	(1) An instantiated Login Form Controller (2) A Mock Form registered with the Screen as "register"		Register button click	The app should show the registration form.	The app showed the registration form	
Authenticator	(1) Logging in without a username	1.1	(1) An instantiated authenticator with a mock DB handler	Call method attemptLogin with the input data	Username: Password: abcd1234 retrieveUserInfo should return an empty array	The authenticator should throw an Invalid User Input Exception with the message "Please enter in your username".	The authenticator threw an invalid user input exception with the appropriate image.	
	(2) Logging in without a password	1.1	(1) An instantiated authenticator with a mock DB handler.	Call method attemptLogin with the input data	Username: greg.james Password: retrieveUserInfo should return an empty array	The authenticator should throw an Invalid User Input Exception with the message "Please enter in your password".	The authenticator threw an Invalid User Input exception with the appropriate image.	
	(3) Logging in with a server connection issue.	1.1	(1) An instantiated authenticator with a mock DB handler.	Call method attemptLogin with the input data	Username: greg.james Password: abcd1234 retrieveUserInfo should return an empty array	The authenticator should throw a Server Connection Failed Exception.	The attempt login threw a server connection failed exception	
	(3) Logging in with valid user credentials	1.1	(1) An instantiated authenticator with a mock DB	Call method attemptLogin with the input data	Username: greg.james Password: abcd1234	The attemptLogin method should return false;	The attempt login method returned false;	

			handler and password hasher.		retrieveUserInfo should return a 2D string array with the row ["greg.james", "", ""] Password hasher should return true			
	(4) Logging in with invalid user credentials	1.1	(1) An instantiated authenticator with a mock DB handler and password hasher.	Call method attemptLogin with the input data	Username: greg.james Password: abcd1234 retrieveUserInfo should return a 2D string array with the row ["greg.james", "", ""] Password hasher should return false	The attemptLogin method should return false	The attempt login method returns false	
	(3) Logging in with valid user credentials with multiple users	1.1	(1) An instantiated authenticator with a mock DB handler and password hasher	Call method attemptLogin with the input data	Username: greg.james Password: abcd1234 retrieveUserInfo should return a 2D string array with the row ["fred.g", "", ""], ["jas.h", "", ""], ["greg.james", "", ""]	The attemptLogin method should return false	The attempt login method returns false	

					Password hasher should return false			
	(4) After a successful login, set the session up	1.1	(1) An instantiated authenticator with a mock DB and Session	Call Method login on authenticator	Username: Greg.james	The authenticator should call the session's method login with the parameter "Greg.james"	The authenticator correctly passed the value forward	
Register Form Controller	(1) Login Button Trigger	1.1	(1) An instantiated register form controller with a mock login button (2) A form registered with the screen as "login"	Trigger Button click.		The screen should switch the login app.	The screen switches the login app.	
	(1) Registering with invalid user input	1.2	(1) An instantiated register form controller with a mock register service, register button and input fields (2) A form registered with the screen as "login"	Fill data into appropriate fields and trigger register button click.	Forename: Greg Surname: James Username: Greg.james Email: g.james@test.co.uk Pass g.james@test.co.uk Password & Retype: abcd1234 The register method in registration throws an Invalid User Input Exception.	(1) The controller should call register and correctly pass the input data to the registration service. (2) The controller should add an alert to the alert pane to show the invalid user input.	The controller calls the registration service and passes the correct values, and also adds an error to the alert pane.	

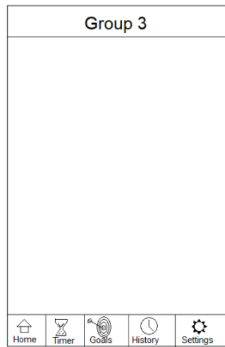
	(2) Registering but Server Connection Fails	1.2	(1) An instantiated register form controller with a mock register service, register button and input fields (2) A form registered with the screen as "login"	Fill data into appropriate fields and trigger register button click.	The register method in registration throws an Server Connection Exception		The registration service is unable to throw the needed exception because handling a server connection hasn't been implemented.	
	(3) Registering with valid data input	1.2	(1) An instantiated register form controller with a mock register service, register button and input fields (2) A form registered with the screen as "login"	Fill data into appropriate fields and trigger register button click.	Forename: Greg Surname: James Username: Greg.james Email: g.james@test.co.uk Password & Retype: abcd1234	(1) The controller should call register and correctly pass the input data to the registration service. (2) The controller should switch to the main page and add an alert to the login page.	The controller switched to the login page and displayed a success alert on the	
Registration	(1) Missing Fields	1.2	(1) An instantiated registration service with a mock DB Handler.	For every input variable, call the register method with the current variable blank.	Forename: Greg Surname: James Username: Greg.james Email: g.james@test.co.uk Password & Retype: abcd1234	The registration service should throw an Invalid User Input Exception with an appropriate message.	The register method threw the correct exception, with the message "Please enter in your X", where X was the missing field's name.	

	(2) Invalid Email	1.2	(1) An instantiated registration service with a mock DB Handler.	Call the register method of registration with the given input data.	Forename: Greg Surname: James Username: Greg.james Email: notanemailaddress Password & Retype: abcd1234	The registration service should throw an Invalid User Input Exception with a message to do with an invalid email address.	The register method threw an Invalid User Input Exception with an appropriate error message.	
	(3) Password mismatch	1.2	(1) An instantiated registration service with a mock DB Handler.	Call the register method with the given input data.	Forename: Greg Surname: James Username: Greg.james Email: g.james@test.co.uk Password: Abcd1234 Retype: abcd1234	The registration service should throw an Invalid User Input Exception saying the passwords don't match	The registration service threw an Invalid User Input Exception saying the passwords don't match	
	(5) Duplicate username	1.2	(1) An instantiated registration service with a mock DB Handler.	Call the register method with the given input data.	Forename: Greg Surname: James Username: Greg.james Email: g.james@test.co.uk Password: Abcd1234	The registration service should throw an Invalid User Input Exception saying the username is already being used.	The registration service threw an Invalid User Input Exception saying the username is already being used.	

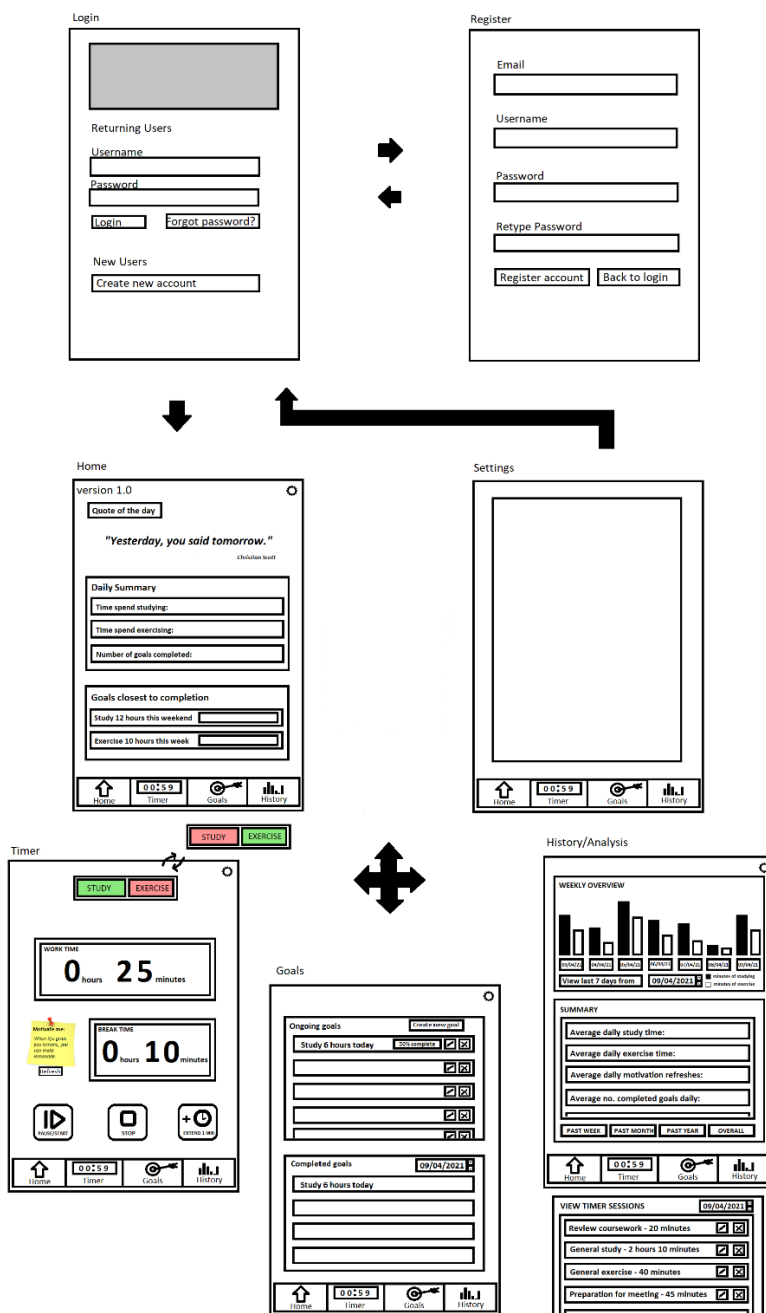
					Retype: abcd1234 DBHandler should return [[“Greg.james”, “”, “”]] when retrieveUserInfo is called. [[“Greg.james”, “”, “”]] when retrieveUserInfo is called.			
	(5) Duplicate email address	1.2	(1) An instantiated registration service with a mock DB Handler.	Call the register method with the given input data.	Forename: Greg Surname: James Username: Greg.james Email: g.james@test.co.uk Password: Abcd1234 Retype: abcd1234 DBHandler should return [[“”, “”, “g.james@test.co.uk”]] when retrieveUserInfo is called. [[“”, “”, “g.james@test.co.uk”]] when	The registration service should throw an Invalid User Input Exception saying the email address is already being used.	The registration service threw an Invalid User Input Exception saying the email address is already being used.	

					retrieveUserInfo is called.			
	(4) Valid data	1.2	(1) An instantiated registration service with a mock DB Handler.	Call the register method with the given input data.	Forename: Greg Surname: James Username: Greg.james Email: g.james@test.co.uk Password: Abcd1234 Retype: abcd1234	The method shouldn't throw any error.	The method didn't throw any kind of error.	
	(5) Failed to connect to server	1.2	(1) An instantiated registration service with a mock DB Handler.	Call the register method with the given input data.	Forename: Greg Surname: James Username: Greg.james Email: g.james@test.co.uk Password: Abcd1234 Retype: abcd1234 DBHandler should return a Server Connection Exception	The register method should throw a Server Connection Exception.	There is no way of detecting a server connection error from the DB Handler, so it wouldn't know when there was a server connection error.	

Appendix D (Initial GUI)



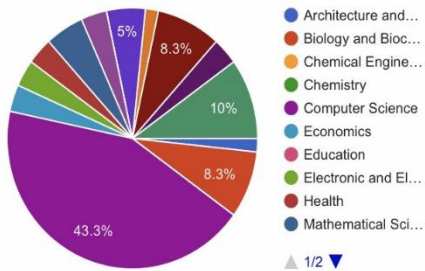
Appendix E (Reworked GUI)



Appendix F (Survey results)

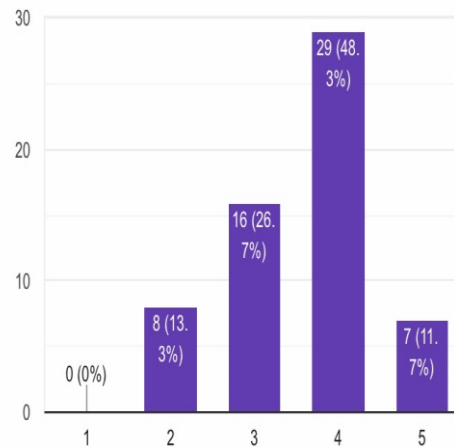
Which department are you in?

60 responses



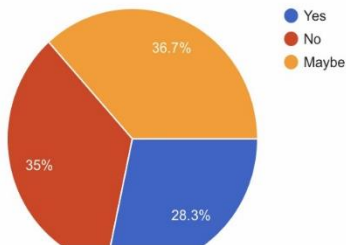
Do you feel a self-tracking app will make you feel in control over some aspects of your life?

60 responses



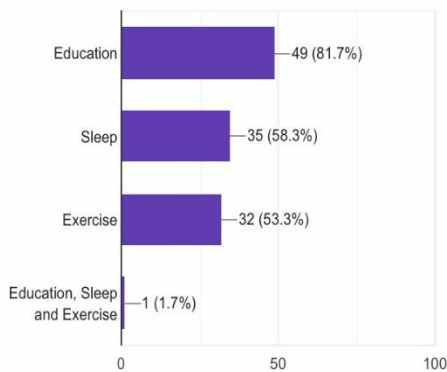
Are you able to manage your time efficiently, on a daily basis?

60 responses



Which of the following features would you like to see in an app? (Select one or more boxes.)

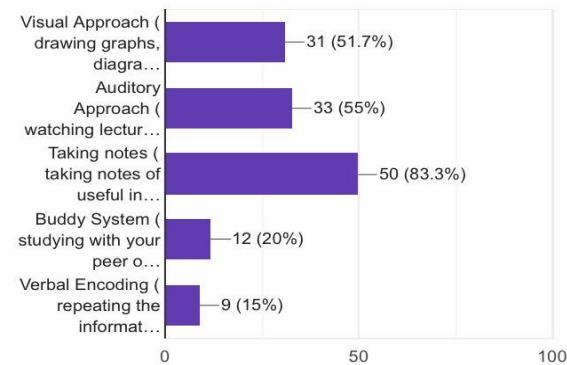
60 responses



Education

What are your methods of study?

60 responses



How often do you set learning and performance goals?



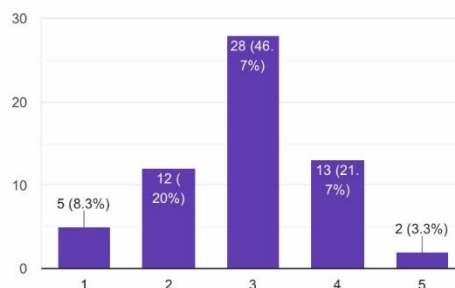
60 responses



How satisfied are you with your current method of study?



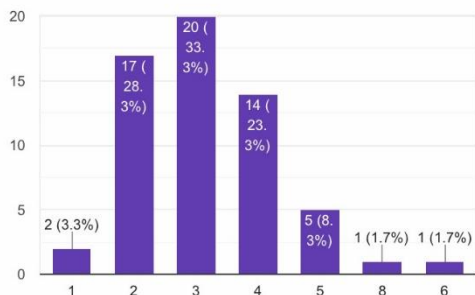
60 responses



Do you feel your productivity is affected after taking a break?



60 responses

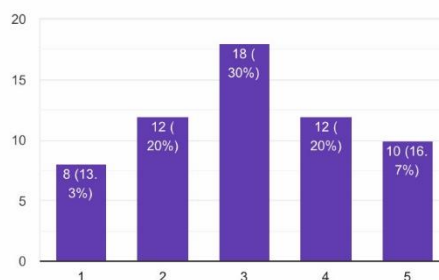


Exercise

How active are you weekly?

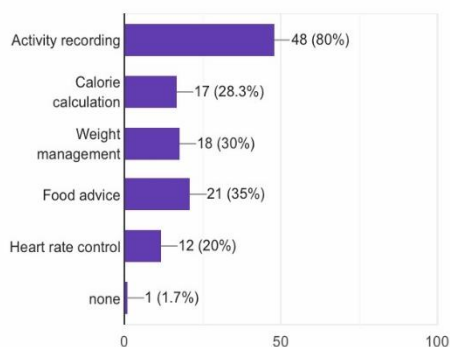


60 responses



What are the main functions that you look in an exercise app?

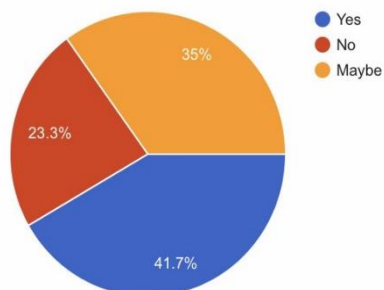
60 responses



Sleep

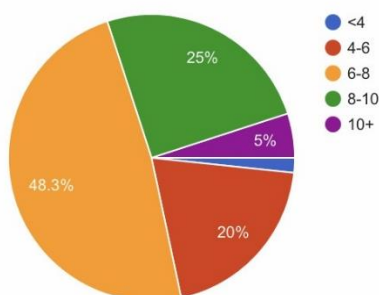
Are you getting enough sleep?

60 responses



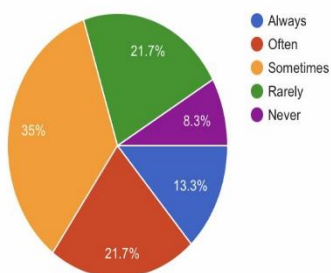
How many hours of sleep are you getting on a day?

60 responses



How often do you feel sleepy during the day, while doing your tasks?

60 responses



Appendix G (Password Hashes)

Reference	Salt
A	EhriHSffB8XrgxI8LRGClb8tt
B	EhriHSffB8XrgxI8LRGClb8tq
C	K5s7HJKhkbaHGK678HKJghsdp

Unencrypted	Hash
""	\$1\$EhriHSffB8XrgxI8LRGClb8tt2Z5m8XyxAKiJ3HNDgCDAJyJrogGGu/AkRcuNsUV12k8=
"1\$1\$1\$1\$"	\$1\$EhriHSffB8XrgxI8LRGClb8ttZseiKukpHeHWe3wE7lLNiLS7YoqYiQQUHOUBN1trAAM=
"\$1\$1\$1\$1\$1\$1\$"	\$1\$EhriHSffB8XrgxI8LRGClb8ttblnNE1CXlyONUPfQMDkSfJ105uGYnJZyq3UOAGT9Qwg=
"1\$1\$1\$1\$1\$1\$"	\$1\$EhriHSffB8XrgxI8LRGClb8ttMDes6LZK+Lzl/8b2RfKHQjfoODuWfIOCvJqWXlIj19Y=
"this is a really long password that you would never use because you would likely forget it"	\$1\$EhriHSffB8XrgxI8LRGClb8tt9T9LfKA/d9AxPlITtAg5eQwc8T2NcP0c226+KmFQgRE=
"!\$^%GJHE87"	\$1\$EhriHSffB8XrgxI8LRGClb8tt7VaSzdrk9LMYb9byuihi5GP1reYJvaZ4r5OTkeYkKlQ=