Dissertation

Mental Workload App

Submitted May 2021, in partial fulfilment of the conditions for the award of the degree **BSc Hons Computer Science with Year in Industry**.

Student ID: 4299117
School of Computer Science
University of Nottingham

I hereby declare that this dissertation is all my own work, except as indicated in the text:

Signature: ***I.D.***

May 10, 2021

4299117

Supervisor: Max Wilson
Module Code: COMP3003

2021/05

1 Abstract

This dissertation project is based around the visualisation and understanding of cognitive data; primarily mental workload. Ahead of this project, a team of researchers have collected a large dataset of people's mental workload rating hour to hour, along with data regarding their sleep and screen time, all recorded over the course of one week.

The collected data has been used as the basis for an application to visualise this data in a user friendly and convenient format, which would allow someone to understand their mental workload data and set goals towards it. This is an app created using the React Native framework, deployable to both Android and iOS.

Contents

2 Introduction

2.1 Background and Motivation

Current research is able to estimate mental workload levels from various behavioural and physiological systems, such as the NASA Task Load Index[1], Short Stress State Questionnaire[2] or Instantaneous Self Assessment of Workload[3], but we don't know how that type of data is useful to people and how it should be visualised. This project aims to do exactly that (discussed properly in section 2: Aims and Objectives).

With the proliferation of wearable technology, it seems imminent that our smartwatches can soon keep track of our mental activity as well as our physical activity[4]. This is also part of the reason behind the interest in this project and the motivation behind it, as this is being shown to be a real-world problem. The research team collaborating with this project are working towards accurately measuring mental workload using physiological sensors; in the meantime, we are beginning to explore how this data could be presented and visualised conveniently to the user, in a way similar to step counters and bodyweight tracking.

As humans, we rely on healthy amounts of mental workload and stress to be productive, as it pushes us to get our work done. While we can estimate these mental workload levels, it is quite difficult to figure out the optimal amounts of stress and mental workload for any given person, as this 'healthy' amount differs person-to-person; some people need higher amounts of stress to perform optimally whereas others require much less and would burnout at those high stress levels. The best we can do is to start with some baseline amount and analyse how effective it is, thus making changes depending on how each individual responds.

A simple and useful method of perceived workload estimation is the Instantaneous Self Assessment (ISA). Originally developed for use in air traffic control; this method gives an immediate rating of the person's subjective workload at any given time. The way this works involves having the participant record their perceived mental workload at regular time intervals, on a scale of 1 (very low) to 5 (excessive).

2.2 Aims and Objectives

The aim of this project is to design and develop a mobile application that allows people to record and visualise their cognitive activity/function as a form of personal data[4]. The app will utilise sample data that has been recorded by some researchers for a PhD project; this will allow for development and analysis to begin without the need to immediately implement a data entry system.

The key objectives of this project are:

1. Produce literature investigating various factors affecting ones cognitive function (i.e. read through currently existing literature to gain a foundation of knowledge).
2. Analyse the aforementioned factors myself/with the research team to draw conclusions based on the given data.
3. Deploy template app structure in React Native
4. Construct some methods which will allow a computer program to visualise such data in a convenient and user-friendly format
5. Build a prototype for the app which proves the validity of the aim of this project.

6. Produce design(s) for the app and discuss with the research team/outside opinions to receive feedback.

7. Deploy an application that puts these all together into a system allowing the user to visualise their cognitive data.

3 Related Work

3.1 Broader Context

The wider context of related work behind this project is that of personal informatics which is a class of systems that help people collect personal information to improve self-knowledge[5]; more specifically the system relating to this project is that of the quantified self, which is any individual engaged in the self-tracking of any kind of biological, physical, behavioural, or environmental information[6]. Current technology utilises the concept of the quantified self (or self-tracking) in a physical sense where we see wearable watches tracking physical activity such as steps taken, average heartbeat rate, calories burned and other such data. New emerging technology is beginning to allow the quantification of emotional and cognitive changes, with certain projects within the field making strong headway, notably SpireHealth[1] and Sentio Solutions[2] tackling not only the collection and quantification of said cognitive data but also the human-computer interaction involved to display such data effectively.

Furthermore, research also exists to estimate stress and mental performance along with the role of eye fixation under such conditions[7]; more relevant research exists which uses physiological measures such as fNIRS (Functional near-infrared spectroscopy) to estimate and quantify mental workload[8]. This technique for estimating mental workload is promising as, in concurrency with EEG (Electroencephalography), this provides more accurate results than other individual modalities. As explained in [8], fNIRS works by measuring the changes in cerebral blood flow and related haemoglobin concentrations through near-infrared light source/detectors on the scalp, this technology is also used by the research team with this project.

In the context of everyday life, research has been conducted which monitors mental workload levels in an office-work scenario[9] with data from a mobile ECG logger. ECG stands for electrocardiogram, which is another physiological measure that can be used to measure mental workload through attaching sensors to the skin[10]; these sensors are used to detect the electrical signals produced by your heart each time it beats, thus checking your heart's rhythm and electrical activity. The research conducted recorded ECG data alongside collecting subjective workload ratings using the NASA-TLX[1] (discussed section 3.3) and reported a high success rate for predicting mental workload levels of these subjects. This shows plenty of promise in this field of research and the possibility of cognitive activity tracking becoming mainstream in the same way physical activity tracking is.

Interestingly, ECG also expresses cardiac features that are unique to the individual[11], which indicates that such cognitive activity tracking can be truly tailored to any given person along with the possible use of it identifying individuals.

3.2 What is Mental Workload?

There are no unanimously agreed upon definitions of mental workload, as it is often used without definition and as a term used depending on context. It can be described as *"the relationship*

[1]https://www.spirehealth.com/
[2]https://www.myfeel.co/

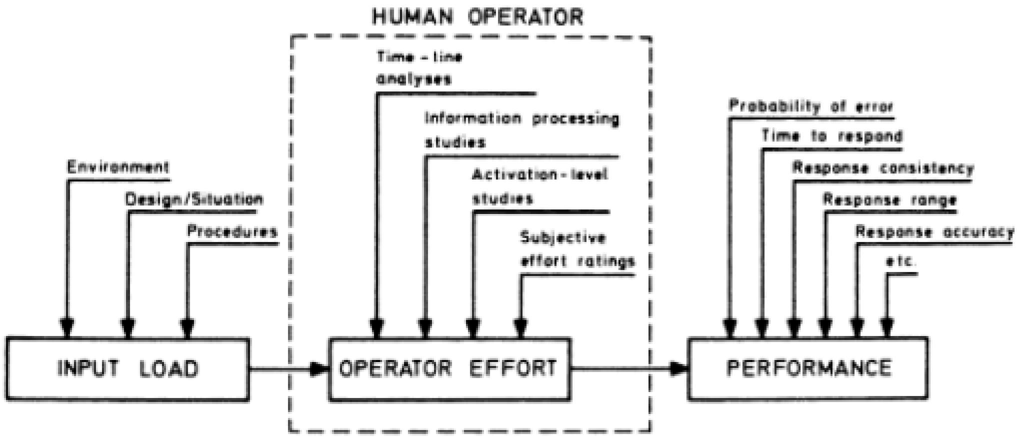


Figure 1: Workload Attributes

between primary task performance and the resources demanded by the primary task"[12]. Within the context of a human operator, the broad area of workload can be defined as being divided into three related attributes: input load, operator (human) effort, performance (Figure 1)[13]:

1. **Input Load:** This is anything taken external to the operator which affects the effort and output of the task (noise, temperature, crew station layout, instructions, task sequencing..)
2. **Operator Effort:** This concerns any factors or events internal to the operator (a direct relationship exists between this and the input load along with the operator's physical state)
3. **Performance:** This is any meaningful output produced by the operator.

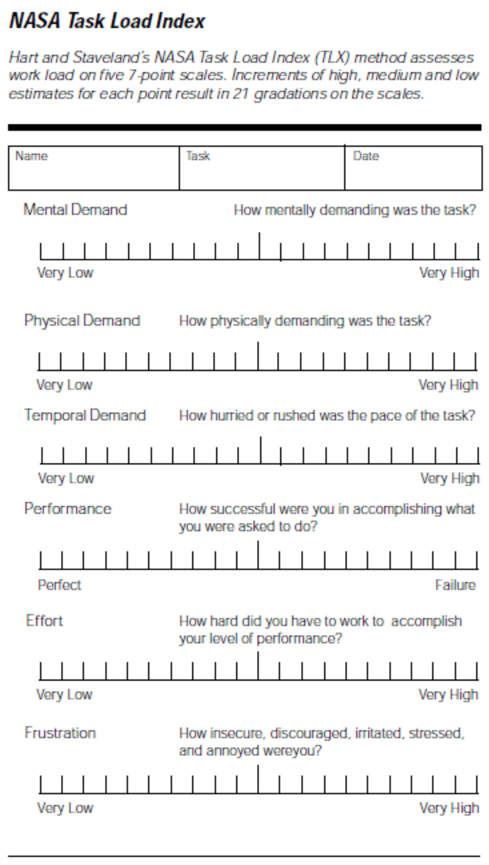
These three attributes link to the context of mental workload within this project as the operator is said to be the user/participant, with the input load being a result of their external environments such as noise around them, temperature, state of their workstation etc., with the operator effort being a function of the input load and the psycho-physical state of the participant (general background, personality, mental well-being).

As discussed in section 2.1, we need healthy levels of mental workload to drive us productively. However, performance can suffer if the demands are too low or too high. This links quite well with stress which, albeit different to mental workload, is connected as short term stress can be beneficial to overcome challenges or dangerous situations[14].

3.3 How do People Measure/Self-Describe Mental Workload?

A range of techniques exist that allow us to capture insights into how people work, those most relevant to this project being self-reporting methods such as NASA Task Load Index (NASA-TLX)[1], Short Stress State Questionnaire (SSSQ)[2] and Instantaneous Self-Assessment of Workload (ISA)[3].

NASA-TLX is *"a subjective workload assessment tool to allow users to perform subjective workload assessments on operator(s) working with various human-machine interface systems. Originally developed as a paper and pencil questionnaire by NASA Ames Research Center's (ARC) Sandra Hart in the 1980s, NASA-TLX has become the gold standard for measuring subjective workload across a wide range of applications."*[15]. Applying this consists of dividing the workload into six different scales, those being:


NASA Task Load Index

Hart and Staveland's NASA Task Load Index (TLX) method assesses work load on five 7-point scales. Increments of high, medium and low estimates for each point result in 21 gradations on the scales.

Name	Task	Date

Mental Demand — How mentally demanding was the task?

Very Low — Very High

Physical Demand — How physically demanding was the task?

Very Low — Very High

Temporal Demand — How hurried or rushed was the pace of the task?

Very Low — Very High

Performance — How successful were you in accomplishing what you were asked to do?

Perfect — Failure

Effort — How hard did you have to work to accomplish your level of performance?

Very Low — Very High

Frustration — How insecure, discouraged, irritated, stressed, and annoyed wereyou?

Very Low — Very High

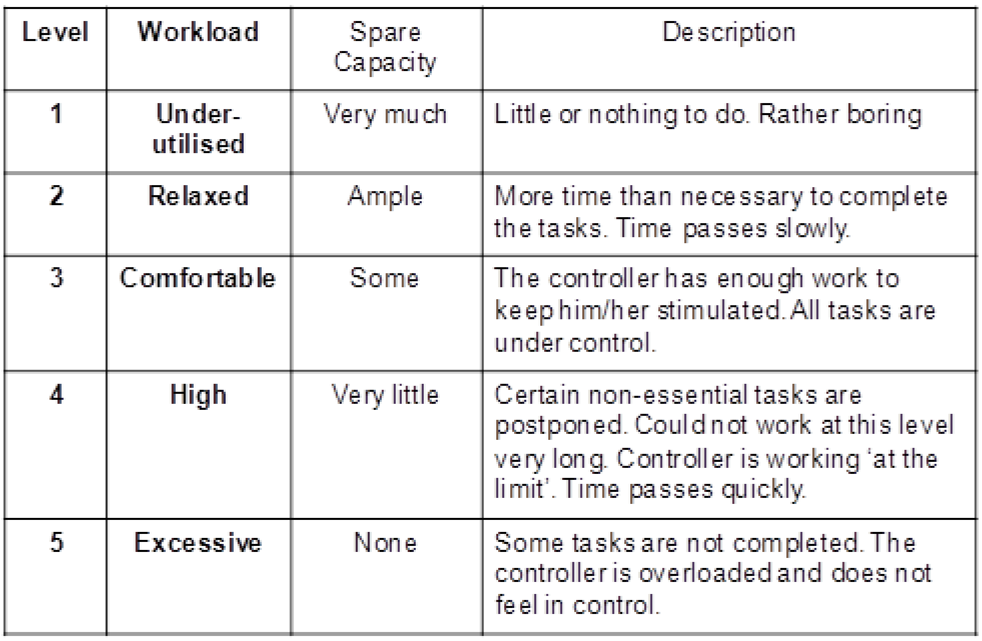
Figure 2: NASA-TLX Score Sheet

1. Mental Demand
2. Physical Demand
3. Temporal Demand
4. Performance
5. Effort
6. Frustration

After the subject has made a rating on all 6 scales, an individual weighting is applied to each scale based on their perceived importance to the subject (depending on the task at hand). This weighting is then multiplied by the score and divided by 15 to end up with a workload score between 0-100.

ISA is a measurement method which *"consists of a 5-point workload rating scale represented by 5 buttons usually located on an 'ISA Box'. ISA was originally conceived at the National Air Traffic Service (NATS) in Bournemouth where it has been used to evaluate Air Traffic Controller workload."*[16]. As discussed in section 1, this is a simple method that allows for an immediate rating of a person's subjective workload at any given time through recording a rating of their perceived workload on a scale of 1-5 at regular intervals.

This can come in very useful as the ratings can be compared across multiple tasks, as the question ("what is your current mental workload level?") remains the same, and the scale (1-5)



Level	Workload	Spare Capacity	Description
1	**Under-utilised**	Very much	Little or nothing to do. Rather boring
2	**Relaxed**	Ample	More time than necessary to complete the tasks. Time passes slowly.
3	**Comfortable**	Some	The controller has enough work to keep him/her stimulated. All tasks are under control.
4	**High**	Very little	Certain non-essential tasks are postponed. Could not work at this level very long. Controller is working 'at the limit'. Time passes quickly.
5	**Excessive**	None	Some tasks are not completed. The controller is overloaded and does not feel in control.

Figure 3: ISA Workload Table [3]

is also the same. The ratings could then be compared to the task at hand and the subject can gain a better understanding of how different tasks/types of tasks affect their stress and productivity.

While the subjective measurements are particularly useful in measuring mental workload without the need for any equipment, the use of equipment such as ECG[9][17] and fNIRS[8] can allow us to work with quantifiable information which is objective, as opposed to subjective. This can take some stress and extra work away from the user/participant if intended to be used commercially. Measuring mental workload objectively could involve collecting data regarding the oxygenation levels in the brain[18] which corresponds to the amount of neural activity. Low oxygenation (less neural activity) can indicate lower mental workload if this is in conjunction with a low task load; however low oxygenation for a high task load comes with a drop in performance and therefore is consistent with disengagement from the task at hand, which can show that too high a mental workload has been measured and has a detrimental effect.

4 Description of the work

This project works on the assumption of the technology discussed in section3 already existing and said data feeding into the app for processing and visualisation. My system is based on ISA data, as it is a rapid and simple technique, which is non-intrusive and easy to utilise, compared to NASA-TLX which has many parts to it and can be quite intrusive to the subject if asked to record regularly. The actual data currently being fed into the system is ISA data and automatically captured screen time data gathered by one of the researchers from the team in advance of this project.

The idea of future wearables has been kept in mind during the development of this project, as in the future it is expected that data from these wearables would have the capacity to be input into the app automatically, similar to how current smartwatches function.

4.1 Requirements

4.1.1 Functional Requirements

1. The user can select a date and have the application display all ISA data related to that day in a graph.
2. The user can select a second date to view ISA data in direct comparison to the first.
3. The user can select a date and have the application display all screen time data related to that day in a graph.
4. The user can select a date range and have the application display all sleep data related to that range in a graph.
5. Developer/Researcher can manually add data from within the app and add generated example data.

4.1.2 Non-Functional Requirements

1. All data must be stored locally, kept on the user's device, and deleted should the application be deleted.
2. Data must be validated before being stored.
3. The UI must be consistent across platforms.

4.2 Given Dataset

To build this project, I have been given a dataset to work with from the research team. The dataset includes data gathered across a week from various participants, detailing a personality test, their sleep, screen time, subjective mental workload ratings, alcohol and food intake, fatigue and stress scales among other such information. The mental workload, sleep and screen time data were used in this project.

4.2.1 Mental Workload Data

Mental workload data was presented in the format of a .csv file with 8 columns containing the following data:

1. Date and Time
2. Question 1: What is your current mental workload level?
3. Rating 1 (from the question above)
4. Question 2: What has your overall mental workload level been since your last rating?
5. Rating 2 (from the question above)
6. Question 3: What would you rate your overall performance since the last rating?
7. Rating 3 (from the question above)
8. Summary

This data has been filled in by the participant anywhere between 1-2x per hour, giving plenty of information regarding the daily fluctuations of mental workload depending on the task. Despite all the data in this file, only 3 columns have been used and implemented within the application: date/time, current mental workload level and summary of the task. This is to minimise 'fluff' data that does not get used by the user or visualised in any effective manner.

4.2.2 Screen Time Data

Screen time data was also presented in the format of a .csv, this one with 7 columns containing the following data:

1. Date
2. Name
3. Group
4. Interval (e.g. 10:00:00-10:05:00)
5. Application
6. Productivity (category)
7. Time

This data has been automatically captured by a piece of software so every application has been logged. The columns being used were date, interval, application, productivity and time. Those are the most useful columns, which allow for effective visualisations to be created showing what applications have been run at what times, all categorised my productivity (productive, neutral, unproductive).

4.2.3 Sleep Data

Sleep data has been presented in the form of a .docx sleep diary with the following questions answered by the participant every morning:

1. At what time did you go to bed last night?
2. After settling down, how long did it take you to fall asleep?
3. After falling asleep, how many times did you wake up in the night?
4. After falling asleep, for how long were you awake during the night in total?
5. At what time did you finally wake up?
6. At what time did you get up?
7. How long did you spend in bed last night (from first getting in, to finally getting up)
8. How would you rate the quality of your sleep last night? (1=v poor, 5=v good)

Questions 2, 3, 7 and 8 were used, as well as the date of each night; the other questions, while useful, only added more bloat given the aim of the project is to represent such data in the form of an app.

5 Methodology

5.1 Project Management

I have approached this project with an agile methodology in mind as there are plenty of possible features which will be developed beyond the initial MVP; having incremental improvements will allow the app to develop naturally with clear milestones to hit and features to work on.

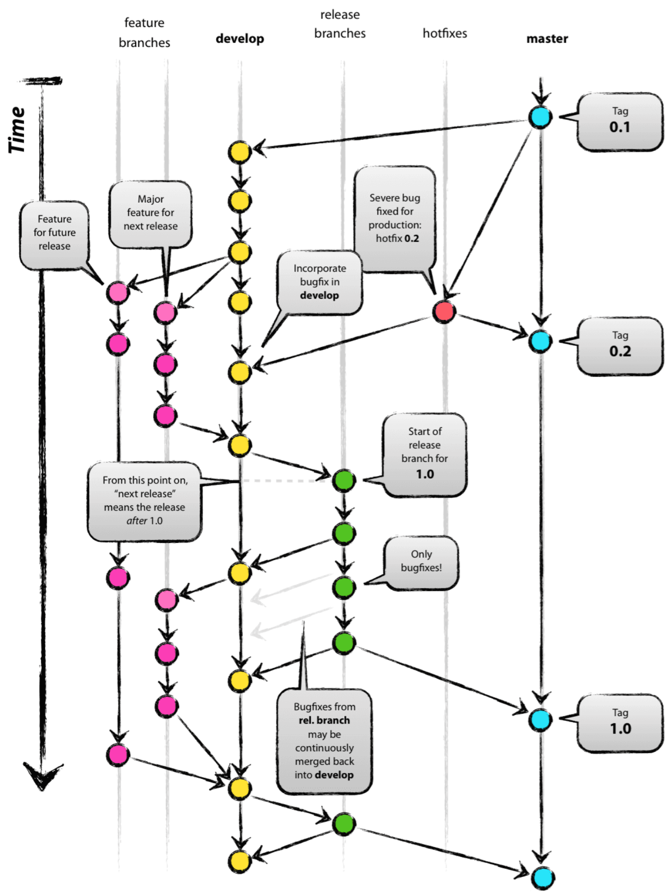

feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Severe bug fixed for production: hotfix 0.2
Major feature for next release
Feature for future release
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Tag 1.0

Figure 4: Visualisation of the 'Git Flow' workflow from:[19]

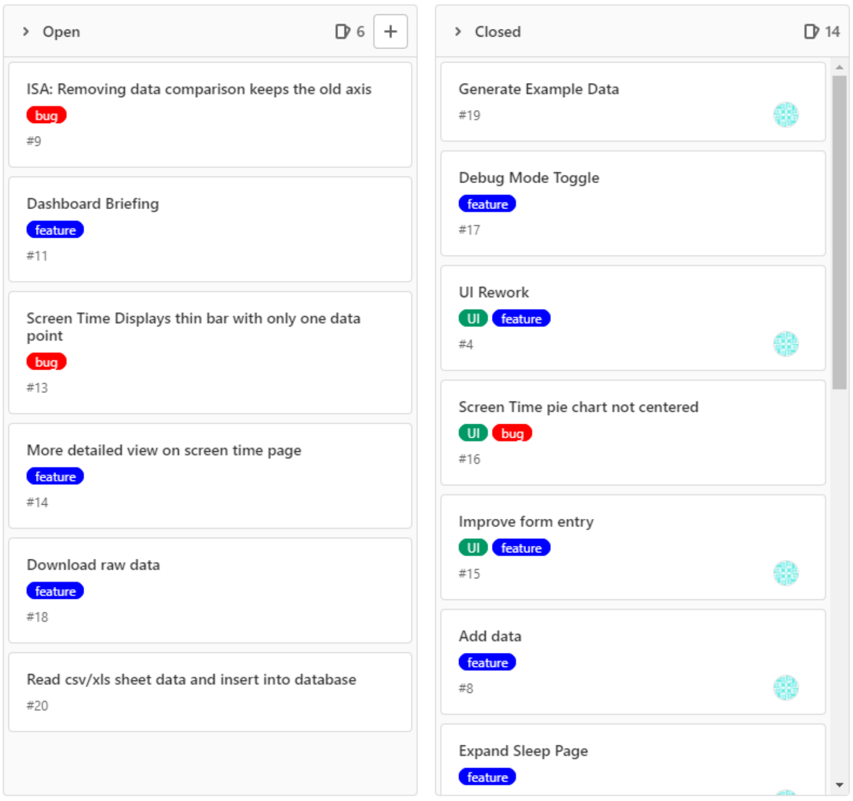
5.1.1 Version Control/Workflow

As described in the proposal I am using git for version control, specifically on GitHub, with the use of "Git Flow"[19] workflow. These are the branches being applied to the project:

- **Master**: This branch will be used to release tagged versions of the project for use in the interim report and discussion with the client team.
- **Develop**: This is the branch from which all the features are branched from/merge to; it should generally be working but expect to find bugs etc in here.
- **Feature**: This branch will be for a singular feature to be worked on; this would branch from and merge to the develop branch.
- **Release**: A release branch is branched from develop, and represents an (almost) fully complete version of the project, any minor bug fixes before release are to be fixed here (along with any metadata changes) after which the branch is merged back to develop and master.
- **Hotfix**: I won't be using the hotfix branch as, due to the nature of this project, there won't be any live users requiring immediate bug fixes in production.

Application of the Kanban process has been made simple due to the easy integration of this process within GitLab. I have created a Kanban board with 3 states: Todo, In Progress, Done. A newly created issue goes directly to the 'Open' column, where it can also be assigned to a milestone (i.e. Basic Architecture) and label(s) such as 'feature', 'bug' etc. Once a pull request has been merged, it automatically moves to 'Closed'. These 2 columns are shown in Figure 5.

Leading on from this, 3 milestones have been created, 'Basic Architecture', 'Core Functionalities' and 'Expanded Functionalities'. These milestones have their due dates which, along



Open 6

ISA: Removing data comparison keeps the old axis
bug
#9

Dashboard Briefing
feature
#11

Screen Time Displays thin bar with only one data point
bug
#13

More detailed view on screen time page
feature
#14

Download raw data
feature
#18

Read csv/xls sheet data and insert into database
#20

Closed 14

Generate Example Data
#19

Debug Mode Toggle
feature
#17

UI Rework
UI feature
#4

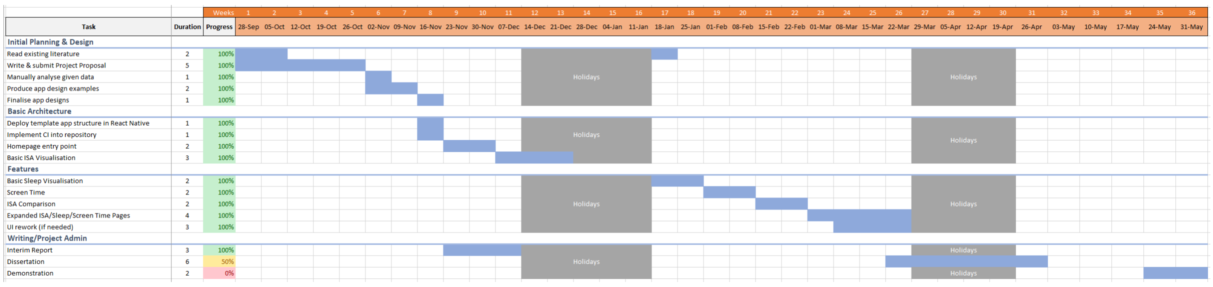
Screen Time pie chart not centered
UI bug
#16

Improve form entry
UI feature
#15

Add data
feature
#8

Expand Sleep Page
feature

Figure 5: Kanban Board



Task	Duration	Progress
Initial Planning & Design		
Read existing literature	2	100%
Write & submit Project Proposal	5	100%
Manually analyse given data	1	100%
Produce app design examples	2	100%
Finalise app designs	1	100%
Basic Architecture		
Deploy template app structure in React Native	1	100%
Implement CI into repository	1	100%
Homepage entry point	2	100%
Basic ISA Visualisation	3	100%
Features		
Basic Sleep Visualisation	2	100%
Screen Time	2	100%
ISA Comparison	2	100%
Expanded ISA/Sleep/Screen Time Pages	4	100%
UI rework (if needed)	3	100%
Writing/Project Admin		
Interim Report	3	100%
Dissertation	6	50%
Demonstration	2	0%

Figure 6: Gantt chart

with the Gantt Chart (Figure 6), provide a reference point for development speed/progress.

A total of 92 commits has been made across 12 different branches; unfortunately, GitLab does not have proper visualisations to show all the commits and the branches they were made to.

6 Design

The user interface aims to provide a clean and intuitive feel to the user with minimal bloat to it as possible. The initial design idea (Figure 7), however, felt too cluttered and had far too many unnecessary parts for the app to look clean. Figure 8 shows the application front page, which has been kept simple and to the point with custom buttons, allowing for navigation where necessary.

Buttons have been placed on-screen instead of in the header/footer as the dashboard does not have enough content on it to justify relocating navigation components away. This can be

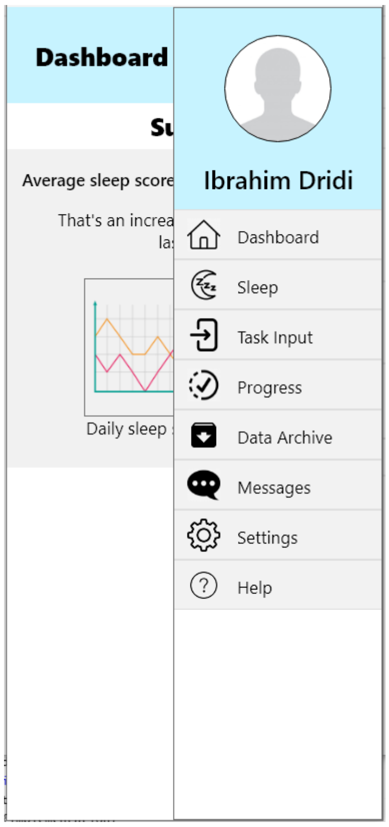


Figure 7: Initial Design Idea

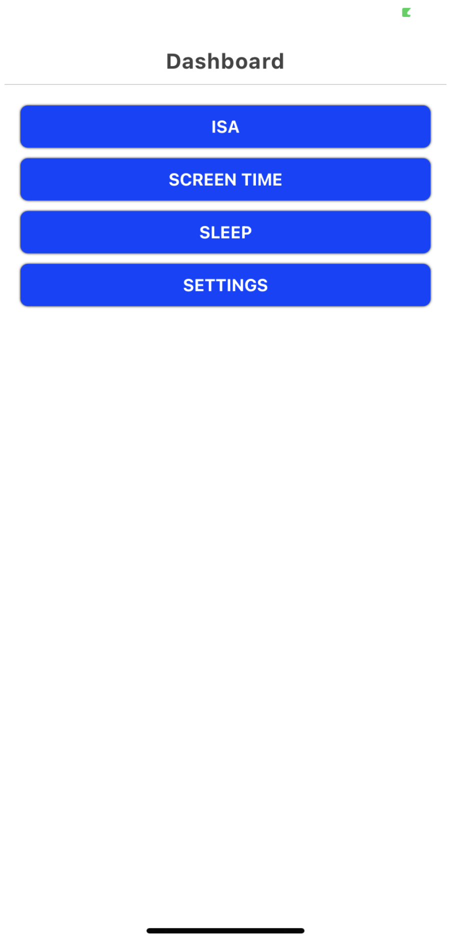


Figure 8: App Dashboard

subject to change in the future of the project if a briefing section is implemented on the dashboard.

Leading on from the buttons, they are custom components I have created for the application to keep the design uniform across platforms and to give the app a consistent theme throughout. This is discussed further in section 7.6.

Graphs and sections of each page have been separated to display on stylised 'cards' (see Figure 16 as an example) to make information on-screen more visually pleasing and easy to look at. The graphs themselves have had horizontal scrolling implemented within them (see Figure 14 as an example) so all data are evenly and appropriately spaced apart, regardless of the number of data points. They scroll within the constraints of the card for a clean and uniform look.

7 Implementation

7.1 Application Framework

The mobile application is being written in the React Native framework, more specifically using the Expo[3] interface as this simplifies setup by handling many configuration steps and allows you

[3]https://expo.io/

to test on a real device within the Expo app. As well as this, native iOS/Android features are still available through the SDK; if any extra flexibility is needed with these native components, it is a simple case of ejecting from expo to access the underlying Android/iOS projects within; this just takes one command from the terminal.

7.2 Storage/Backend

In line with the requirements, all application data is stored locally in an SQLite database using the expo-sqlite[4] library. The database consists of 3 tables:

1. **isa** - This table stores ISA data in the rows named: ***dateTime*** (date and time), ***workloadRating*** (ISA rating) and ***summary.***
2. **sleep** - This table stores sleep data in the rows named: ***date*** (date of night slept), ***hoursInBed*** (total time spent in bed), ***hoursUntilSleep*** (time taken from getting into bed to being asleep (in hours)), ***timesWokenUp*** (number of times woken up throughout the night) and ***sleepQuality*** (subjective sleep quality rating 1-5).
3. **screenTime** - This table stores screen time data in the rows named: ***name*** (program name), ***date*** (date of entry), ***interval*** (which half-hour time interval this entry resides in), ***time*** (time spent on the program (in seconds)), ***category*** (category of the program: productive, neutral, unproductive).

I have implemented the database functionality by creating a database helper class (dbHelper) which can be imported by the components that need it; this helps to abstract the specific workings of the database from individual components and separates the responsibility of querying the database/retrieving data and displaying such data. The dbHelper class has functions to query the database and perform various tasks surrounding it:

1. **initDB()** - This function initialises the database by creating all 3 tables if they don't exist, and is called every time the application is run.
2. **exampleData()** - This function populates the database with example data to be used for development/researcher purposes.
3. **getISAData(date, xAxis)** - This function retrieves ISA data to be displayed on a graph; more specifically the workload rating and date-time are selected from the database for all rows which match the date selected by the user. The time and workload ratings are then pushed to an array of dictionaries, and all unique times are pushed to the x-axis array for use by the graphing library
4. **getScreenTimeCondensed(date)** - This function selects all screen time data and condenses all the information to return productive, neutral and unproductive time spent for each interval, along with total screen time and total productive/neutral/unproductive screen time.
5. **getSleepData(fromDate, toDate)** - This function retrieves all sleep data between two dates and returns the information for each row (hours in bed, hours until sleep, times woken up, sleep quality) in an array of dictionaries where each dictionary has a value for the data and the row information.
6. **insert functions** - there are 3 insert functions, and they all insert given data into their relevant table.

[4]https://docs.expo.io/versions/latest/sdk/sqlite/

7.3 Navigation

I implemented stack navigation[20] to traverse through screens. This works by pushing new screens onto the stack and popping them off the stack to go back to the previous page.

My navigation implementation uses the react-navigation-stack library. More specifically, I have a component named ‘homeStack’ which creates a stack navigator naming each screen in the application along with any information regarding the screen. The navigator component from this home stack is then imported by the default file ‘App.js’ and is rendered as the only component. Given the dashboard page is the first component in the stack navigator, this is used as the landing page. The following are the screens:

- **Dashboard** - Landing page/dashboard (Figure 9).
- **ISA** - Select one/two dates to view/compare mental workload ratings throughout the day (Figure 14).
- **ScreenTime** - Select a date to see screen time data, how much productive/neutral/unproductive time was spent on your screen for a given day (Figure 16).
- **Sleep** - Select a date range to view/compare sleep data such as hours spent in bed, hours until sleep, times woken up and sleep quality rating (Figure 18&19).
- **Settings** - Settings page with developer options allowing to enable debug mode for adding data, and data deletion (Figure 10).
- **AddData** - Select which data you wish to add.
- **AddISA** - Submit ISA data to the database (Figure 11).
- **AddScreenTime** - Submit screen time data to the database (Figure 12).
- **AddSleep** - Submit sleep data to the database (Figure 13).

7.4 Graphing

The first choice for a graphing library was react-native-chart-kit[5] for the (mostly) simple integration into the project. This library has minimal setup and allows for good looking charts without the need for much customisation. Towards the end of the ‘Screen Time’ page implementation, as I was struggling to display the necessary information properly, I performed some further research and found the victory-native[6] graphing library which suited the needs of the project much more; more customisation was available beyond just displaying basic information on a single graph. While extra work is needed to make graphs look good with this library, it is worth the manual styling as there is a much wider range of charts available to create, along with higher customisation with each chart.

7.4.1 ISA Graph

After the user has selected a date, the database is queried using the **getISAData()** function in the database helper class to retrieve the workload ratings and date/times for a given day; once the user has picked a date they are also given the option to pick a second date, the data of which displays on the same graph in a different colour to allow for comparison(Figure 14).

[5]https://www.npmjs.com/package/react-native-chart-kit
[6]https://formidable.com/open-source/victory/docs/native/

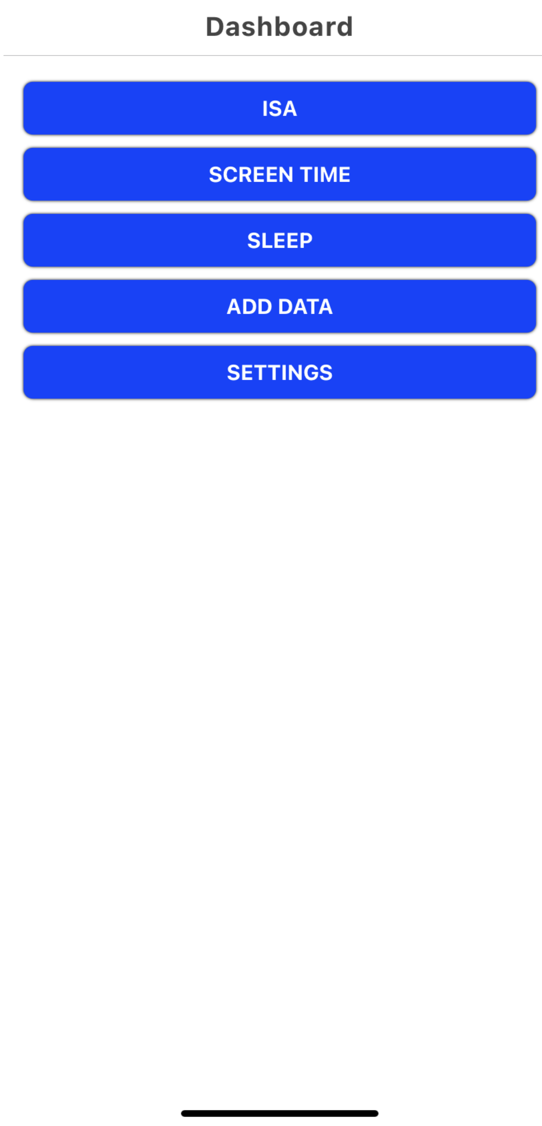
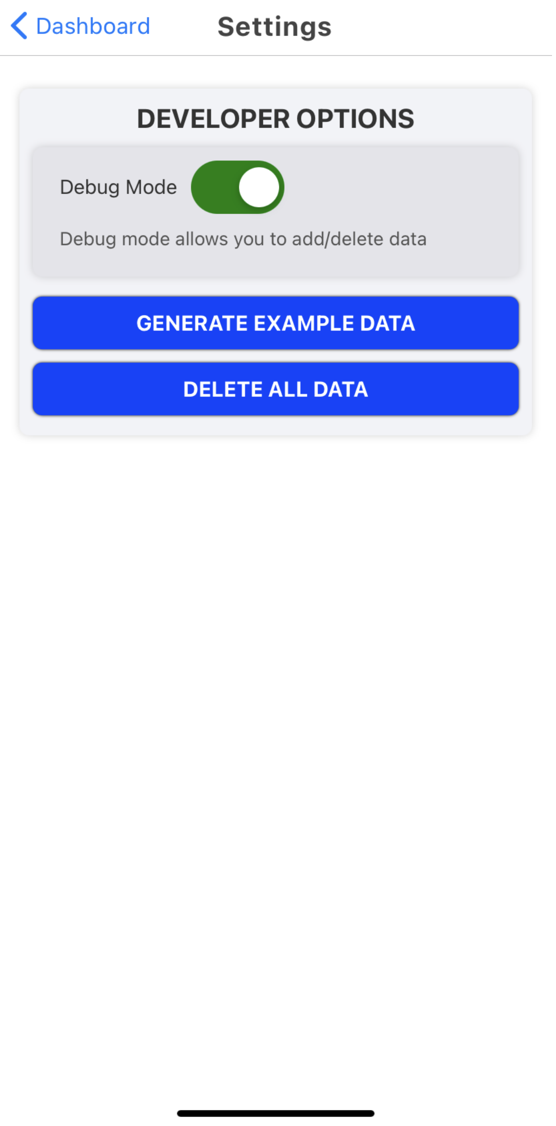


Figure 9: Dashboard Page



Debug Mode: on

Debug mode allows you to add/delete data

Figure 10: Settings Page

(Figure 15) The individual charts (Figure 14) are created by encapsulating a 'VictoryLine' and 'VictoryScatter' component in a 'VictoryGroup' component holding the data for the chart; this draws a line chart and scatter plot on the same graph with the same data to show both the lines and the dots of the individual data points. These components are also encapsulated within one 'VictoryChart' component which allows us to set various information regarding the graph as a whole (height, width, legend, axis values). To compare against another day's data, another 'VictoryGroup'/'VictoryLine'/'VictoryScatter' component is created holding the data of the second date; this is also encapsulated by the same 'VictoryChart' as the first graph, meaning both components display on the same chart and same axis, thus allowing for easy comparison.

7.4.2 Screen Time Graph

The user selects a date, after which the database is queried using the **getScreenTimeCondensed()** function in the database helper class, pulling all the screen time data for the day

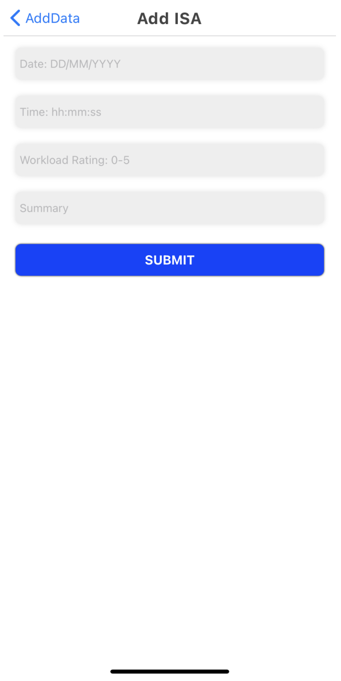


Figure 11: Add ISA Page

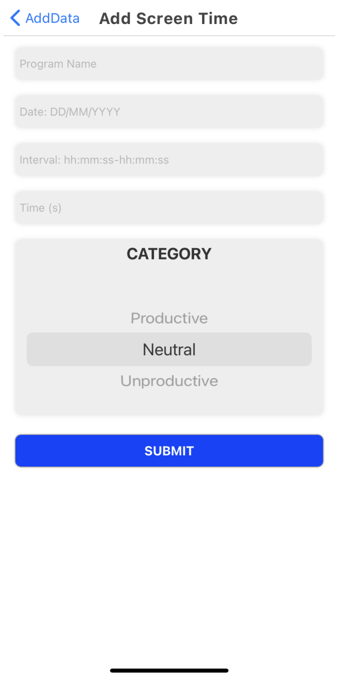


Figure 12: Add Screen Time Page

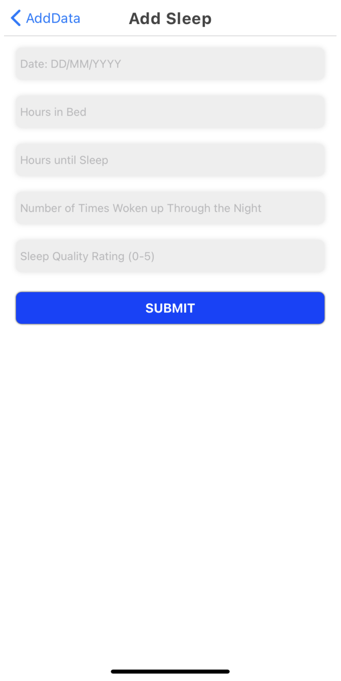

AddData
Add Sleep
Date: DD/MM/YYYY
Hours in Bed
Hours until Sleep
Number of Times Woken up Through the Night
Sleep Quality Rating (0-5)
SUBMIT

Figure 13: Add Sleep Page

in a convenient format for the graph (Figure 16) to display (integers, and arrays of dictionaries).

(Figure 17) The graph is displayed by creating 3 'VictoryBar' components which, for all time intervals recorded, display a bar for all the productive, neutral and unproductive data respectively. These 3 are encapsulated in a 'VictoryStack' component which stacks all bars with the same x-axis value (time interval) together. Again, this is also then encapsulated within a 'VictoryChart' component which holds chart information.

7.4.3 Sleep Graph

After selecting a date range, the database helper class function **getSleepData()** is called to retrieve all the rows from the sleep table given the date resides within the provided range. The data from the rows are then pushed into dictionary arrays containing the date and data.

Two graphs (Figure 18&19) are displayed on this page rather than one (discussed in Visualization Effectiveness section) to better separate the data. The first graph displays the hours spent in bed alongside the hours until sleep, as an area graph. The second graph is a line/scatter chart displaying the times woken up against the sleep quality rating.

The first graph is displayed by encapsulating two 'VictoryArea' components (hours in bed, hours until sleep) in a 'VictoryChart' component. The area chart displaying hours in bed is displayed first, as the second graph will be displayed over it, essentially showing us what portion of the total time in bed was spent asleep and what portion was spent awake. This works more effectively than encapsulating both area charts in a 'VictoryStack' as stacking them would produce incorrect data, as it would add the two together as opposed to combining them.

The second graph is displayed by encapsulating two 'VictoryGroup' components within a

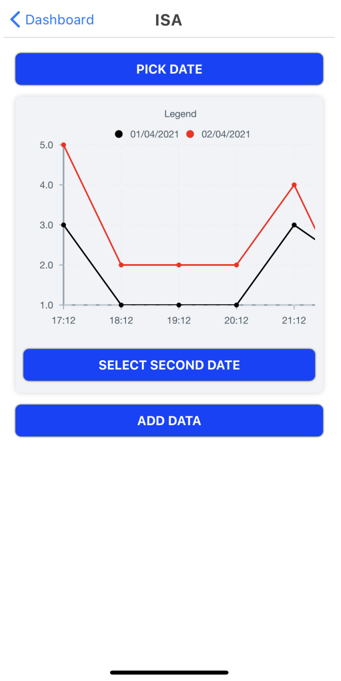
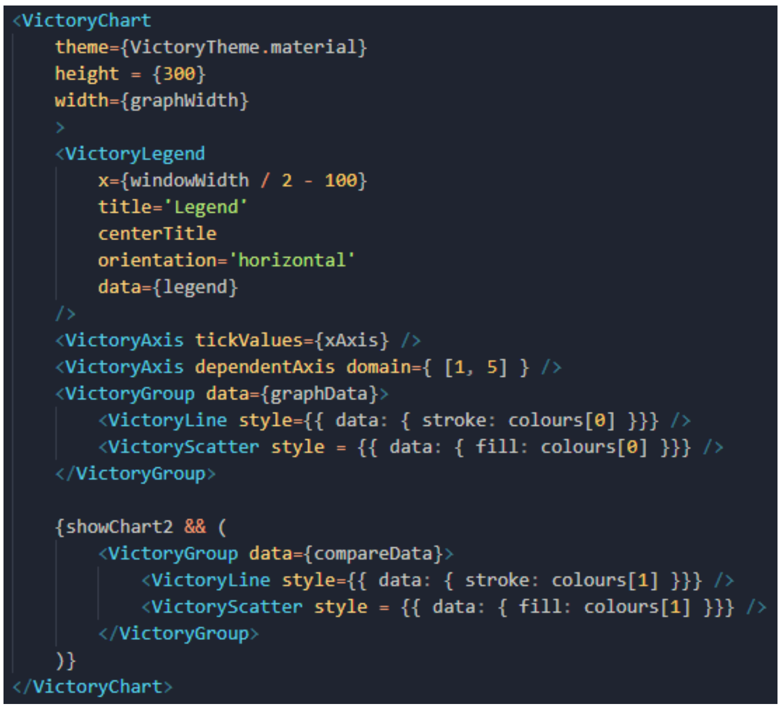


Figure 14: ISA Page



```
<VictoryChart
    theme={VictoryTheme.material}
    height = {300}
    width={graphWidth}
    >
    <VictoryLegend
        x={windowWidth / 2 - 100}
        title='Legend'
        centerTitle
        orientation='horizontal'
        data={legend}
    />
    <VictoryAxis tickValues={xAxis} />
    <VictoryAxis dependentAxis domain={ [1, 5] } />
    <VictoryGroup data={graphData}>
        <VictoryLine style={{ data: { stroke: colours[0] }}} />
        <VictoryScatter style = {{ data: { fill: colours[0] }}} />
    </VictoryGroup>

    {showChart2 && (
        <VictoryGroup data={compareData}>
            <VictoryLine style={{ data: { stroke: colours[1] }}} />
            <VictoryScatter style = {{ data: { fill: colours[1] }}} />
        </VictoryGroup>
    )}
</VictoryChart>
```

Figure 15: ISA Graph Code

'VictoryChart'. Both 'VictoryGroup' components contain a line chart and scatter plot for times woken up/sleep quality.

7.5 Data Entry

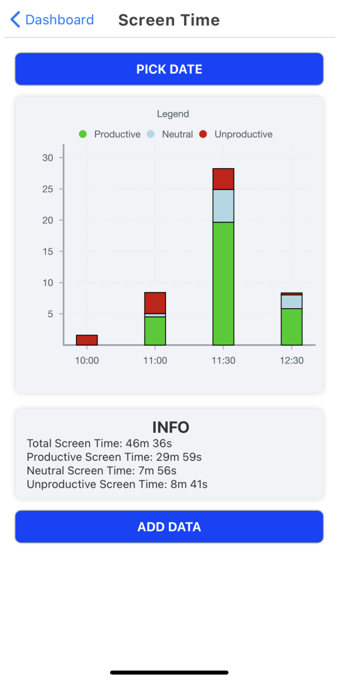
Data entry is provided through the debug mode (available in settings, under 'Developer Options') and allows the user to generate example data or input their own data through in-app forms. This feature is used particularly for development/researcher purposes since a user would likely want to input their data through other means, such as having the application read an excel spreadsheet with the data within it. While full data entry such as this is something that has been planned to be implemented in the application, the timescale has not allowed me to implement the feature in the duration of this project as the foundations and effective visualizations of data had to be prioritised.

Data validation is implemented within the form submission to ensure the data in each field is in the correct format and within the pre-defined constraints; should any fields be filled in incorrectly (in the incorrect format or not filled in when they should) a useful error message is displayed from the bottom of the screen to alert the user. This also applies if an error occurs within the database and thus the data does not enter the database.

7.6 Android/iOS Cohesive Design

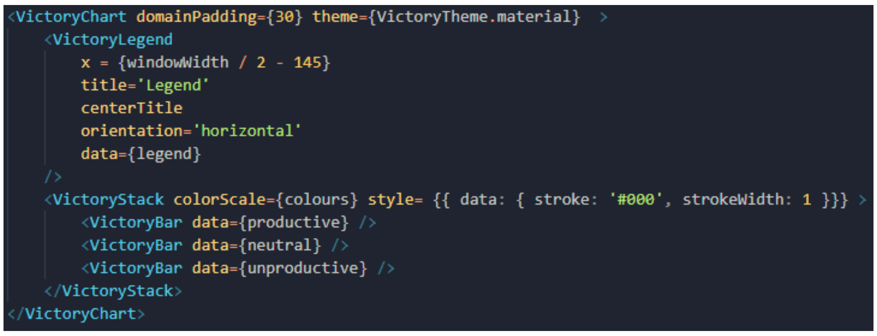
Using the default native components made the Android and iOS versions of the app look quite different (primarily the default button and fonts); I made the design decision to create a custom button component for a better put-together look and to achieve better cross-platform cohesiveness, along with a change in header font/font size.

Other than this, React Native handled the cross-platform consistency quite well by itself, and I just needed to add the odd adjustment to margins/padding dependent on the platform



INFO
Total Screen Time: 46m 36s
Productive Screen Time: 29m 59s
Neutral Screen Time: 7m 56s
Unproductive Screen Time: 8m 41s

Figure 16: Screen Time Page



```
<VictoryChart domainPadding={30} theme={VictoryTheme.material} >
    <VictoryLegend
        x = {windowWidth / 2 - 145}
        title='Legend'
        centerTitle
        orientation='horizontal'
        data={legend}
    />
    <VictoryStack colorScale={colours} style= {{ data: { stroke: '#000', strokeWidth: 1 }}} >
        <VictoryBar data={productive} />
        <VictoryBar data={neutral} />
        <VictoryBar data={unproductive} />
    </VictoryStack>
</VictoryChart>
```

Figure 17: Screen Time Graph Code

for some components.

8 Testing

The testing of this project involved manual testing, unit testing, snapshot testing and mocking data.

8.1 Manual Testing

Regular manual tests have been performed on the application to ensure all possible interactions work as expected; this includes selecting various dates for the graphs and testing the data validation of the forms.

8.2 Snapshot Testing

Snapshot tests[21] are simple to implement and ensure that the UI behaviour doesn't change unexpectedly. A snapshot test is created for each component within the project. The snapshot artifact is re-generated each time a working UI change is made.

Figure 22 shows an example of a snapshot test for the home component ("App"). This works by calling the test function to signify a test being created. The two arguments are the name of the test as it will appear when being run (in this case 'renders correctly') and the test itself (in this case an arrow function taking no arguments); the test renders the specified component, "App", to a JSON file and compares this to the currently saved snapshot (which is used as the reference point/standard, a guaranteed fully functioning snapshot) using the *expect(_).toMatchSnapshot()* function.

Should this test fail, the log will output the UI differences between the snapshots which allow for any errors in the code to be fixed; if the shown differences are intentional and the behaviour

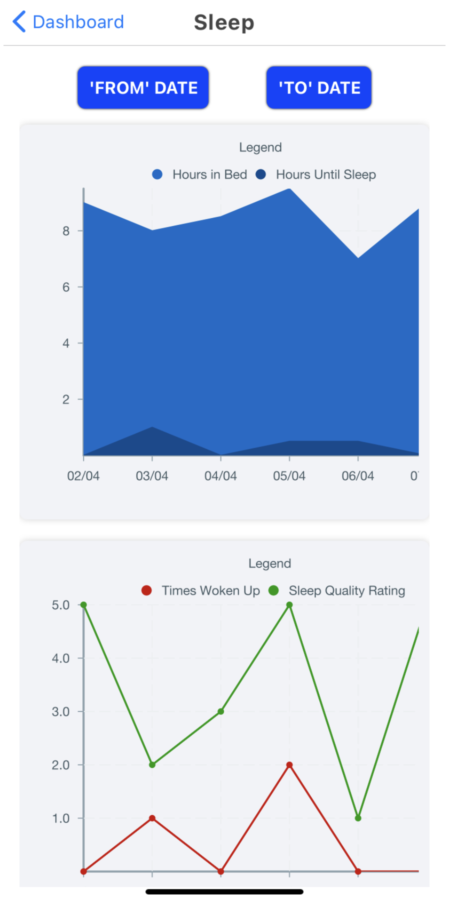


Figure 18: Sleep Page(1)

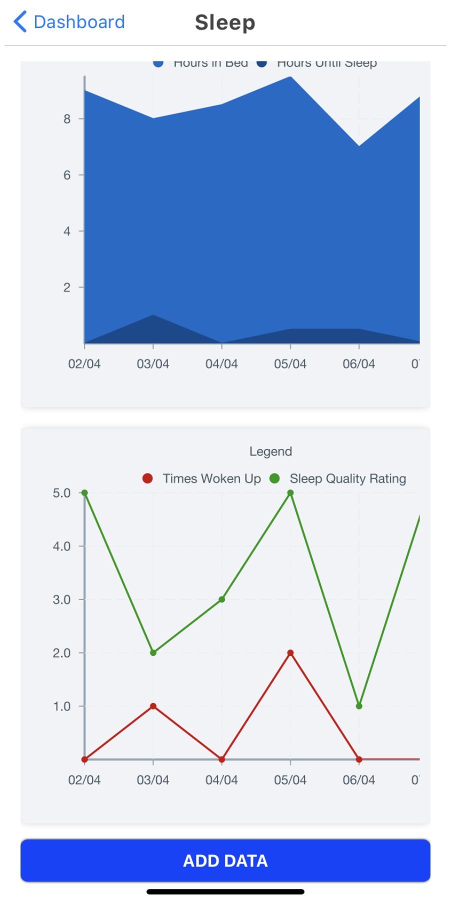


Figure 19: Sleep Page(2)

is expected, Jest can be run with the *–updateSnapshot* flag, which updates the snapshot file and uses that as the reference point/standard. Having to update snapshots after changes cause a fail in the test means being mindful of what pages end up looking like.

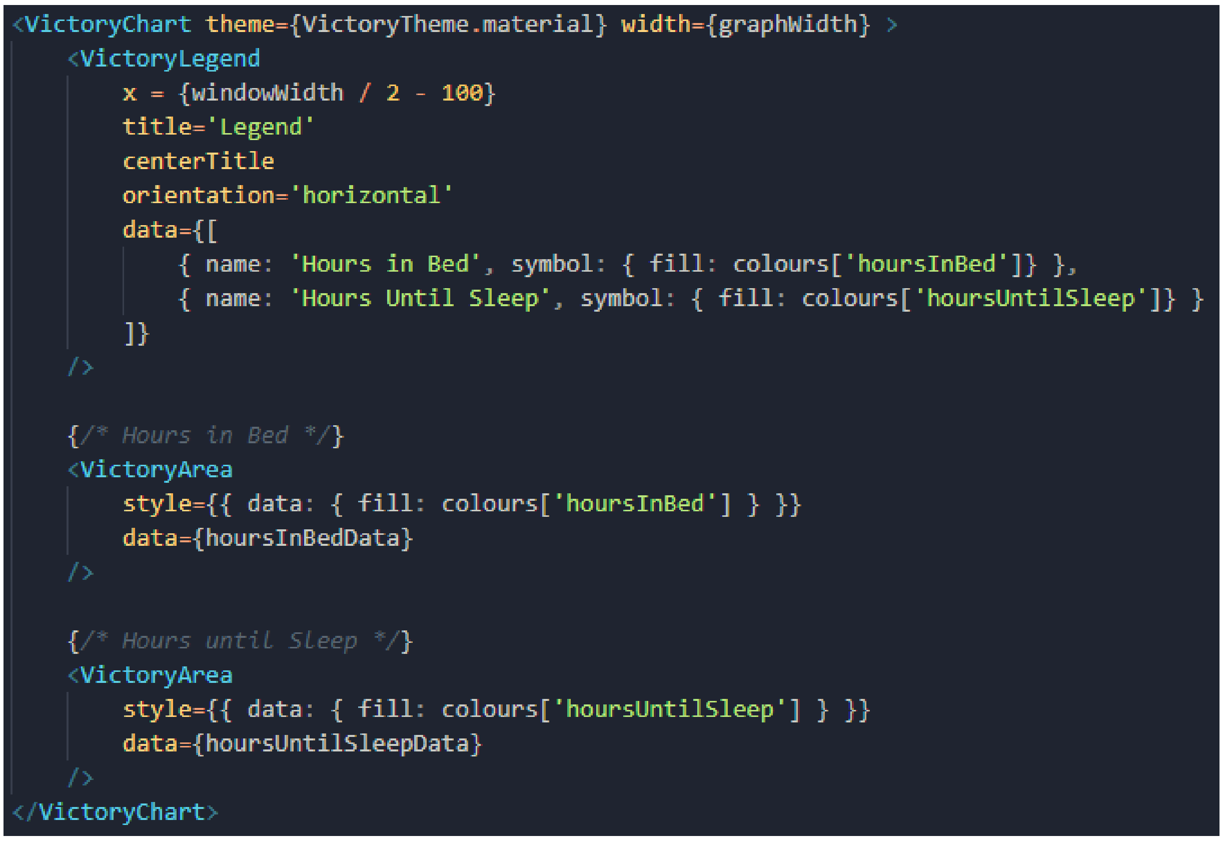
8.3 Unit Testing

Unit testing[22] involves testing whether the behaviour of each component is correct, as opposed to just checking for consistency for UI. In this case, the unit tests are checking to see if the components have the expected amount of children (testing correct behaviour); given that the number of children can change depending on if the user has enabled debug mode, this is also checked for (this requires data mocking, details discussed in section 8.4. Unit tests are written in the same test suites as snapshot tests.

Figure 23 shows an example of a unit test for the ISA visualisation page. This works by having the renderer create a JSON export of the ISA page component and comparing the number of children to what is expected; the test passes if the two values are equal. This assert only compares the number of children 'one level deep' much like a shallow renderer[23].

8.4 Data Mocking

Data mocking[24] is used to simulate the behaviour of real data in the testing process, and to ensure the tested components behave expectedly with different data.



```
<VictoryChart theme={VictoryTheme.material} width={graphWidth} >
    <VictoryLegend
        x = {windowWidth / 2 - 100}
        title='Legend'
        centerTitle
        orientation='horizontal'
        data={[
            { name: 'Hours in Bed', symbol: { fill: colours['hoursInBed']} },
            { name: 'Hours Until Sleep', symbol: { fill: colours['hoursUntilSleep']} }
        ]}
    />

    {/* Hours in Bed */}
    <VictoryArea
        style={{ data: { fill: colours['hoursInBed'] } }}
        data={hoursInBedData}
    />

    {/* Hours until Sleep */}
    <VictoryArea
        style={{ data: { fill: colours['hoursUntilSleep'] } }}
        data={hoursUntilSleepData}
    />
</VictoryChart>
```

Figure 20: Sleep Graph 1 Code

In the case of this project, the application context values needed to be mocked to test instances of debug mode being on or off. Figure 24 and Figure 25 show examples of two test suites for the ISA page component, one with debug mode off and one with debug mode on (respectively). The debug mode context value is mocked by wrapping the 'ISA' component in an 'AppContext.Provider' with the values being manually inserted. For this test, in particular, two children are expected when debug mode is off (the default page having just the date button and the 'FlashMessage' component), and three children when debug mode is on (the addition of the 'Add Data' button).

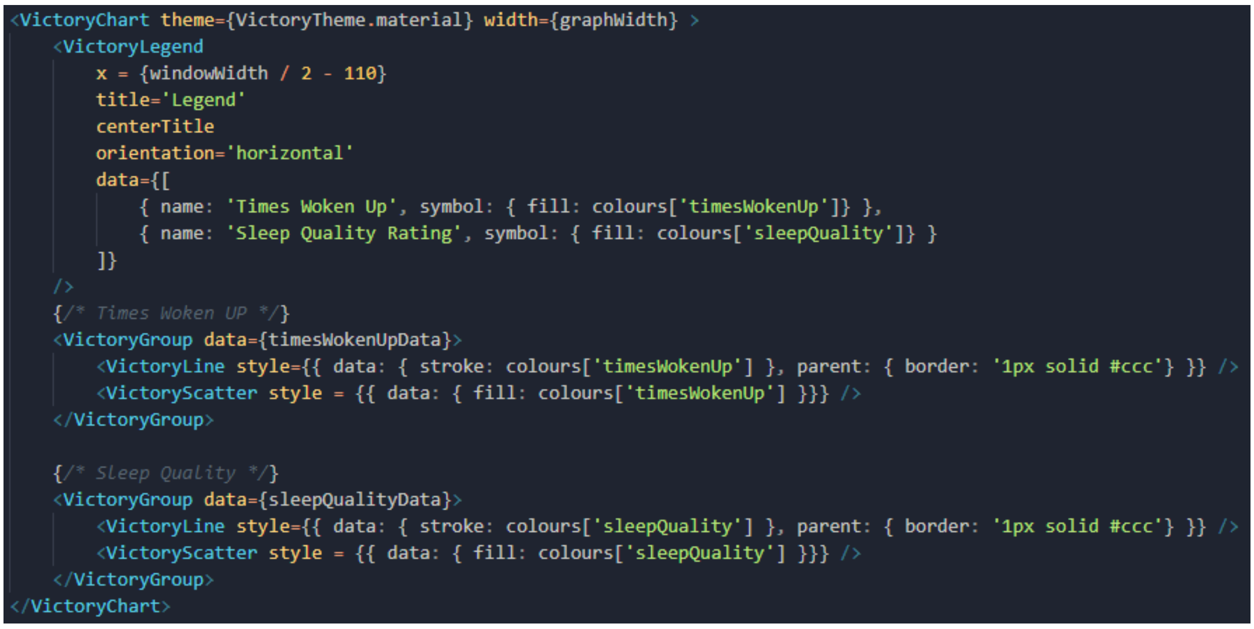
Another part of the testing process which involved mocking was the database calls. While this is not being mocked in the current version, it was earlier in development. Figure 26 shows an example of the 'DBHelper' class being mocked; the Jest automatic mock[25] is used which replaces the constructor and all functions of the class with dummy functions that all return 'undefined', thus allowing testing to be done on the component at hand.

8.5 Continuous Integration

I am using GitLab CI/CD[7] to implement continuous integration[26] in the development of this project. With each push to the repository, GitLab will automatically run the test suites before building the project and publishing it to expo as the latest release given the tests all pass.

As shown in Figure 27, lines 1-16 (inclusive) install the dependencies needed to run the jest tests in the CI environment along with optimising jest performance using caching (line 7). Lines 17 onward deploy a new build to Expo, the *$EXPO_USERNAME* and *$EXPO_PASSWORD*

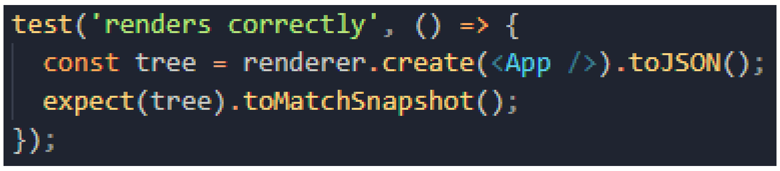
[7]https://docs.gitlab.com/ee/ci/



```
<VictoryChart theme={VictoryTheme.material} width={graphWidth} >
    <VictoryLegend
        x = {windowWidth / 2 - 110}
        title='Legend'
        centerTitle
        orientation='horizontal'
        data={[
            { name: 'Times Woken Up', symbol: { fill: colours['timesWokenUp']} },
            { name: 'Sleep Quality Rating', symbol: { fill: colours['sleepQuality']} }
        ]}
    />
    {/* Times Woken UP */}
    <VictoryGroup data={timesWokenUpData}>
        <VictoryLine style={{ data: { stroke: colours['timesWokenUp'] }, parent: { border: '1px solid #ccc'} }} />
        <VictoryScatter style = {{ data: { fill: colours['timesWokenUp'] }}} />
    </VictoryGroup>

    {/* Sleep Quality */}
    <VictoryGroup data={sleepQualityData}>
        <VictoryLine style={{ data: { stroke: colours['sleepQuality'] }, parent: { border: '1px solid #ccc'} }} />
        <VictoryScatter style = {{ data: { fill: colours['sleepQuality'] }}} />
    </VictoryGroup>
</VictoryChart>
```

Figure 21: Sleep Graph 2 Code



```
test('renders correctly', () => {
  const tree = renderer.create(<App />).toJSON();
  expect(tree).toMatchSnapshot();
});
```

Figure 22: Snapshot Test for App.js UI Component

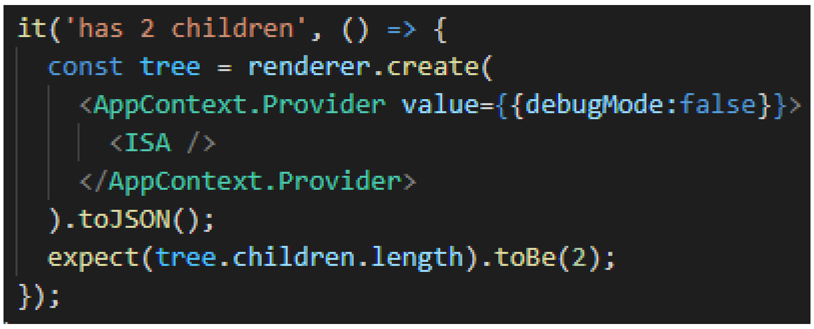
variables are set as environmental variables within GitLab.

This was previously done using Travis when the project was being developed on GitHub, however, this was easily ported over to GitLab's integrated CI/CD after moving project development to GitLab, as I had a better understanding of continuous integration and the script changes were relatively simple.

8.6 Testing Results

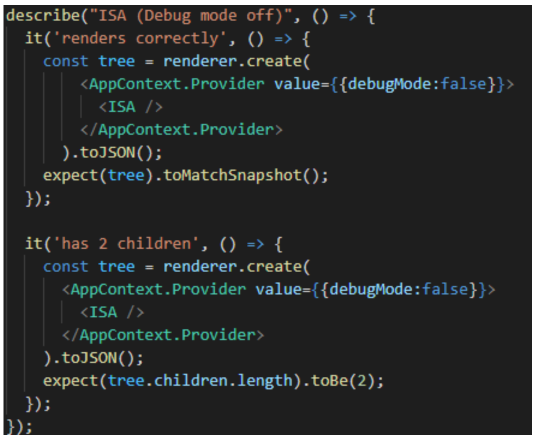
Figure 28 shows every single test and test suite written for this project (24 tests across 9 different test suites and the description of each test. According to the last instance of this pipeline, the tests took a total of 34.513s to run and pass. Comparing to the other pipelines this was roughly the average time; however it is also worth noting that this was just the time taken to run the tests, the full pipeline including tests and building/deployment took 10-12 minutes for a successful pipeline (Figure 29).

Also in Figure 29, we can see that across 45 pipelines, 39 were successful and 6 failed, which gives us a success ratio of 86.67%. Looking at the failed pipelines, they were shown to mostly be forgotten snapshot test updates meaning there were not many cases of the app showing unexpected behaviours after pushing back to the repository; these were particularly helpful however since it reminded me to test them locally first. Given this data, it is fair to say that being mindful of the testing process has led to better code being pushed, and the app responded successfully to the tests.



```
it('has 2 children', () => {
  const tree = renderer.create(
    <AppContext.Provider value={{debugMode:false}}>
      <ISA />
    </AppContext.Provider>
  ).toJSON();
  expect(tree.children.length).toBe(2);
});
```

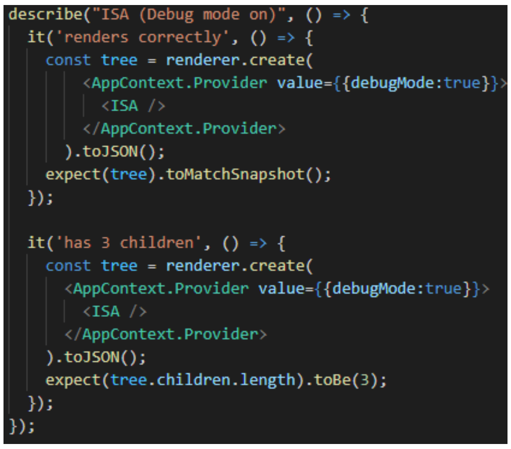
Figure 23: Unit Test for isa.js component



```
describe("ISA (Debug mode off)", () => {
  it('renders correctly', () => {
    const tree = renderer.create(
      <AppContext.Provider value={{debugMode:false}}>
        <ISA />
      </AppContext.Provider>
    ).toJSON();
    expect(tree).toMatchSnapshot();
  });

  it('has 2 children', () => {
    const tree = renderer.create(
      <AppContext.Provider value={{debugMode:false}}>
        <ISA />
      </AppContext.Provider>
    ).toJSON();
    expect(tree.children.length).toBe(2);
  });
});
```

Figure 24: ISA Page Test Suite (debug off)



```
describe("ISA (Debug mode on)", () => {
  it('renders correctly', () => {
    const tree = renderer.create(
      <AppContext.Provider value={{debugMode:true}}>
        <ISA />
      </AppContext.Provider>
    ).toJSON();
    expect(tree).toMatchSnapshot();
  });

  it('has 3 children', () => {
    const tree = renderer.create(
      <AppContext.Provider value={{debugMode:true}}>
        <ISA />
      </AppContext.Provider>
    ).toJSON();
    expect(tree.children.length).toBe(3);
  });
});
```

Figure 25: ISA Test Suite (debug on)

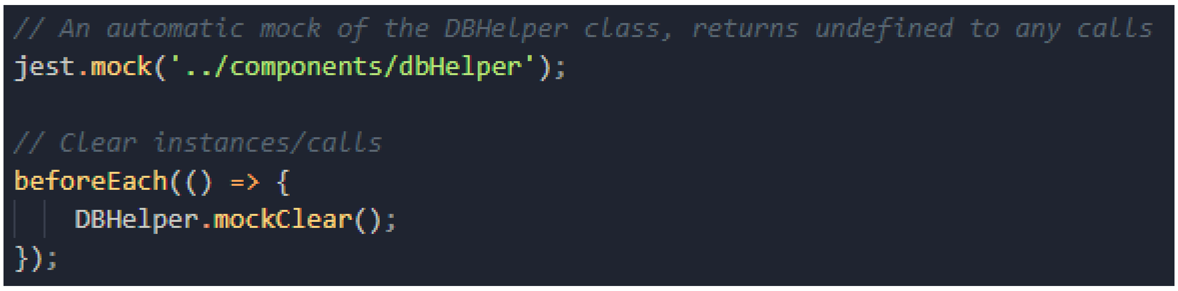
9 Deployment

During the development stage, a working version of the application is deployed to my Expo profile automatically using GitLab's CI integration; once all tests pass the application is automatically built and deployed. The expo project is set to public so the link can be shared, meaning the client has access to it should they wish to view the progress of the project at any time; it is also helpful for a development team to allow comparison of any branches against the deployed development version.

10 Maintenance

Partway through development, the project was moved over to the Brain Data Group GitLab page so the maintenance handover can remain as smooth as possible; the research team can stay in the loop with development progress as they can view all issues, commits and repository information rather than it all being on my private GitHub page. To facilitate a smoother transition, I continued to add issues for bugs and features regardless of whether they will be completed by myself or not, to follow proper software development practices.

Moving over to GitLab also meant switching continuous integration from Travis to the GitLab integrated service. This was just a case of changing the CI script (Figure 27) and going through the setup process once more on GitLab.



```
// An automatic mock of the DBHelper class, returns undefined to any calls
jest.mock('../components/dbHelper');

// Clear instances/calls
beforeEach(() => {
    DBHelper.mockClear();
});
```

Figure 26: DBHelper class mock

11 Client Discussion/Evaluation

Given where the project was at and the stage of development, no user evaluation was necessary; a more useful evaluation step was to meet with the client (research group) and discuss the application.

After having completed the core functionalities of the application I had a discussion with the research team (the client) about the project progress and where it is headed, any changes in direction or improvements to be made, and things to keep in mind/consider during further development.

While the core functionalities were in the right direction and allowed for basic visualisation of the stored data, further refinements needed to be made to the pages and visualisations. This primarily referred to the sleep page (Figure 30) which displayed all the stored sleep data onto the same graph and axis. This was an ineffective visualisation and needed changing (changes and reasoning discussed in detail in section 12.2).

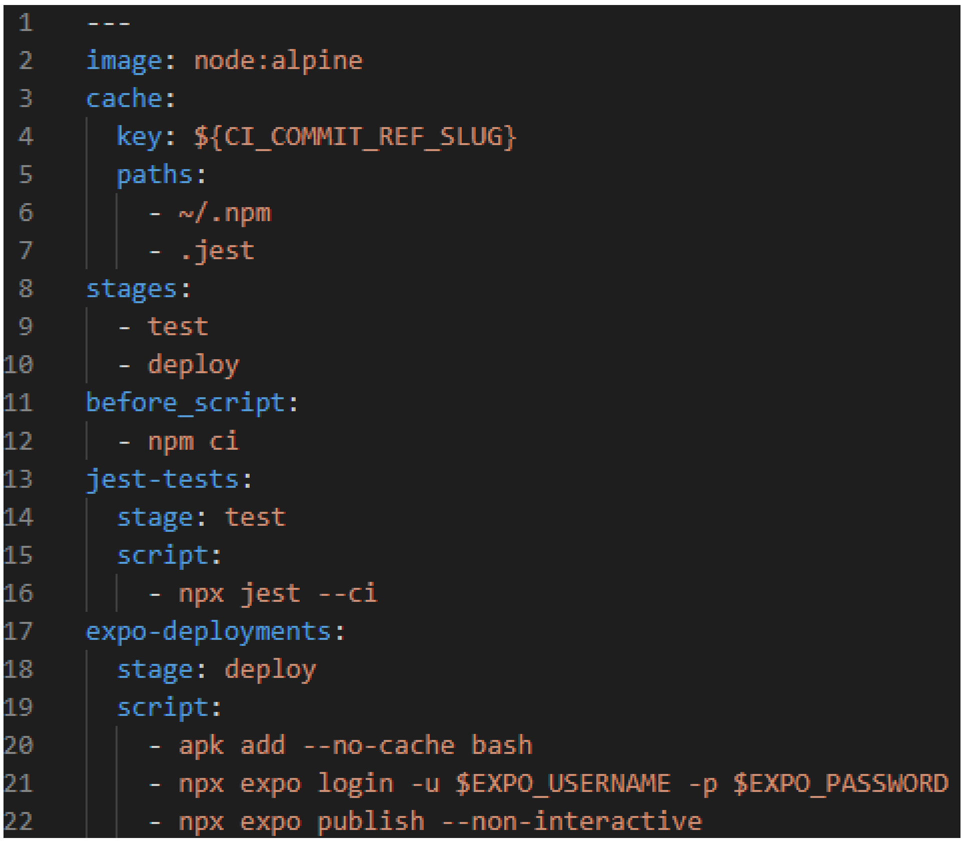
Another point brought up in discussion was the implementation of a briefing section on the dashboard page to summarise data for each different type of data (ISA, sleep, screen time) to avoid having to navigate to each section. This could involve having one index/measurement for each category to track progress and give a quick and convenient insight, with an option to view the whole day or the past few hours depending on the data. Part of the brief insights could be looking for outlier data such as any particular high-low sleep times on any given nights, or comparing related types of data such as sleep and screen time; information such as the way screen time affects sleep quality/time could be particularly useful.

A deeper look into the target of the application could prove useful in clarifying the direction of this project; is the target a user or researcher, and how is this app useful for each? A researcher may want to view all the data in more detail and have more flexibility with the way it is displayed to reach better scientific conclusions, whereas a standard user may prefer to have the data simplified as much as possible in a manner that allows them to make easy decisions on their lifestyle to improve cognitive function.

Even if the features discussed are not able to be implemented by me, keeping these in mind will ensure the app is set up and developed in a way to allow them to be created, such as having the right data collected and stored in the correct format(s).

11.1 Supportive Documentation

This project needs to have sufficient supportive documentation to allow for it to be picked up by the next development team in a smooth handover process. To do this, I created a wiki(Appendix



```yaml
---
image: node:alpine
cache:
  key: ${CI_COMMIT_REF_SLUG}
  paths:
    - ~/.npm
    - .jest
stages:
  - test
  - deploy
before_script:
  - npm ci
jest-tests:
  stage: test
  script:
    - npx jest --ci
expo-deployments:
  stage: deploy
  script:
    - apk add --no-cache bash
    - npx expo login -u $EXPO_USERNAME -p $EXPO_PASSWORD
    - npx expo publish --non-interactive
```

Figure 27: GitLab CI/CD Test/Build Script

A) outlining the steps required to set up the project on a local machine to continue development, and the main libraries used with links to their documentation. On top of this, a project architecture page (Appendix A.2) is in the wiki which shows the hierarchical structure of the project and where the main project files are in relation to each other.

A more implicit form of supportive documentation in this project was commenting throughout the codebase to keep everything clear, including what different components are and what they are used for; all of these together should allow anyone to pick up the project and understand the code/have the ability to continue from it.

12 Discussion

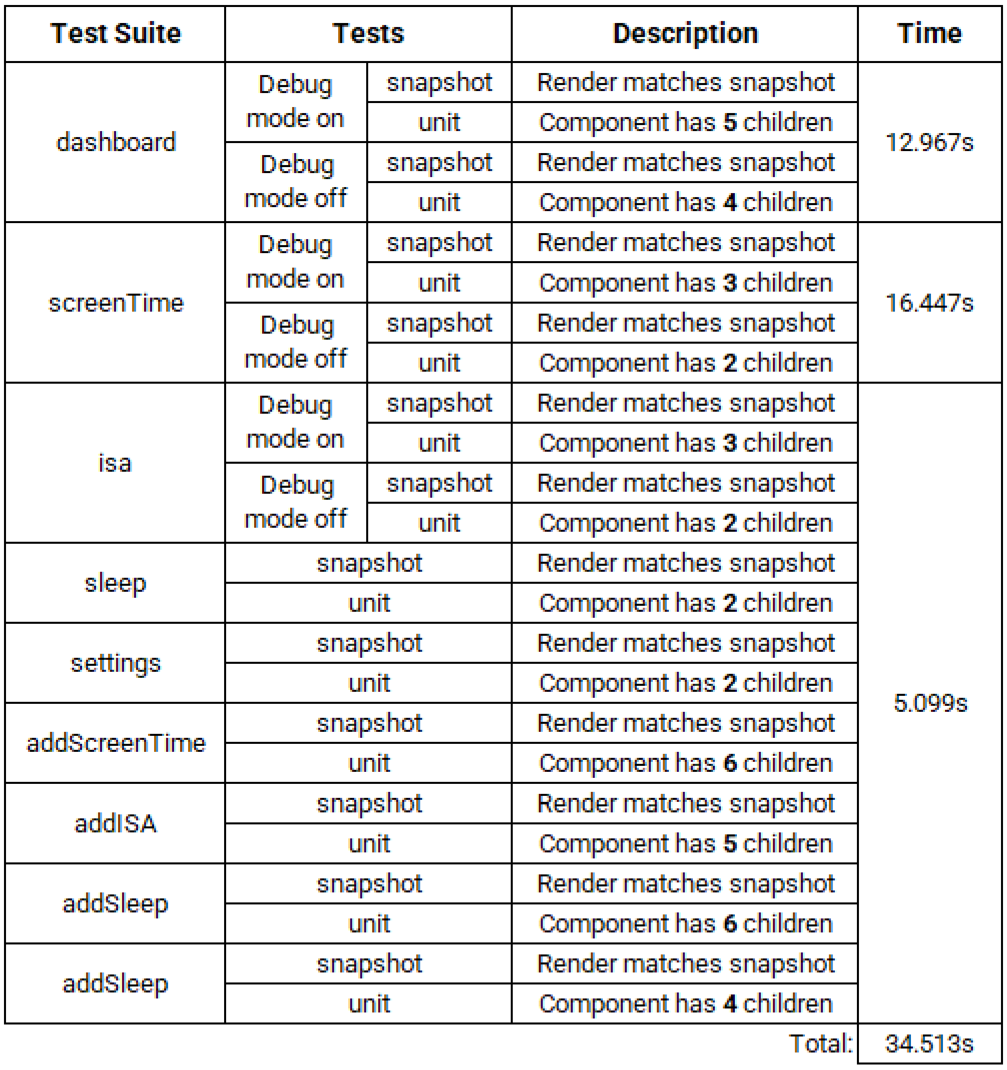
12.1 Interest in Project

I have a personal interest in this project as I revolve a lot of my actions and schedule around optimizing cognitive function to improve focus and productivity. A big part of this is tracking physical activity such as steps, exercising and food intake. However not much has come yet to allow for tracking cognitive activity, which I find would be helpful, and the research in this field along with the work on this project is promising in showing that this could be done.

12.2 Visualization Effectiveness

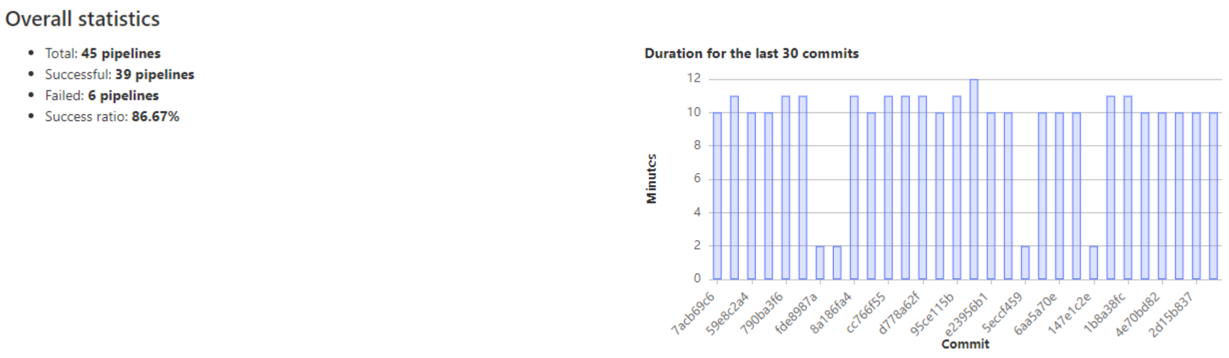
12.2.1 ISA

The ISA data is visualized by combining a line chart and scatter plot on the same graph/axis (Figure 14); the x-axis represents the time the reading was taken, and the y-axis represents the



Test Suite	Tests		Description	Time
dashboard	Debug mode on	snapshot	Render matches snapshot	12.967s
		unit	Component has **5** children	
	Debug mode off	snapshot	Render matches snapshot	
		unit	Component has **4** children	
screenTime	Debug mode on	snapshot	Render matches snapshot	16.447s
		unit	Component has **3** children	
	Debug mode off	snapshot	Render matches snapshot	
		unit	Component has **2** children	
isa	Debug mode on	snapshot	Render matches snapshot	5.099s
		unit	Component has **3** children	
	Debug mode off	snapshot	Render matches snapshot	
		unit	Component has **2** children	
sleep	snapshot		Render matches snapshot	
	unit		Component has **2** children	
settings	snapshot		Render matches snapshot	
	unit		Component has **2** children	
addScreenTime	snapshot		Render matches snapshot	
	unit		Component has **6** children	
addISA	snapshot		Render matches snapshot	
	unit		Component has **5** children	
addSleep	snapshot		Render matches snapshot	
	unit		Component has **6** children	
addSleep	snapshot		Render matches snapshot	
	unit		Component has **4** children	
			Total:	34.513s

Figure 28: Table of all tests



Overall statistics

- Total: **45 pipelines**
- Successful: **39 pipelines**
- Failed: **6 pipelines**
- Success ratio: **86.67%**

Figure 29: Testing Pipeline Analytics

mental workload rating for that time. The points on the scatter plot are the primary source of visualisation, as this is positional and represents the quantitative information (mental workload ratings) most accurately. This is then aided by the line chart which is represented as a slope and connects the data points to spot trends (seeing a visual representation of an increase or decrease in workload between data points).

As shown, again in Figure 14, a second plot exists to allow for a direct comparison of mental workload data between two days. Given the two graphs utilise the same data, differentiating two dates is necessary, and this was done by colour which has shown to be effective.

This form of visualising the ISA data has proved to be the most effective, allowing the user to understand what the data is representing effortlessly.

12.2.2 Sleep

Starting with the first instance of visualising sleep data, this was done by displaying all the data on the same axis a combination of a line chart and scatter plot (Figure 30). The data points being graphed were:

1. Hours in bed
2. Hours until sleep
3. Times woken up
4. Sleep quality rating

This was highly ineffective and demonstrated poor practice, as there were two different scales and types of measurement being displayed on the same axis (hours and quantity). It made sense to separate the data visualisations into two graphs, one containing the hours in bed and hours until sleep (hours), and the other containing times woken up and sleep quality rating (quantity).

The first graph (Figure 18) is a visualisation containing two area charts displayed on the same axis; the x-axis represents the date and the y-axis represents time (in hours). The first area chart to be rendered is the ‘hours in bed’ data, and then the ‘hours until sleep’ data is graphed on top of the first area chart. This is different to displaying it as a stacked area chart, and it is important to recognise the difference, as the ‘hours in bed’ data already describes the total time spent in bed, so stacking the two on top of each other would give a false representation of the data. The configuration of the visualisation being as such makes it so that three pieces of information can be easily seen from the two charts:

1. Total time spent in bed: this is seen by looking at the peak of the graph for any given night.
2. Time taken to sleep: this is seen by looking at the shaded area of the graph.
3. Time spent asleep: slightly different to the total time spent in bed, this is seen by looking at the lighter area of the graph.

The second graph (Figure 19) is a visualisation containing two combinations of a line chart and scatter plot. The two datasets being represented on the said graph are the number of times woken up and the sleep quality rating, with the differentiating factor between them being colour.

The separation of the data into two separate graphs proved to be much more effective at giving the user a more convenient and well-created visualisation with the correct scales and measurement types. To make the comparison of the two graphs visually easier, the cards and axis’ lined up vertically so the data points are in vertical alignment.

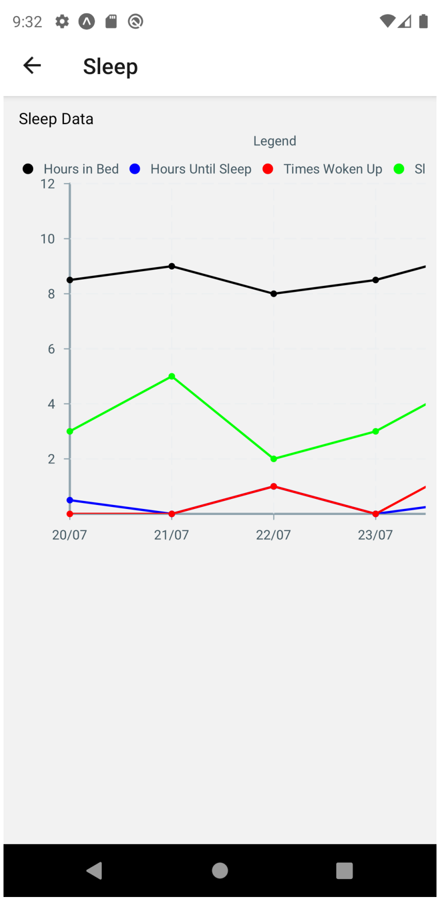


Figure 30: Initial graph design of sleep data

12.2.3 Screen Time

Screen time data is visualised by stacking bar charts on top of each other on the same axis (Figure 16); the x-axis represents a time interval and the y-axis represents time spent on-screen (in minutes). Every time on the x-axis represents the beginning of the half-hour time interval the data resides in (e.g. 10:00 represents the 10:00-10:30 time interval). All productive, neutral and unproductive screen time is added up and displayed as a bar, then the categories are stacked on top of each other for each time interval. The categories are then differentiated from each other by colour: productive is green, neutral is grey-blue, unproductive is red.

This visualisation strategy was deemed most effective as the colours work alongside what a user would naturally classify each category as, on top of the fact that the visualisation allows the user to see how much individual time was spent per category and how much time was spent in the entire interval. As well as this, it is easy to compare the time spent in each category per time interval as the size of each bar directly represents the amount of time spent; the user does not need to inspect the graph in detail to understand what is being visualised.

12.3 COVID-19

Given the world climate at the beginning of the academic year (and up until this point too), the uncertainty of the direction our day-to-day life would take regarding COVID-19 meant that I had to prepare for a worsening situation.

As I have access to my own personal computer, Android phone (and emulator) and iOS phone, all development was completed from home. I also had regular contact with my supervisor through weekly meetings on Microsoft Teams, along with occasional meetings with the research team (client/user) to discuss the research and project progress; there was little to no impact on communication.

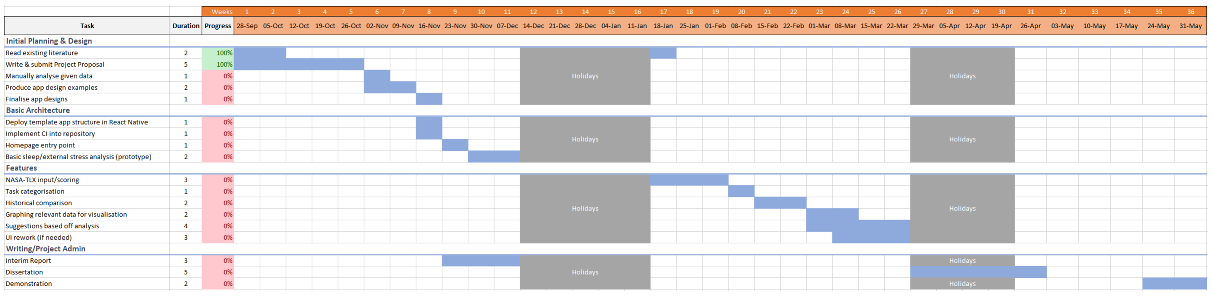
As expected and planned for, the COVID-19 situation hasn't affected the progress of this project with regards to access to the necessary equipment/technology.

13 Summary and Reflections

13.1 Project Management

Figure 31 shows my original plan as submitted in my project proposal. My initial understanding of this project was an app that could take user input directly through the app itself and work on analysing it to provide feedback to the user based on the data, however after discussion with the research team and further thought I pivoted as there was a misunderstanding; this way would present some complications as to the accuracy of the analysis, as well as the limited use it would have within brain data research. Instead, the visualisation of the data would prove much more useful as displaying the data in a convenient format would allow for the individual, or research group, to draw their own informed conclusions by e.g. comparing mental workload ratings to their sleep ("I only slept 4 hours last night, it makes sense my mental workload ratings for this task was ranging 4-5 when it's usually 2-3").

Due to this, the work plan was slightly delayed but this was reasonable. The data analysis task primarily consisted of meeting with the research team and working through some of the core files within the folders and gaining a better understanding of the raw data collected and how it fitted into this project/how this data is useful and could be implemented to help gain



Task	Duration	Progress
Initial Planning & Design		
Read existing literature	2	100%
Write & submit Project Proposal	5	100%
Manually analyse given data	1	0%
Produce app design examples	2	0%
Finalise app designs	1	0%
Basic Architecture		
Deploy template app structure in React Native	1	0%
Implement CI into repository	1	0%
Homepage entry point	1	0%
Basic sleep/external stress analysis (prototype)	2	0%
Features		
NASA-TLX input/scoring	3	0%
Task categorisation	1	0%
Historical comparison	2	0%
Graphing relevant data for visualisation	2	0%
Suggestions based off analysis	4	0%
UI rework (if needed)	3	0%
Writing/Project Admin		
Interim Report	3	0%
Dissertation	5	0%
Demonstration	2	0%

Weeks
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
28-Sep 05-Oct 12-Oct 19-Oct 26-Oct 02-Nov 09-Nov 16-Nov 23-Nov 30-Nov 07-Dec 14-Dec 21-Dec 28-Dec 04-Jan 11-Jan 18-Jan 25-Jan 01-Feb 08-Feb 15-Feb 22-Feb 01-Mar 08-Mar 15-Mar 22-Mar 29-Mar 05-Apr 12-Apr 19-Apr 26-Apr 03-May 10-May 17-May 24-May 31-May
Holidays

Figure 31: First version of Gantt chart

an understanding of the numbers.

Deploying the template app structure in React Native was relatively simple, but I did experience some difficulty implementing CI into the repository. This primarily came from a lack of experience with continuous integration itself, as I had never had to set up automated tests/builds, the closest being writing Ansible playbooks for automated application deployments on a virtual machine (which isn't quite the same). However, this was set up and working in reasonable time; upon pushing to the repository the script automatically runs all Jest tests and builds the project to Android and iOS, before publishing the project to my personal Expo account. This quite easily allows me to compare the current state of the project (in whichever feature I'm working on) to the latest development version.

Another part of the initial project management that wasn't sufficiently accounted for accurately was the stress/workload of other modules I have been undertaking; while I accounted for them to some extent, I underestimated the effect they would have on my studies, especially the time other coursework projects would take. This, as well as my strict training schedule outside of my studies, meant that this project was not prioritised as well as it should have in the first semester. Using this experience of time management and the better understanding of the project's direction/purpose, I updated the Gantt chart (Figure 6) to reflect a more accurate time estimation of future tasks and prioritise the most important aspects of this application (visualising ISA/Sleep/Screen Time data).

After the winter holiday when the semester started again, I began the project again from scratch as I believe the learning opportunities and mistakes made during early development lead to a quality of implementation which I was not happy with; bringing the project up to the same point did not take too long as I had the benefit of practising the previous version. This also helped aid further development as the foundation of the project was cleaner and had a better implementation.

Towards the end of February, the snapshot tests stopped working due to the components being tested needing access to the database to display the graphs. This happened to just one component earlier in the month and thus the test was temporarily disabled to allow for the features to be completed; however, this reached a point where it must be addressed as most components were using the database and required a fix in order to keep tests running. Despite the difficulty in finding documentation regarding this, it was fixed within 2 weeks through the use of creating a database helper class and then mocking the class in the test scripts. Development resumed with all tests functioning properly.

Another setback during development was the switch to the 'Victory Native' graphing library from 'React Native Chart Kit' (section 7.4) in the middle of March. The need for this change came about due to being unable to effectively visualise screen time data and the limited functionality of the previously used graphing library. The switch slowed down development temporarily as I had to re-implement the previous graphs, but this allowed for higher quality implementation and better options for future development.

13.2 Time Management

A consistent effort has been put into this project since the project proposal went through, and while it did not show in development until it started picking up speed in January/February, this is due to the learning curve of the framework as well as the background reading and understanding that had to be done. I think I managed my time effectively and adapted to changing situations as well as I could, however critically thinking I believe starting the learning process earlier would have yielded more effective results and would have prevented the need to restart the codebase again.

13.3 Objectives and Requirements

All 7 key objectives (section 2.2) have been met as per their descriptions throughout the duration of the project.

13.3.1 Functional Requirements

1. **The user can select a date and have the application display all ISA data related to that day in a graph:** this requirement has been met as shown in Figure 14.

2. **The user can select a second date to view ISA data in direct comparison to the first:** this requirement has been met as shown in Figure 14.

3. **The user can select a date and have the application display all screen time data related to that day in a graph:** this requirement has been met as shown in Figure 16.

4. **The user can select a date range and have the application display all sleep data related to that range in a graph:** this requirement has been met as shown in Figures 18&19.

5. **A developer/researcher can able to manually add data from within the app and add generated example data:** this requirement has been met, as shown in Figure 10 which shows the developer options allowing for example data to be generated and manipulated, as well as Figures 11&12&13 which show the ability to add data manually given debug mode is enabled from developer options.

I have successfully met all functional requirements set at the beginning of this project.

13.3.2 Non-Functional Requirements

1. **All data must be stored locally, kept on the user's device, and deleted should the application be deleted:** as discussed in section 7.2, all data has been stored locally in an SQLite database and no communication is made outside of the user's mobile device. On top of this, not only will the data be deleted upon application deletion, but the option to delete all data is available within settings under developer options after enabling debug mode (Figure 10). This requirement has been met.

2. **Data must be validated before being stored:** this requirement has been met, as discussed in detail in section 7.5.

3. **The UI must be consistent across platforms:** I believe this requirement to have been strongly met despite being a more subjective requirement to assess; the use of a custom button component and cohesive design being thought out throughout the whole process has meant that the app looks almost identical across both iOS and Android.

13.4 Achievements and Final Reflection

While there were plenty of setbacks in the duration of this project, this is a normal part of any development process so I handled it with the approach of using it as a learning opportunity, and it also meant that the process became quicker and more efficient as time went by.

I believe I worked consistently and effectively during the development stage, and following good practices which meant the quality of the deliverable was high and, along with the supportive documentation and commenting throughout, should allow for a smooth transition for whoever would end up continuing development next. I also think that despite the setbacks these were handled well and the project was brought back to speed, such as the switch to Victory Native graphing library which was effectively done.

A slightly different approach I would take if I were to do this again would be to prioritise spending more time on reading the documentation of the libraries I plan on using and to practise the use of the framework before beginning the project, as my lack of practical experience in React Native beyond some basic apps meant that I had to learn and adapt from my mistakes during the process; however I think I handled the changes well and the next time I use this framework the process will be quicker and easier, and the choice of using React Native for this project was still the most appropriate given the needs of the app.

References

[1] M. Hagmüller, G. Kubin, and E. Rank, "Evaluation of the human voice for indications of workload induced stress in the aviation environment," 2005.

[2] T. C. Hankins and G. F. Wilson, "A comparison of heart rate, eye activity, eeg and subjective measures of pilot mental workload during flight.," *Aviation, space, and environmental medicine*, vol. 69, no. 4, p. 360, 1998.

[3] *Instantaneous Self Assessment of workload (ISA)*, 2009. `https://ext.eurocontrol.int/ehp/?q=node\%2F1585`.

[4] M. L. Wilson, N. Sharon, H. A. Maior, S. Midha, M. P. Craven, and S. Sharples, "Mental workload as personal data: designing a cognitive activity tracker," 2018.

[5] I. Li, A. Dey, J. Forlizzi, K. Höök, and Y. Medynskiy, "Personal informatics and hci: design, theory, and social implications," in *CHI'11 Extended Abstracts on Human Factors in Computing Systems*, pp. 2417–2420, 2011.

[6] M. Swan, "The quantified self: Fundamental disruption in big data science and biological discovery," *Big data*, vol. 1, no. 2, pp. 85–99, 2013.

[7] N. Herten, T. Otto, and O. T. Wolf, "The role of eye fixation in memory enhancement under stress–an eye tracking study," *Neurobiology of learning and memory*, vol. 140, pp. 134–144, 2017.

[8] H. Aghajani, M. Garbey, and A. Omurtag, "Measuring mental workload with eeg+ fnirs," *Frontiers in human neuroscience*, vol. 11, p. 359, 2017.

[9] B. Cinaz, B. Arnrich, R. La Marca, and G. Tröster, "Monitoring of mental workload levels during an everyday life office-work scenario," *Personal and ubiquitous computing*, vol. 17, no. 2, pp. 229–239, 2013.

[10] "Electrocardiogram (ecg) - nhs." `https://www.nhs.uk/conditions/electrocardiogram/`.

[11] S. A. Israel, J. M. Irvine, A. Cheng, M. D. Wiederhold, and B. K. Wiederhold, "Ecg to identify individuals," *Pattern recognition*, vol. 38, no. 1, pp. 133–142, 2005.

[12] J. R. Wilson and S. Sharples, *Evaluation of human work*. CRC press, 2015.

[13] G. Johannsen, "Workload and workload measurement," in *Mental workload*, pp. 3–11, Springer, 1979.

[14] N. H. Alsuraykh, M. L. Wilson, P. Tennent, and S. Sharples, "How stress and mental workload are connected," in *Proceedings of the 13th EAI International Conference on Pervasive Computing Technologies for Healthcare*, pp. 371–376, 2019.

[15] A. Cao, K. K. Chintamani, A. K. Pandya, and R. D. Ellis, "Nasa tlx: Software for assessing subjective mental workload," *Behavior research methods*, vol. 41, no. 1, pp. 113–117, 2009.

[16] A. Leggatt, "Validation of the isa (instantaneous self assessment) subjective workload tool," in *Proceedings of the International Conference on Contemporary Ergonomics*, pp. 74–78, 2005.

[17] B. Cinaz, R. La Marca, B. Arnrich, G. Tröster, *et al.*, "Monitoring of mental workload levels," 2010.

[18] S. C. Bunce, K. Izzetoglu, H. Ayaz, P. Shewokis, M. Izzetoglu, K. Pourrezaei, and B. Onaral, "Implementation of fnirs for monitoring levels of expertise and mental workload," in *International Conference on Foundations of Augmented Cognition*, pp. 13–22, Springer, 2011.

[19] V. Driessen, "A successful git branching model." `https://nvie.com/posts/a-successful-git-branching-model/`, Jan 2010.

[20] T. N. Ninja, "React native tutorial #21 - navigating around." `https://www.youtube.com/watch?v=PMX6GP1TXGo`, 2019.

[21] S. Bekkhus, "Snapshot testing." `https://jestjs.io/docs/snapshot-testing`, 2021.

[22] F. Ikechi, "Unit testing in react native applications." `https://www.smashingmagazine.com/2020/09/unit-testing-react-native-applications/`, 2020.

[23] "Shallow renderer." `https://reactjs.org/docs/shallow-renderer.html`.

[24] R. Hamilton, "Data mocking - a way to test the untestable." `https://blog.scottlogic.com/2016/02/08/data-mocking.html`, 2016.

[25] S. Lorber, "Es6 class mocks." `https://jestjs.io/docs/es6-class-mocks#automatic-mock`, 2021.

[26] "Setting up continuous integration." `https://docs.expo.io/guides/setting-up-continuous-integration/`.

A Appendix A - Project Wiki

A.1 Home

Project Setup

Initial Setup

1. Clone this repository
2. Follow this tutorial to set up Expo: https://docs.expo.io/get-started/installation/
3. In the root directory of the project, run command `npm install` to install project dependencies to the local node_modules folder.
4. Run command `npm add expo` to add expo to your local version of the project.

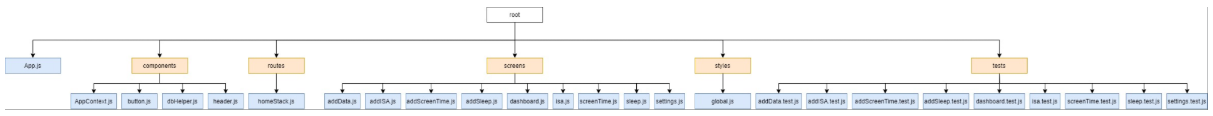
Running the Application

1. Run command `npm start` .
2. If you're on an Android device, scan the QR code from within the Expo Go app. If you're on an iOS device, scan the QR code with the camera app and click the link to open.

Main Libraries

Graphing - Victory Native
Alerts - React Native Flash Message
Database Storage - Expo SQLite
Date Picking - React Native DateTimePicker
Testing - Jest

A.2 Project Architecture


root
App.js
components
routes
screens
styles
tests
AppContext.js
button.js
dbHelper.js
header.js
homeStack.js
addData.js
addISA.js
addScreenTime.js
addSleep.js
dashboard.js
isa.js
screenTime.js
sleep.js
settings.js
global.js
addData.test.js
addISA.test.js
addScreenTime.test.js
addSleep.test.js
dashboard.test.js
isa.test.js
screenTime.test.js
sleep.test.js
settings.test.js