CMT403 – Final Report – 60 credits

A Personal Diary App for People with Obsessive-Compulsive Disorder

Author: Joe Pointon

Student Number: C1623024

Project Supervisor: Alia I Abdelmoty

Project Moderator: Dr Katarzyna Stawarz

04/11/2021

MSc Computing School of Computer Science and Informatics, Cardiff University

Abstract

This dissertation project covers the design, implementation, and evaluation of a journaling application for people with OCD and their therapists. This project is significant because it makes it easy for people to keep a journal and improves the connection between patients and therapists.

The app was designed with user-centred principles in mind and implemented using React Native. User testing was carried out with a System Usability Scale questionnaire and observations. The app received positive feedback and the participants agreed with statements such as “I thought the system was easy to use” and “I think I would like to use this system frequently”, showing the real-world function of the app. The significant implication of this is that online journaling could play a large roll in CBT, demonstrating the growing demand for previously unconsidered technological solutions for both patients and therapists.

Acknowledgments

I would like to thank my supervisor Alia Abdelmoty for help in shaping the direction of the project and her guidance throughout.

I would also like to thank my sister for proofreading the report, my partner for picking me up when I was down, and those who volunteered their time to participate in my testing.

Contents

1- Introduction

Obsessive-Compulsive Disorder (OCD) is a debilitating mental illness affecting many people globally. Currently the most successful treatment is Cognitive Behavioural Therapy (CBT), which can improve the symptoms of 75% of patients (OCD UK 2020). Journaling is an essential part of CBT because it allows patients to track their thought patterns and make connections between their thoughts and their behaviours. However, this often relies on patients recording this information in a physical journal, which can be inconvenient in many ways. While there are existing applications available for CBT journaling, none of them are designed specifically for OCD and they lack features such as being able to share entries with therapists.

This project therefore looks at this problem in a new way and aims to create a thought journal mobile application that assists patients and therapists in the process of treating OCD through CBT.

In order to meet this aim, the following main objectives have been set out:

1. To allow users to add a journal entry describing a triggering event along with all relevant aspects of the event and the subsequent thought processes
2. To allow therapists to view and edit their list of patients, and view and search their patients' journal entries
3. To allow users to view/edit/delete their past journal entries in a searchable list, which can be filtered in ways such as date or time period
4. To allow users to perform a quick check-in of their emotions and activities
5. To allow users to read and learn about cognitive distortions

In this report I will cover the background knowledge of OCD and its treatment through CBT (Section 2), the problem the app will solve, reviews of existing solutions and their shortcoming (Section 3), the approach taken to the implementation and management of the solution (Section 4), the definition of requirements and use cases (Section 5), initial user interface design and a heuristic analysis of these (Section 6), a detailed account of the implementation of the app (Section 7), an evaluation of the product through the use of test cases and user testing (Section 8), a conclusion on the success of the project (Section 9), recommendations

for future work (Section 10) and, finally a reflection on my learning experience throughout the project (Section 11).

2 - Background

2.1 - Obsessive-Compulsive Disorder

OCD is a recent medical term for a medical condition that has been around for centuries. Evidence of what are now recognised as common OCD related behaviours can be observed throughout the 18th and 19th centuries; examples of washing, checking, and excessive fear of disease were described in literature, and in the 19th century the modern theory of OCD was formed. In the 20th century the theory was expanded by the distinction between obsessions and compulsions. (OCD UK 2019)

In modern times OCD is acknowledged as a serious mental condition that affects 1.2% of the population (~750,000 people). The American Psychiatric Association (2013) state that around 50% of cases are severe and <25% are mild.

An important aspect of the understanding of OCD is the distinction between obsessions and compulsions. Obsessions are intrusive, unwanted, and often irrational thoughts, whereas compulsions are the behaviours people with OCD feel the need to perform in response to their obsessive thoughts. Usually, people with OCD can tell that their thoughts are irrational, but still feel the need to perform compulsive behaviours in response to an obsessive thought. These actions normally provide a temporary relief form the anxiety brought about due to their obsessive thoughts. (OCD UK 2018)

The most common categories of obsessive thought are (NICE 2005):

1. Contamination form dirt, germs, viruses, etc. (38%)
2. Fear of harm (24%)
3. Excessive concern with order or symmetry (10%)
4. Obsessions with the body or physical symptoms (7%)

The most types of compulsive behaviours are (NICE 2005):

1. Checking (e.g. repeatedly checking gas taps are off, doors are locked or that a loved one is ok) (29%)
2. Cleaning and washing (e.g. repetitively washing hands after using a public bathroom, shaking hands or touching a door handle) (27%)

3. Repeating acts (11%)

4. Mental compulsions (e.g. repeating a series of words in your head) (11%)

2.2 - Impacts of OCD

Some OCD suffers may spend a large amount of their time carrying out compulsions and be unable to carry out normal daily activities, while others may seem to be coping outwardly but are suffering a great amount of stress and anxiety from their obsessions. In either case the impacts of OCD are vast and can affect every aspect of a person's life, as well as those of their friends and families.

Impacts may include but are not limited to:

- Poor performance in education or employment
- Strain on relationships
- General quality of life
- Physical damage from compulsions (e.g. raw skin from repetitive cleaning)
- Substance abuse
- Friends and family may become involved in obsessions (e.g. avoiding certain places, excessive washing)

2.3 - Treatment of OCD

One method of treatment for OCD is cognitive behavioural therapy (CBT). Ponniah et al. (2013), a meta-review of 45 randomized controlled trials, found CBT to be 'efficacious and specific' for OCD treatment. The theory behind CBT is based on the idea that dysfunctional thoughts, emotions, and behaviours are all related, and that examining your thoughts at a deeper level can lead to an improvement in condition (Beck 2011).

The purpose of CBT for OCD is to break the link between obsessive thoughts and compulsive behaviours. Patients can learn to deal with their thoughts through the realization that everyone experiences intrusive thoughts, but attach different meanings and react in different ways. By controlling their reactions to intrusive thoughts, they can lessen/overcome their OCD. (Veale 2007)

One way in which people can analyse their thought, behaviour, and emotion patterns is the ABC model developed by Albert Ellis (1957). The model is a framework for users to gain an understanding of their behaviours by looking at:

A. The activating event – An event that is the source of some obsessive thinking. (e.g. touching a door handle in a public place)

B. Beliefs – The beliefs that occurred due to the event (e.g. feeling as though you have been contaminated and will become ill)

C. Consequence – The emotions and actions that follow on from the beliefs (e.g. anxiety and the compulsive need to repeatedly wash your hands)

Using the ABC framework is a method to break the link between thoughts and behaviours by emphasising to users that it is their beliefs about an event, not the event itself, that trigger their compulsive behaviours.

2.4 - Journaling

Journaling is often used as part of CBT and has been shown to have many benefits. For example, it is a useful tool in the process of connecting thoughts, feelings, and actions (Hubbs and Brand 2005) and helps with the mental processing that occurs after stressful events (Ullrich and Lutgendorf 2002). Also, Purdon et al. (2006) demonstrated that there was a significant correlation between the severity of someone's OCD and the amount to which they tried to suppress their obsessive thoughts, implying a direct benefit to patients who record their thoughts in a journal.

3 - Problem

Traditionally, physical paper journals are used during therapy. However, these do come with some drawbacks, as well as some challenges related to journaling in general. Summarised below are the issues identified in Haymen et al. (2012), King and LaRocco (2006), Stone et al. (2002) and Phipps (2005), with two of my own general ideas in addition in the last two rows.

Table 1 - Issues with paper journals and benefits of E-journals.

Issue with a physical journal	How an E-journal could solve this issue
Patients may not be able to take their physical journal everywhere with them as it could be cumbersome, they may also often forget to keep it on their person.[B]	The journal cannot be lost as the data would be stored in the cloud.[D] Nowadays it is normal to always carry your mobile phone with you, so patients are likely to always be able to make a journal entry.
Therapists may find it hard to read the patients handwriting, this may be exaggerated if the patient wrote their journal entry while in a panicked/ anxious state.[D]	Journal entries would be typed
Patients often forget or do not want to fill out their journal for a few days and then retrospectively fill out those days' entries. This is an issue as recall is not reliable and it changes the journaling exercise from a reflective exercise to a memory test.[C, D]	Some data such as timestamps can be automatically[D] included so that the therapist would be aware if journal entries had been retroactively filled out. Also, notifications could be sent to prompt users if they hadn't completed any journal entries for a certain amount of time.
Patients may feel exposed due to writing down personal/intimate details in the journal,[A] and because of the possibility that anyone could pick up and read a physical journal.	E-journals would be inherently more secure than a physical journal. The information would be stored behind a password so only the user and therapist would be able to see it.
The awkward nature of data transfer from a physical journal - patients would have to take pictures or scan their journal if their therapist wanted to see it before their session, or the therapist would have to read the journal at the start of a therapy session.	Therapist could have continuous access to their patients' journals, data can easily be shown the therapist on demand.[D,E]
Current stigma around mental health issues may mean that people feel self-conscious filling out their journal in public and skip journal entries.	Filling out an e-journal on a phone is unlikely to garner any attention in public.

A = (Hayman et al 2012), B = (King and LaRocco 2006), C = (Stone et al. 2002), D = (Phipps 2005), E = (Billings 2006)

3.1 - Existing solutions

There are many apps available for people undergoing CBT, as well as the general population, who wish to keep a journal of their thoughts and feelings. In order to identify key features of these apps and where they fall short, I have reviewed a selection apps available on the google play store.

Table 2 - CBT Thought Diary – Mood tracker, Journal & Record (Inquiry Health LLC)

Primary Functions	This app has two primary functions that are accessed from the home page, “Check In” and “Guided Journals”. **Check-In** Check-In is a quick way of inputting some data about your current feelings without having to do a full journal entry. To check-in you go through a series of 4 screens to enter: “how are you doing” rated on a series of faces, “what emotions did you feel” chosen out of a series of positive and negative emotions, “what activities have you been doing” chosen from a list, and then the option to type some text to elaborate if the user wants to. The ability to input data from a list of options makes the process quick and is a useful way to prompt users. While there is a good range of options, adding custom ones is a premium feature which limits the experience for non-premium users. This feature has little guidance for users not sure about what to enter; for example, the final screen is simply a textbox with the title ‘want to elaborate?’. Some stimulating questions or example entries might help users who are not used to writing about their thoughts, although considering the function is just for checking in perhaps the users are not expected to write much here. Overall, this feature fulfils its purpose as a simple and quick way to record your feelings, but could use slight improvements. **Guided journal** The Guided Journal function allows the user to make various types of journal entries, guiding them through each one using questions to which the user must type their response. The categories for the guides are: Thoughts & Feelings, Gratitude, Relationships, Goals & Values, and Take Action. Two guides, analyse thought and practice gratitude, are available to users without the premium subscription. Analyse thought takes the user through a series of steps to come to a deeper understanding of their mental process: 1. ‘What negative thoughts do you have?’ 2. ‘Did your thoughts contain any cognitive distortions’ with a list to choose from such as catastrophizing, emotional reasoning and self-blaming?’ 3. ‘How can you challenge your negative thoughts?’ 4. ‘What is another way of interpreting the situation?’ These questions can be a very useful function for users new to journaling, helping guide their thinking. Bringing up each question one step at a time means that users are less likely to find the task too dauting as they only have to consider one section at a time. Only having 2 of 28 guides available to non-premium users greatly limits the utility of this function, which is unfortunate considering that it is one of the main parts of the app. While this app is not primarily designed for OCD users, the ideas here could easily be adapted for this more specific purpose.
Secondary Functions	**Insights**

	As implied by its name, this function aims to provide users with an insight into their condition through the use of statistics. The statistics provided on this page are: Current Streak, Total Entries, Totals for Each Mood, Mood by Day, Mood by Time, Top Positive Emotions, Top Negative Emotions, Top Cognitive Distortions, and Post-Entry Feelings. Insights is a promising feature, and many of the statistics it provides could be interesting to users. However, a system that was able to draw together and compare different aspects of the users' data would provide much more insight. For example, comparing users' mood with activities undertaken or people met may be very useful to users trying to identify what events trigger certain emotional states. **Discover tab** The discover tab aims to inform the user about a variety of topics related to mental health. The function coves topics such as: tackling negative thoughts, cognitive distortions, stress, and relationships. Upon clicking on a topic information is provide through series of screens that you must swipe through. Most of the topics are only available to premium users of the app; if you are a premium user then this function fulfils its purpose and provides copious information in an easily digestible manner.
Additional comments	When you load each section of the app for the first time you are given a quick tutorial on how to use it. This improves app useability and makes it more accessible to users who are not very computer savvy. The premium version of the app costs £29.99 per year and provides these additional features: personalized insights, sync, backup and passcode, and custom emotions are listed as the benefits of pro version. Having so many features behind paywall greatly limits the useability of the app and could put off a lot of users.
Link	https://play.google.com/store/apps/details?id=com.moodtools.cbtassistant.app
Rating	Average rating of 4.5/5, 2786 total reviews
Reviews 1 star – "… The update is ugly, complex, and reduces functionality. It is now effectively an advertisement for the pro version." "Under how do you feel now your options are: better, a little better, worse than before. When you click on worse than before, a message pops up that say "oh thats a bummer.." It dismisses something serious and makes you feel horrible. …" 3 star – "Used to be great. But with the new update, I don't like it as much. U have to pay for a password protection now. I don't want people knowing what I write in case they use my phone." "The app is good but we can't edit the date to record past incidents, new entry always come with todays date only which is not editable. It also doesn't have the google drive or cloud back up" 5 star – "Helpful app that has been essential to my personal recovery of OCD. Highly recommend." "Easy to use and helpful. A nice alternative to journaling."	

Table 3 - Mind journal: Diary, Mood tracker & Gratitude (Bazimo)

Primary Functions	**Emotion** This function is a way for users to quickly add an emotion and an associated event to their day. From the main screen (journal tab) you can choose to add an emotion to the day. You then get to choose from a series of emotions with the ability to add your own custom emotions. Once an emotion is chosen you are then taken to a new screen where you are asked "what did you do?" with a series of possible activities

	such as work, sleep, or cooking, as well as the ability to add custom activities. At the bottom of this page there is also the option to add a note in a simple text box. Once an emotion has been added it is displayed on the current day's page in the journal tab, which is set out like pages of a physical journal. This function is a useful and simple way for users to quickly note how they are feeling, and the journal-style layout makes for a simpler transition for users switching from a regular journal to an E-journal. **Create a story** From the journal tab you also have the option to create a story, which is a way to add some more detail to your day than an Emotion entry in a set format. This function is a series of 5 pages that you swipe up to proceed through which ask the user: "how was your day?" using a round slider on a scale from awful to awesome, "how was your day? Events, emotions, feelings…" with a text box, your top 3 victories of the day, a final question that changes each time, and finally "would you like to give your story a name?". Examples of the final question are: "What are some fears I used to have? How was I able to overcome them?" or "If my worst fears were to come true, what would I do? Would I be able to handle that?". Firstly, the layout of this function is not ideal: the vertical scrolling is not very intuitive and sometimes cuts off the question, and round slider for inputting emotions is an unusual format. These design choices could be very frustrating to users with low technical knowledge. On the text input sections there is nothing in the way of guidance for the section and no example inputs. There is an audio guide for each section when using the app for the first time that explained the purpose and how to fill it out, which made the app feel very welcoming and personalised and would be of great help to users new to journaling. It would be a big improvement if this audio guide were always available, not just on the first use, and if it were also available in text format. The question at the end of this section that changes each time is an interesting idea which is potentially useful in keeping people interested in using the app and encouraging them to think in different ways, as it might otherwise get repetitive filling out the same questions every day. Another good feature of this function is the way it encourages users and is positive towards them. Firstly, when typing, the app has a progress bar encouraging the user to write a certain amount, saying "Type a bit more for a better effect", "Keep it Up!" and "You're done!". Secondly the app has the section for the user to fill out their top 3 victories of the day, which offers the user some positive support. Overall, this function does allow the user to add some basic information about their day, although the user interface can be clunky at times.
Secondary Functions	**Feed** The feed section aims to educate the user about general mental health wellness information. It achieves this with a simple user-friendly interface and by presenting the information in small sections with pictures. **Insights** This function contains reflection prompts designed to increase self-awareness, and guided meditations covering gratitude, anxiety, mindfulness and much more. As this is a premium feature I cannot comment on its effectiveness.
Link	https://play.google.com/store/apps/details?id=com.diary.journal
Rating	Average rating of 4.6/5, 9793 total reviews
Reviews 1 star – "Is not really working. You can't customise dates, so if you are writing your journal after midnight, it will automatically set the date, and you can't write for the next date."	

3 star – “Like the story concept and mood tracking but this app is lacking basic features. Unable to edit story title once saved. No option to add photos to stories. Don't see external saving or export options. Difficult to navigate & select the moods at times. UI is occasionally glitchy. Uninstalled.” 5 star – “I like this app it helps me a lot, I feel positive when something good happens and I write about it, when there's something bad then I feel light after writing. Its nice to have someone ask you how was ur day and remind u of ur little victories.”

Table 4 - Thoughts – CBT trainer and thought diary (Vivid mind)

Primary Functions	**Learn CBT** Clicking on learn CBT from the home page takes you to the learn CBT page. This page contains a list of topics (e.g. automatic thoughts, distortions, core beliefs) which when clicked take you through to another page with information about each topic. The information is presented in digestible sections alongside illustrations, often with concrete examples of each topic. The user interface is sometimes inconsistent; for example, the button to return to the list of topics sometimes says continue, sometimes says back, and on one occasion is missing. Also, many of the screens are slightly bigger than the phone screen meaning you have to scroll to see the continue/back button, but without any indication that you are able to scroll. These flaws in the design could make this section hard to use for users with low technological skills, and annoying for others. If the purpose of this function is to teach CBT, as the name suggests, then it falls short. However, this is probably a naming issue, and though the interface could be simpler and more consistent the function does provide some useful mental health/ mindfulness information. **Practice CBT** Although this function is called practice CBT, what it really aims to do is allow the users to express their feelings in a diary entry. This function contains two sections; a guide on how to use the diary, and the diary itself. The guide gives help on how to fill out each part of the diary with example entries. The diary pages talk the user through making a diary entry step by step across 3 pages. On the first page you enter the date, the event/trigger, how stressed you felt (on a slider), and what emotions you are feeling now. On the next page you enter your thoughts in response to the event, any cognitive distortions you can identify from a list, and an alternative perspective you can take on the situation. On the final screen you can enter your behaviour, your distress after your actions (on a slider), and if there is anything else that you could have done to make the situation better. The how to use the diary page is a useful part of this function, and the explanation and example inputs could be very helpful to users who are either unsure what to write or lacking confidence in their writing. It would be an improvement if this help was available as you went through the process of making a journal entry, as with the current setup if a user wanted to look at the help whilst still completing a diary entry, they would lose their progress by doing so. For the process of making a diary entry, it is helpful that the inputs are spread out over three pages and that the titles of the inputs are descriptive. For example, rather than an input box saying ‘thoughts’ the title is “Identify the thought that arose in response to the event. Identify the thoughts that are leading to the emotions or feelings that you entered”. This is helpful for users who are struggling with what to write and may help them to get the most out of the journaling process.

	There are some design and ease of use improvements that could be made to the diary entry screens. Firstly, the cursive font (also used on the home page) used for some of the headings in yellow on a white background makes it difficult to read. Furthermore, the text input fields only have a border along the bottom, meaning it is not always obvious where to click to enter text. Finally, the titles for the inputs are placeholder text inside the textboxes that disappears once you start typing, something that would be inconvenient for users who would like to refer back to the prompts in the title. Altogether this function succeeds in allowing the user to make diary entries, and helps them to do so with useful guidance and thought-provoking question covering a large range of topics. However, the user interface could be improved by making it clearer where to type, by not having the titles as placeholder text, and by making some of the text easier to read.
Secondary Functions	This app contains no other functions.
Link	https://play.google.com/store/apps/details?id=org.thevividmind.thoughts
Rating	Average rating of 4/5, 35 total reviews
Reviews 5 star – “Great job. Examples are very helpful to understand the concepts.” 4 star – “Great! Although the explanations of the 'distortions' would be nice to have an option to read it dor longer, as the explanations only seem to show for a quick second before they disappear.”	

Table 5 - CBT Diary (Continuum)

Primary Functions	**Record Event** The record event function allows the user to enter: Time, Date, Event, Body Sensation, Emotions, Behaviour, Automatic Thoughts, Belief, Automatic Thought, Circumstances and Remarks. This function succeeds in its basic purpose that a user can record an event in a structured and analytical manner; however, there is no explanation of how to fill out the form, why they might want to, or any example inputs. **Record Thought** Recording a thought allows the user to enter: Time, Date, Thought, Belief, Body Sensation, Behaviour, Circumstances and Remarks. As with the record event function, this function would be useful to someone looking for a very simple diary app but not for a new user who needs guidance. **Record Emotions** This function provides a way for the user to rate 9 emotions from 0-9 by simply typing the number in each respective box. There is a method for the user to add custom emotions in the settings, although this would be more convenient if it were contained in the emotions function. The function is a successful basic way for the user to quickly record their emotions.
Secondary Functions	**Show Charts** This function is a method for visualisation of the users’ data. On the first screen you must enter chart type (Pie Top Emotions, Line Top Emotions, Pie All Emotions, Line Chosen Emotions) and the start and& end day or period. Showing the user’s emotions over time is a great addition to the app that could be very useful in tracking their condition. **Generate Report**

	The generate report function allows the user to choose a time period from which to extract the data and then generates a report of their entries. The ability to produce a report from your diary entries could be very useful in a therapy setting where the user needs to regularly share their entries with a therapist. As this function is behind a paywall I cannot comment on its effectiveness. A license can be bought for 3 months, 1 year or 99 years, with the cheapest option being £36.49 for 99 years.
Link	https://play.google.com/store/apps/details?id=pl.com.continuum.cbtbasicemotionsdiary
Rating	Average rating of 4.2/5, 256 total reviews
Reviews 1 star – "I became so sick of the constant alerts that I eventually had to uninstall it. I'd not even used it, and tried changing the settings innumerable times with no joy. Way too much hassle and annoying!" 3 star – "…However there are times now I rely want access to it on my laptop/desktop so that I can record events faced at work with angry emails etc. " 5 star – "I appreciate the simplicity of the app. It brings some discipline into my daily journalling and I can always access my information no matter where I am. I can also see how my emotional states progress throughout a week and re-read my previous entries" "Exporting option is a great plus for sharing your records (probably to a therapist) in a spreadsheet. My only complaint: pressing the help button while having partially filled a record seems to cause your record getting lost" "…I really like how I can add thoughts, emotions, etc. instead of relying on the default mode…"	

3.2 Summary of App Reviews

Common Primary Functions –

Rating emotions:

Three of the apps have a function where the users can quickly add an entry where they rate their emotions. In all of the app's users could choose emotions from a series of options, often with accompanying emoticons. This speeds up data entry for users significantly in comparison to typing out each emotion they want to enter, and having the visual aspect of the emoticons can also work to help users identify how they're feeling in a way that a blank text box might not. The CBT Diary app was the only one to make the users rate their emotions out of ten rather than simply choosing which emotions they felt; this method provides more insight into the user's state of mind without requiring much more effort. CBT Thought Diary and Mind Journal both have additional questions about the activities the user has been doing chosen from an editable list - again this is useful to speed up data entry and simplify the experience for the user. These extra questions also add more purpose to this function, as recording that you felt happy after a certain activity or anxious after another is much more useful and insightful than simply recording that you were happy or anxious.

Diary Entries:

All of the apps had methods for adding various forms of diary entries. There were a few common themes where this function was implemented well. The first of these was spreading out form inputs over multiple pages. This allows the user to focus on one part of the process at a time without becoming overwhelmed by the whole process. Also, for analyses such as the ABC model with defined different areas of analysis, having these sections split across separate pages could help users to separate them mentally. The second theme was helpful guidance on journal entries. This ranged from the guided journal entries on 'CBT Thought Diary' to the audio guides on 'Mind Journal' and the example inputs on 'Thoughts'. Each of these features makes for a comfortable, supportive experience for users, something which is very important in an environment where users may feel anxious and exposed about expressing their intimate thoughts. Finally, a simple UI was a key aspect to successfully implementing this function. Mind Journal was the only app to have problems with the UI such as dodgy scrolling, sometimes cutting off questions, and circular sliders. Issues such as these may be very frustrating to users with low levels of technical knowledge and could put them off using the app.

Common Secondary Functions –

Mental Health Information:

Three of the apps had a section which included information about mental health. Each of these was laid out well and presented the information to the users in readable chunks accompanied by illustrations. For my purposes, information about cognitive distortions will be the most important to include as it gives the users the means to identify cognitive distortions in their beliefs about their mental intrusions.

Statistics:

CBT Thought Diary and CBT Diary both have a function for displaying some summary statistics to the user based off their journal entries. CBT Diary have implemented this feature slightly better as it provided a way for users to visualize how their emotions have changed over time, whereas CBT Thought Diary mainly listed cumulative statistics such as top emotions and top cognitive distortions.

Missing/ poorly designed features:

- Lack of continuity between different sections of the app. For example, in 'Thoughts' the help is all available on one page so if you want to view the help halfway through a journal entry you have to discard your progress to be able to go to the help page.
- Only CBT Diary had the ability to extract data from the app to share easily with a therapist through its create report feature. This would be very important for users using the app in a therapy setting, and solves one of the problems identified in section 2.4 with physical journals.
- None of the apps have a way to search through past journal entries.

Additional well-designed features:

- On the first time loading the app a tutorial explaining how to use the app is useful to help introduce users to the app.
- The forms on all of the apps had very few required fields, often just the first one. This could be very useful to users who are not feeling up to writing a full diary entry and just want to get some information down and fill out the details later.

3.3 - User Personas

In order to keep my work on track I have written the following 3 user personas that capture the main user groups my application is intended to serve. Keeping these personas in mind through the development process will help to focus my work on the important aspects that serve my user's needs, and allow me to think of the product through not just my own perspective.

Patient Persona 1

Name: Edward Shaw

Description: Age 20, he has just started with therapy and has a low level of knowledge about his condition and OCD in general. He has moderate OCD, with obsessive thoughts about contamination which leads to compulsive cleaning behaviours. Edward has a high level of technical knowledge.

Needs/ aims: Edward has no previous experience of journaling so is worried he might not know how to fill out the journal or might not stick to journaling regularly. For these reasons he wants an E-journal that can give him some guidance on his entries and encourage him to

use it regularly, as well as the benefit of always having it with him on his phone. Through journaling he wants to gain an understating into his mental processes, e.g. thought patterns and cognitive biases, to help with their CBT.

Patient Persona 2

Name: Emma Cook

Description: At age 40, Emma has been undergoing therapy for a number of years and is familiar with her condition and thought processes. She has severe OCD, mainly related to obsessive checking due to a fear of leaving their doors unlocked, leaving the oven on etc.

Needs/ aims:

Emma has been journaling every day for a while with a physical journal but is looking to switch to an E-journal as she wants to be able to easily share her journal with her therapist, view statistics and visualisations based on her journal entries, and easily search through previous journal entries. Emma has a low level of technological knowledge so is looking for an E-journal that is easy to use with simple interfaces.

Therapist Persona

Name: Eve Brown

Description: Has been a therapist carrying out CBT with people with OCD for a number of years.

Needs/ aims: Eve finds that her patients often forget their physical journals or don't want to fill them out in public. She would therefore like to start using E-journals with them but there are currently no E-journal applications designed specifically for OCD. Also, she finds it cumbersome to either have to be sent pictures of journals or read through her patient's journals in her therapy sessions. She would therefore like an E-journal that is tailored to OCD and allows her patients to easily share their data with her.

3.4 - OCD Diary Research

To be able to understand what needs to be included in my application to meets my user's needs, some examples of CBT diaries and exercises for OCD must be reviewed.

In 'The OCD Workbook' (2006) Hyman and Pedrick describe a method of cognitive restructuring designed to challenge participants' faulty beliefs by identifying the cognitive

errors at work, and writing a more realistic appraisal of/ response to the situation. The exercise requires the participant to record 8 items:

1. Activating event
2. Intrusive thought
3. Discomfort on a subjective units of distress scale (SUDS)
4. Faulty belief
5. How much they believe their faulty belief to be true (0-100%)
6. What type of cognitive errors are at work
7. Realistic response or coping statement, used to talk back to the OCD
8. How much they believe the realistic response to be true (0-100%)

An example of this exercise from the book is:

1. Touching a handle in a public bathroom with bare skin
2. What if I contract a disease?
3. 50
4. I will become sick if I do not act
5. 40%
6. Overestimating risk, intolerance of uncertainty
7. If I don't act on my need for absolute certainty, the urge to do a ritual will diminish. I must learn to take a chance in order to get better
8. 50%

This exercise is intended to show the participants that they have a choice in the way that they interpret their thoughts, that doing so can relieve the symptoms of their OCD, and that by analysing their thoughts they can become more objective the next time they experience them and learn to control their reactions.

Clark (2007) highlights the importance of recognizing an additional step in thinking past analysing the faulty beliefs. Clark proposes that in addition to misinterpreting their obsessions, people with OCD also falsely evaluate their efforts to control their obsessions and the perceived consequences of failing to control their obsessions. Expanding on this idea, he proposes that these false appraisals of thought control are a key factor in the frequency of obsessions, and that there are two ways to reduce the reaction to unwanted mental intrusions. Firstly, the recognised route of interpreting the reaction differently, and secondly through adapting the appraisal of failed thought control. One example Clark gives of this thinking is a religious individual who has unwanted sexual obsessions; they may interpret their lack of/failure to control these obsessions as proof that they are of a sinful nature. In this case and the others Clark highlights, individuals are continually struggling to control their obsessions due to their falsely perceived consequences/implications of not doing so.

Clark also gives an example of a similar exercise to that of Hyman and Pedrick (2006), in which individuals must note: date/ time, situational trigger, obsession, interpretation of importance, and main appraisal patterns (cognitive errors). This exercise is used to teach patients to separate their faulty interpretations of their obsessions from the obsessions themselves, then to challenge these faulty appraisals.

Two additional diary exercises (figures 1 and 2) studied also included questions about triggers, emotional response, obsession/worry, reaction/compulsion, alternative response, and the outcome.

Trigger	Response	Obsession	Compulsion	Realistic Assessment	Outcome
Identify the triggering situation, **intrusive** thought, image or initial feeling	Identify the negative response and feelings - Rate intensity 0-100%	Describe the worry or obsessional thoughts that follow the trigger or intrusion	Describe the safety behaviours, neutralising habits or mental routines	Identify a helpful alternative interpretation or response	Re-rate emotion 0-100%

Figure 1 - Example OCD thought record sheet (Think CBT 2017)

Situation & Trigger	**Emotion/s** Rate 0 – 100% **Physical Sensations?**	**Initial thought, image, doub, feeling, worry** **Meaning of the initial thought or image.** What might happen?	**Alternative response** What would be a healthier more balanced perspective?	**What did I do?** How long for? How many times?	**What's the outcome?** **What could I do or have done instead?** Defusion technique? What's the best response? **Re-rate Emotion**

Figure 2- Second example OCD though record sheet (Get self-help 2010)

Based on these observations, I will prompt users to write about: activating events and the associated intrusive thoughts, the following obsessions and associated anxiety, how much they believe their evaluation of the situation to be true, any cognitive errors they can identify

in their thinking, and a more realistic assessment of the situation. In the final section I will include information about Clark's second step of thinking to get the users to also think about their lack of control over their obsessions.

3.5 - Constraints

There were two main constraints on the scope of this project; the amount of time, and my knowledge going into the project.

In total there were 14 weeks to complete the project. This included all aspects of the project development lifecycle (research, design, implantation, testing and evaluation). Fitting all of these stages into 14 weeks is a tight schedule, and limits the number of features than can be included in the end product. To help mitigate this issue a plan was created at the start of the project to keep each major part of the work within certain timeframes.

At the start of the project, I had very little knowledge about two important aspects of the work; OCD and React Native applications (my chosen application framework as will be discussed in later sections). My lack of knowledge of OCD meant that during the research phase I often had to do extra reading to understand many of the common terms and themes in the literature, and had to make sure that what I was reading was consistent with the current schools of thought. My lack of knowledge of React Native applications and lack of experience developing them firstly slowed down the implementation phase of the project, as I was tackling many problems for the first time without prior experience to draw on, and secondly increased the difficulty of the design phase as I was sometimes unsure of the possibilities and limitations within React Native.

4 - Approach

4.1 Development methodology

I will use the Kanban board development methodology for the implementation phase of this project. Kanban boards are an agile project management tool which use three sections to organize the work; to do, in progress, and complete. By splitting up the development work into small individual tasks they can be easily organised on the board in order to visually track their progress, along with the overall progress of the work. Further, as an agile methodology this allows for more tasks to be easily added if they arise as the work progresses. This is useful to an individual like myself who has limited experience planning large projects, as if I were using a non-agile methodology such the waterfall process, I would not be able to adapt the project throughout the process.

My development will focus initially on core features of the app defined as the 'must have' features in the requirements definition (Section 5). Only once these features have been implemented will I move onto the 'should have' and 'could have' features of the application.

To keep my progress on track, at the start of the development phase I will plan a timeframe for when I intended to complete each feature. This will allow me to keep within the strict timescale for the project and give me many smaller deadlines throughout the process to keep me motivated.

4.2 Mobile app vs website

I have chosen to produce a mobile app, rather than a website, as they are generally more responsive and easier to use than mobile websites. As users may want to write in their diary 5+ times a day, it is far more convenient to use an app installed on their phone than it is to have to load a website every time. (Diduh 2021)

4.3 Technologies chosen:

4.3.1 Framework

To create my application, I will be using React Native. React Native is a framework that allows you produce apps for both android and IOS devices using one language (JavaScript), as opposed to creating each platform's app in a different language (Swift for IOS and Kotlin/Java/C++ for android). This is a great advantage for this project, as it would have been unfeasible to create an app for each platform due to time and knowledge constraints. However, with React Native, I can effectively double the potential user base of the app. React

Native also integrates well with the libraries I plan to use for the back-end of the app; Node.js, Express.js, and MongoDB, as they all use JavaScript and JSON data.

4.3.2 Database

As my application will require the storage of information such as usernames, passwords, and diary entries, I will need a way to securely store all this information. For this need I will be using Node.JS, Express.JS, MongoDB, and mongoose.

MongoDB

MongoDB is a NoSQL database that stores documents with a JSON style notation (Taylor 2021). This is my preferred method over using a SQL database, as I will be able to use JSON data throughout the application, thus simplifying my workflow. To host the database I will be using MongoDB Atlas, where a free account provides between 512MB to 5GB of storage.

Node.js and Express.js

Node.js is a JavaScript runtime environment that was built to facilitate the running of JavaScript programs outside of a browser (Patel 2018). Express.js is a web framework built for node which allows the handling of http requests using routes and http verbs.

Mongoose

Mongoose is an object modelling library that provides a layer of abstraction on top of MongoDB. In a mongoose schema you define the properties of a MongoDB collection. For the properties you can define attributes such as data type, whether they are required fields, and whether they are unique fields.

Node.js, Express.JS, and mongoose will be used to connect the application to the database by using a series of routes to handle different requests. For example, if the front-end made a GET request to the URL /:user/entries, there may be a express.js route that uses mongoose to retrieve all the diary entries from the database for that user and then sends them back to the front-end.

4.3.3 UI

When designing my UI, I will be incorporating Google's material design standards where I see fit. This will mean that the design of my app is in line with what users are used to experiencing, thereby minimizing the time it takes for users to feel comfortable using my app. When implementing the UI I will be making use of the npm package react-native-paper. This package

provides a collection of React Native elements that can be used to assemble a consistent UI in accordance with the material design principles.

4.4 – Learning resources

Prior to starting the development work on the project, needed to acquire new skills and learn to use the chosen technologies. First of all, the tutorial in the React documentation was followed (React [no date][c]). This tutorial gives an introduction to react concepts such as props, state, class and functional components. Secondly, to learn the fundamentals of node.js and express.js a YouTube tutorial by “freeCodeCamp“ was used (freeCodeCamp.org 2021). To learn how to set up a mongoDB database and integrate it with node.js and express.js a YouTube tutorial by “The Net Ninja” was used (The Net Ninnja 2020). To supplement these resources a further set of two videos demonstrating the creation a full stack app were used (JavaScript Mastery 2020a and 2020b). Finally, when starting to code, the introduction guide in the React Native documentation was used (React Native 2021).

5 – Products

5.1 Requirements

Based on the reviews of existing solutions and OCD exercises in the literature, the following functional (table 6) and non-functional (table 7) requirements for my solution have been defined

Table 6 - Functional Requirements

No.	Requirement	Acceptance Criteria
Must have		
1	The system will allow patient users to write a diary entry	Patients have the option to add a new diary entry, which on completion saves the entry to a database
2	In the diary entry, the patient can include information about an activating event, associated intrusive and obsessional thoughts and their related anxiety	Within a diary entry, the user is prompted and able to enter information about an activating event, associated intrusive and obsessional thoughts and their related anxiety
3	In the diary entry the patient can include information about cognitive errors and their faulty beliefs	Within a diary entry, the user is prompted and able to enter information about cognitive errors and their faulty beliefs
4	In the diary entry the patient can include information about alternative ways to interpret their intrusive thoughts	Within a diary entry, the user is prompted and able to enter information about alternative ways to interpret their intrusive thoughts
5	In the diary entry the date and time should be automatically filled, although the user should be able to edit them	Within a diary entry the date and time are automatically set to the current values, the user can choose to edit them if they wish
6	The system will allow patient users to search through their previous entries	Patients can enter a word or series of words and the system will return the diary entries matching all or parts of the string
7	The system will allow patient users to view all the details of a previous diary entry	Patients can choose a specific diary entry from a list of all their entries, and view the details of that entry
8	The system will allow patient users to edit their previous diary entries	Patients can choose to edit any of their previous diary entries, upon saving their edits the database will be updated with the new version
9	The system will allow patient users to view their previous entries and filter and sort by date	Patients can choose to view their diary entries and see a list of all diary entries. Patients are able to choose a date or range of dates to filter the entries by and can sort by ascending or descending chronological order
10	The system will allow user to select entries to become private, which their therapist can't see	Patients can specify when creating a diary entry, or in retrospect, that the entry should be private. This entry will then not be visible to their therapist
11	The system will require all users to create an account with an e-mail and password	On first us of the application the system will require users to create an account, users will not be able to access the application without being logged into an account

12	The system will allow users to change their password	While logged in, a user can change their password by entering their old password and new password, which will then be saved in the database
13	The system will allow patient user to perform a check-in	Patients have the option to check-in, which on completion will be saved into the database
14	In the check-in the patient can choose and rate their emotions from a list	Patients can select any number of emotions from a list of options and rate the strength of each of these emotions on a scale of 1-10
15	The system will allow patient users to edit the list of emotions	Patients can remove default emotions and add or delete their own custom emotions
16	In the check-in the patient can choose what activities they have been doing	Patients can select any number of activities from a list
17	The system will allow the patient users to edit the list of activities	Patients can remove default activities and add or delete their own custom activities
18	In the check-in the patient can add a note	Patients can type a text note that is saved with the check-in
19	The system will allow users to view and edit a previous check-in	Patients can view a list of their check-ins and can choose to edit any of these, upon saving their edits the database will be updated with the new version
20	The system will allow therapist users to view their patients	Therapists are able to see a list of all of their patients
21	The system will allow therapist users to add/delete patients to their list	Therapists are able to add new patients to their list and delete patients they are no longer seeing from their list
22	The system will allow therapist users to see and search through all of the non-private entries of each of their patients	Upon selecting a patient, the therapist can see a list of all of their non-private diary entries. The therapist is able to search this list using search terms and sort the list by date
Should have		
22	The system should have a function to provide information about cognitive errors to the patient users.	Patients are able to view a list of cognitive errors, upon clicking on an error the user is taken to a page with information about that error
23	The system should have a function to display summary statistics to the users	Patients can choose to see summary statistics based on their diary entries, such as common emotions, activating events, or cognitive distortions
24	The system should have a function to display useful comparisons to patient users.	Patients can view useful comparisons drawn from their entries, for example a graph of their emotions over time or a list of which activities produce the most anxiety
Could have		
25	In a diary entry, the patient user could be able to tag their location	Patient is able to add their location to a diary entry either through their devices location or by typing a location
26	The system could send a notification to the patient user if they have not done an entry in an amount of time specified by the user	Patient can choose to be sent a reminder notification after their chosen time period of no diary entries. A notification prompting the user will be sent after the time period

27	The system could let patient users save common items such as people and places so they can be referenced between diary entries	User can tag common items and use them when writing their diary
28	In a diary entry, the patient user could be able to add a photo	Patient is able to attach a photo to a diary entry either through taking a photograph or choosing a photo from their device

Table 7 - Non-Functional Requirements

No.	Requirement	Measure
Must have		
1	The system must provide example inputs for each section of the diary	On each diary input section, patients will be able to view an example input for that section to demonstrate the type of thing they might want to enter
2	The system must explain the purpose of each section of the diary	On each diary input section, patients can view some text that explains the reasons behind and purpose of that section
3	The system must be easy to use, navigate, and understand	A heuristic evaluation of the UI will be carried out to evaluate the useability of the system
4	The system must be reliable	Test cases will be produced and carried out to ensure the system functions correctly
Should have		
5	The system should be fast to respond to user input	Other than actions that interact with the database, the system should respond very quickly (<1 second) to user input

5.2 Use cases

The following use cases cover the main tasks that each type of user will be able to undertake within the app.

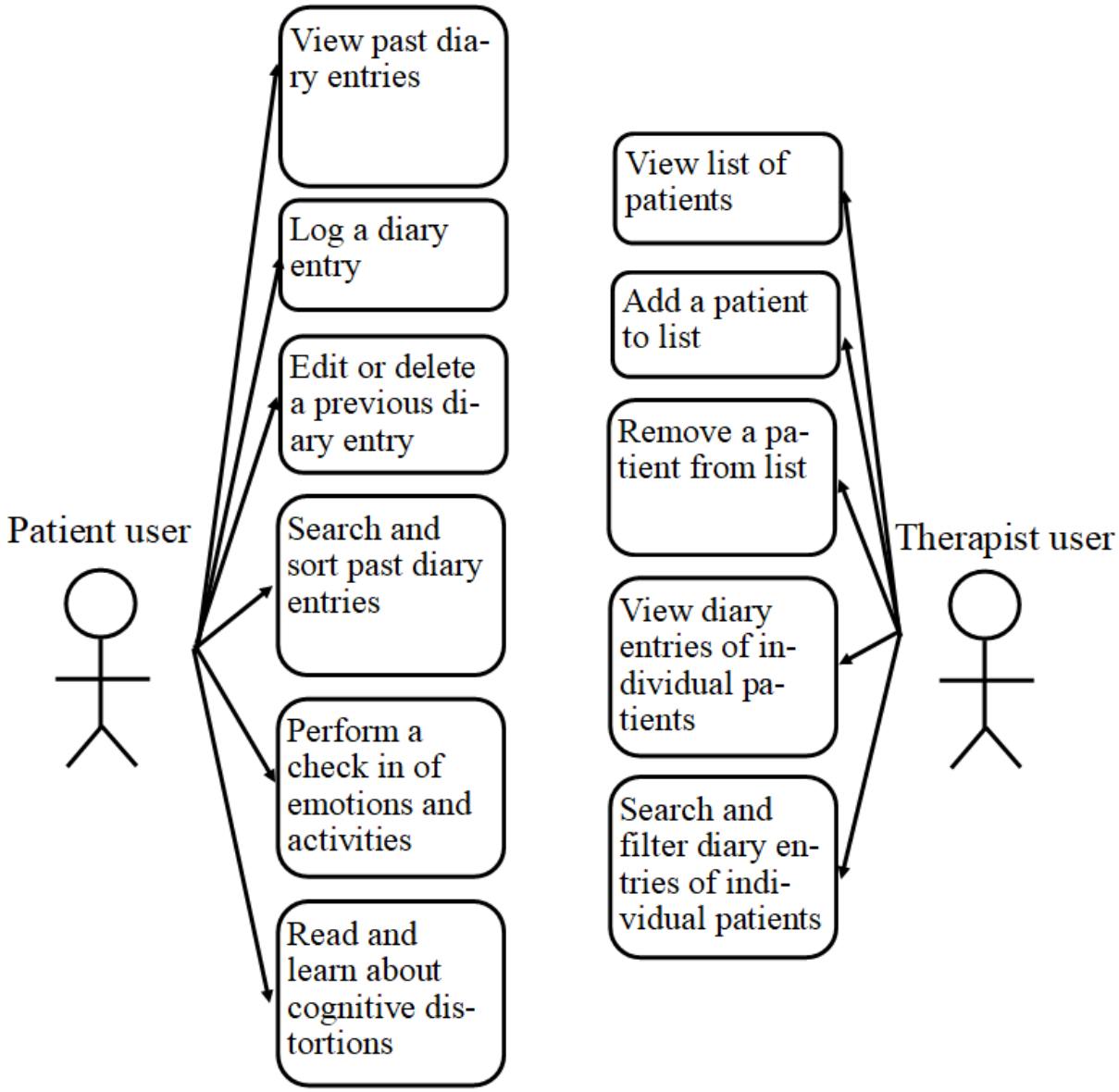


Figure 3 - Use case diagram

Patient Use Cases

Table 8

Name: Log a diary entry	Number: 1
Description: Users can log a guided diary entry	
Pre-conditions and Trigger: User must be logged into a patient account	
Main Flow: 1. User selects “New Diary Entry” on the main menu 2. User enters title 3. User enters data and time (optional, as these are auto filled to current values) 4. User enters activating event and intrusive thoughts	

5. User enters their obsessions 6. User rates their anxiety on a slider 7. User clicks next page button 8. User rates their belief in their evaluation on a slider 9. User chooses cognitive errors from a list 10. User enters identified faulty beliefs 11. User clicks next page button 12. User enters alternative interpretation of situation 13. User selects radio button if they would like the entry to be private 14. User clicks submit entry button
Alternative Flow: User can click the hamburger menu and click save and exit at any time after they have entered the title to exit the flow back to the main menu.

Table 9

Name: Edit or delete a previous diary entry	Number: 2
Description: From viewing a list of diary entries the user can choose to edit or delete a diary entry	
Pre-conditions: User must be logged into a patient account and have completed 1 or more diary entries.	
Main Flow: 1. User selects “view search and filter entries” from the main menu 2. User shown list of previous diary entries 3. User clicks on delete button next to their chosen entry 4. User click confirm and the entry is deleted	
Alternative Flow 1: 1. User selects “view search and filter entries” from the main menu 2. User shown list of previous diary entries 3. User clicks on edit button next to their chosen entry 4. System displays their previous inputs 5. User edits their desired sections and click save	

Table 10

Name: Perform a check-in of emotions and activities	Number: 3
Description: Users can do a quick check-in where they can choose emotions from a list and activities from a list and add a text note.	
Pre-conditions: User must be logged into a patient account	
Main Flow: 1. User selects “Check-in” from the main menu 2. User chooses the emotions they are feeling from a list of emotions 3. User chooses the activities they have been doing from a list of activities 4. User enters a text note 5. User click submit	
Alternative Flow 1: 1. User opts to add a custom emotion at stage 2 of the main flow by clicking add emotion 2. User enters name of emotion 3. User chooses an emoticon to represent the emotion	

4. User clicks save and returns to main flow Alternative Flow 2: 1. User opts to add a custom activity at stage 2 of the main flow by clicking add activity 2. User enters name of activity 3. User chooses an emoticon to represent the activity 4. User clicks save and returns to main flow

Table 11

Name: Read and learn about cognitive distortions	Number: 4
Description: User is able to read about common cognitive distortions that may be present in their thinking.	
Pre-conditions: User must be logged into a patient account	
Main Flow: 1. User selects “Learn about cognitive distortions” from the main menu 2. System displays a list of cognitive distortions 3. User clicks on a cognitive distortion 4. System displays information about the chosen cognitive distortion	

Table 12

Name: View past diary entries	Number: 5
Description: Users can view can a list of their diary entries, where they can select individual entries to view.	
Pre-conditions: User must be logged into a patient account and have completed one or more diary entries.	
Main Flow: 1. User selects “View search and filter entries” from the main menu 2. User shown list of previous diary entries 3. User clicks on a chosen entry 4. System displays all the information from that entry (from here they can edit or delete an entry, use case 2)	

Table 13

Name: Search and sort past diary entries	Number: 6
Description: Users can search through their diary entries or filter them by date.	
Pre-conditions: User must be logged into a patient account and have completed 1 or more diary entries.	
Main Flow: 1. User selects “view search and sort entries” from the main menu 2. User enters search terms into search box and clicks search 3. System returns the list of entries matching the search terms	
Alternative Flow: 1. User selects “view search and sort entries” from the main menu 2. User chooses sort newest or oldest first 3. System returns the list of entries in the chosen order	

Table 14

Name: View summary statistics based off inputs	Number: 7
Description: User can view summary statistics, based on their check-ins and diary entries, such as most common emotions and most common cognitive distortions.	
Pre-conditions: User must be logged into a patient account and have completed one or more diary entries	
Main Flow: 1. User selects “View Statistics” from main menu 2. System displays statistics to user	

Therapist Use Cases

Table 15

Name: Add a patient to list	Number: 8
Description: Therapists add a new patient to their list	
Pre-conditions: User must be logged into a therapist account	
Main Flow: 1. User selects “Add Patient” from main menu 2. User enters email of patient they wish to add to their list 3. If the email is registered as a patient account the associated user is added to the list. If there is no associated account the user is prompted to check the email is correct.	

Table 16

Name: View list of patients	Number: 9
Description:	
Pre-conditions: User must be logged into a therapist account	
Main Flow: 1. User selects “View Patients” from main menu 2. System displays a list of all the patients for the therapist currently logged in	

Table 17

Name: Remove a patient from list	Number: 10
Description: Therapists can delete patients they are no longer seeing from their list	
Pre-conditions: User must be logged into a therapist account and have one or more assigned patients	
Main Flow: 1. User selects “Remove Patient” from main menu 2. User enters patients name or e-mail and clicks “Remove Patient” 3. User clicks Confirm and patient is removed from their list	
Alternative flow: 2. User enters patients name or e-mail and clicks “Remove Patient” 3. System displays message saying “no user found with that name/e-mail” 4. User re-enters correct name/e-mail 5. User clicks Confirm and patient is removed from their list	

Table 18

Name: View Diary entries of individual patients	Number: 11
Description: Therapists can select a user and view their diary entries	
Pre-conditions: User must be logged into a therapist account and have one or more assigned patients	
Main Flow: 1. User selects “View Patients” from main menu 2. System displays a list of all the patients of the therapist currently logged in 3. User clicks the “View Diary” button next to the desired patient 4. System displays a list of the patient’s diary entries 5. User can select a diary entry to view all the information	

Table 19

Name: Search and filter diary entries of individual patients	Number: 12
Description: Therapists can select a user and view their diary entries	
Pre-conditions: User must be logged into a therapist account and have one or more assigned patients	
Main Flow: 1. User selects “View Patients” from main menu 2. System displays a list of all the patients of the therapist currently logged in 3. User clicks on “View Diary” button next to the desired patient 4. System displays list of diary entries for that patient 5. a. User enters search terms and clicks search b. User enters date or data range in filter tab and clicks filter 6. System displays searched/filtered list of diary entries	

Table 20

Name: View summary statistics of individual patients	Number: 13
Description: User can view summary statistics of their patients, based on their check-ins and diary entries, such as most common emotion and most common cognitive distortions.	
Pre-conditions: User must be logged into a therapist account and have one or more assigned patients	
Main Flow: 1. User selects “View Patients” from main menu 2. System displays a list of all the patients of the therapist currently logged in 3. User clicks on the “View Statistics” button next to the desired patient 4. System displays statistics for the chosen patient	

6 – Design

To ensure a high standard of design for my application, wireframes of the UI have been produced to show how the user will be able to interact with the system. These wireframes were produced with the use cases and requirements in mind to guarantee all of the functions of the application can be carried out. The wireframes will be explained and then subject to a heuristic analysis according to Nielsen's 10 heuristic principles, after which any identified improvements will be made.

6.1 – Colour

In accordance with the material design principles, I will make use of a primary colour (with multiple hues) as the main theme for the app, and a secondary colour to bring users attention to important areas of the application. The chosen colours are shown below in figure 4. Blue was a common colour in the apps reviewed in section 3.2, and has been chosen as it is commonly associated with feelings of tranquillity or calmness (Cherry 2019). Light orange will be used as the secondary colour as it contrasts with the blue, and together they are suitable for colourblind users (Shaffer 2016). The secondary colour will be used to highlight buttons to the user; by giving them all a common theme they will stand out and provide a consistent experience throughout the app.

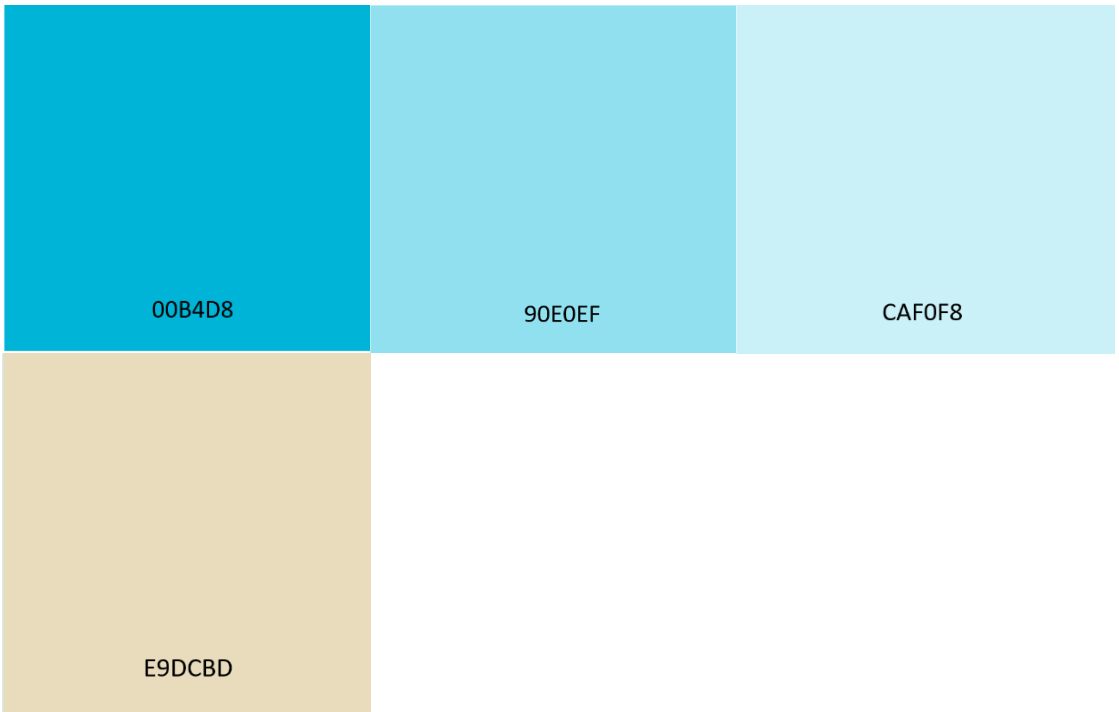


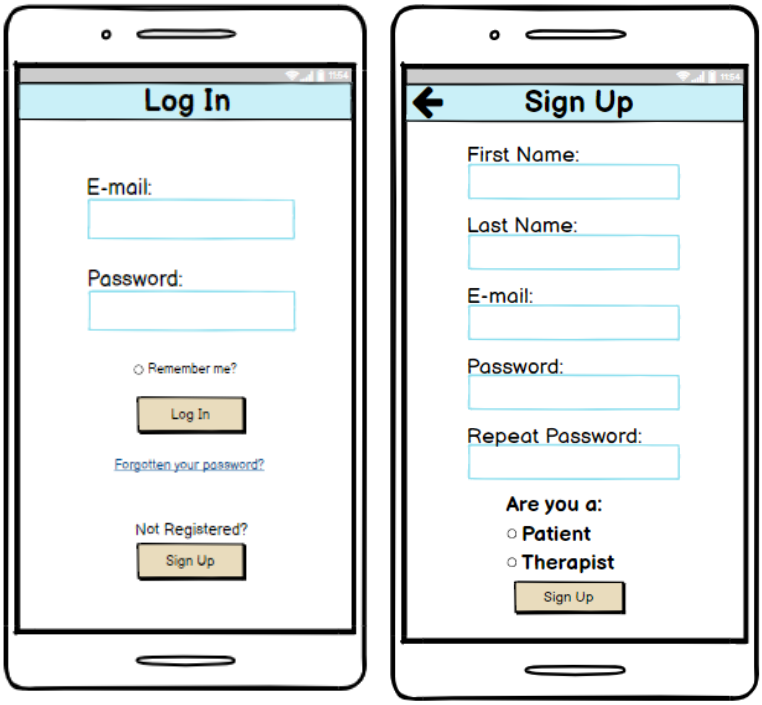
Figure 4 - Colour scheme

6.2 – Initial UI Wireframes

Note: images are represented by placeholder icons of a square with a cross

Patient and Therapist screens

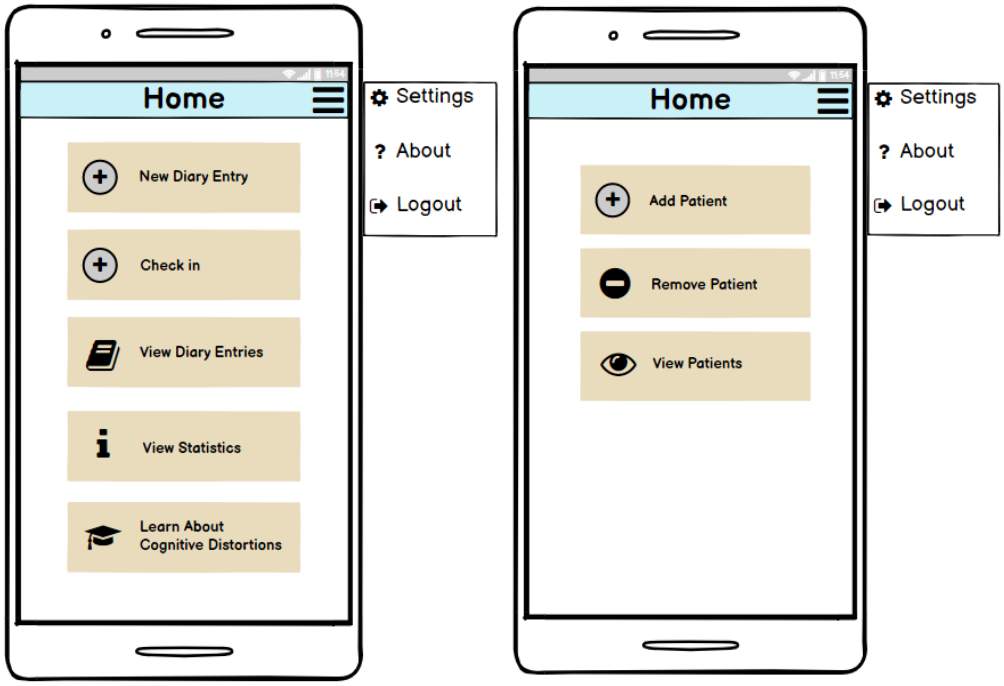
Table 21 – Log in and sign up screens



Screens 1 and 2

The screen on the left is the log in screen where users will log in to the application with their email and password. Users can also choose to not have to log in each time they load the app by selecting the 'remember me' radio button. This is the first screen a user will see when loading the app for the first time. If a user is not registered, they can click the sign-up button, which will take them to screen 2, the sign-up screen. On this screen user can enter their name, email, password and choose whether they are a patient or therapist to create an account.

Table 22 – Patient home and therapist home

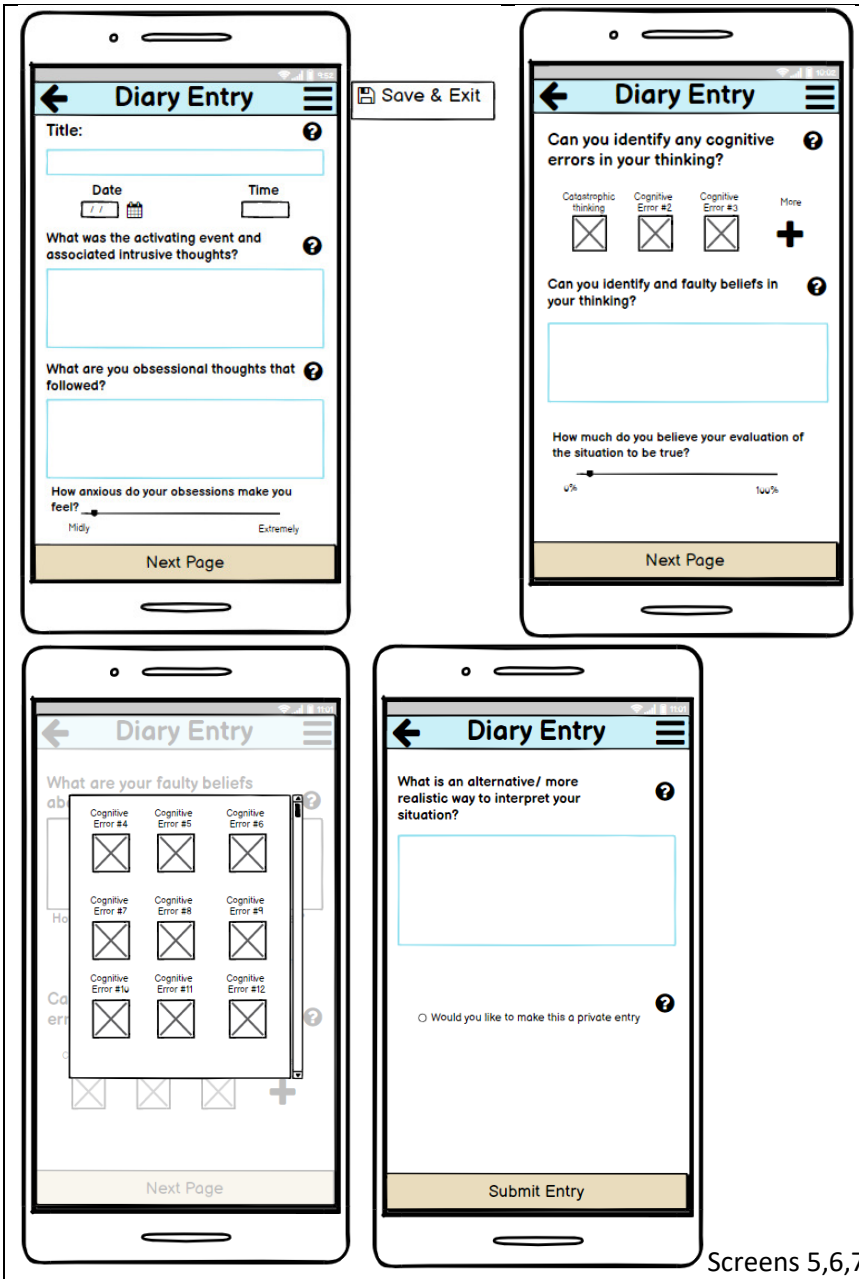


Screens 3 and 4

These screens are the home pages for the patient and therapist accounts, on the left and right respectively. These are the home screens that users will see once logged in. From here users can access all the main functions of the app. Clicking the hamburger menu icon in the top right brings up the menu shown above, where users can access the settings, about page, and logout.

Patient's Screens

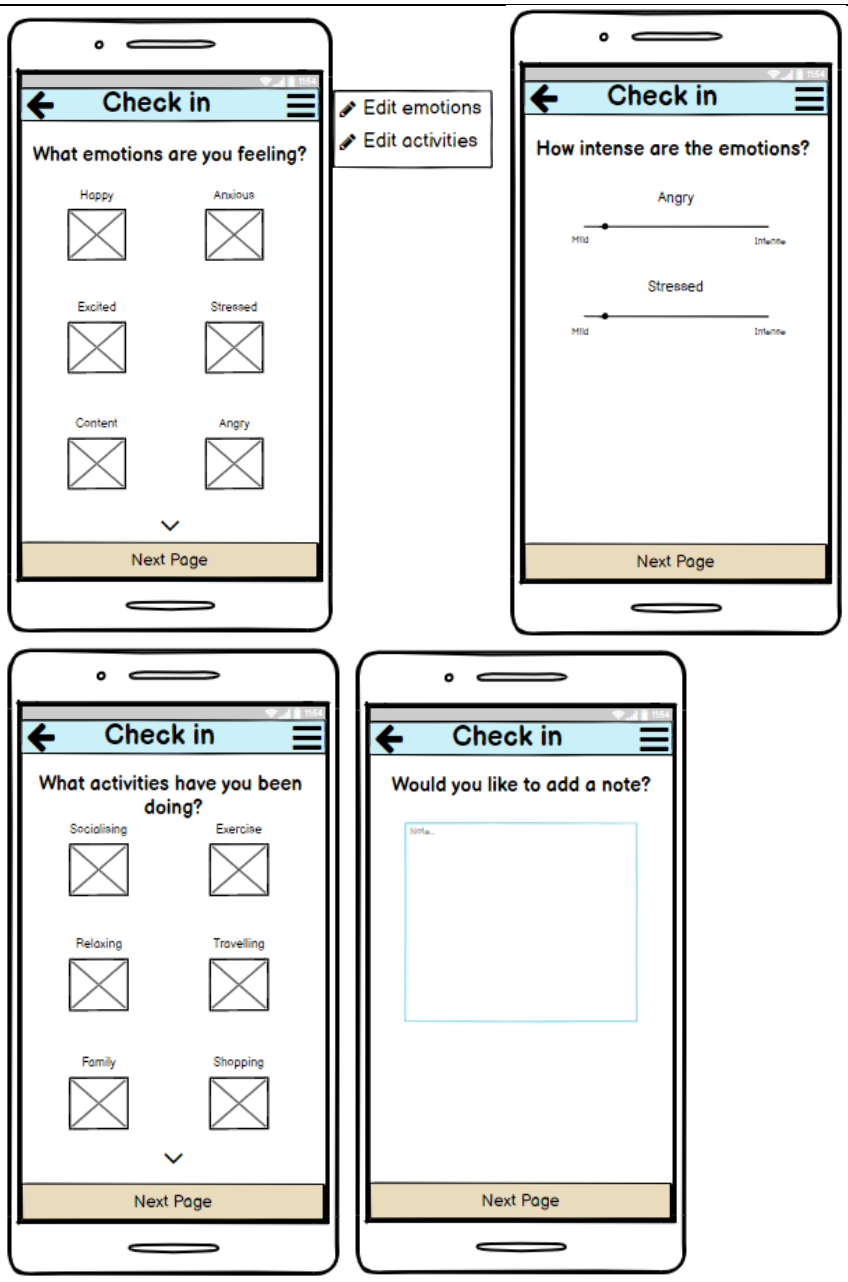
Table 23 – Diary entry screens



Screens 5,6,7 and 8

These four screens show the process of making a diary entry, the primary function for patients. The process is accessed through the "New Diary Entry" button from the home page (screen x). Clicking the question mark next to each input box will bring up a help dialog explaining the purpose of the section and describing some example inputs. Users must click the 'next page' button at the bottom of the screen to advance through the process, and the back arrow in the top left corner if they wish to go back a step. Screen 7 shows the pop-up displayed when clicking on the plus sign in the 'cognitive errors' section of screen 6. Only the title field will be required, and if the user wishes to exit at any time they can press the hamburger menu and select "save & exit", as shown in screen 5. The date and time will be automatically filled out to the current values.

Table 24 – Check-in screens



Screens 9, 10, 11, and 12

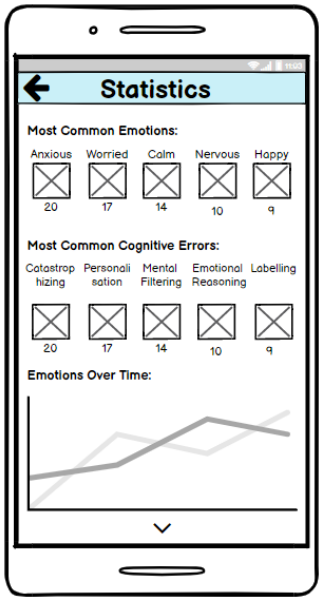
These 4 screens show the process of checking in, accessed through the “Check-in” button on the home screen (screen 3). As with the diary entry screens, the user can progress through the process by pressing the ‘next page’ button at the bottom, and go back with the back arrow in the top left. On screens 9 and 11 the arrow at the bottom indicates to the user that they are able to scroll down to view more emotions/activities; this is a common icon which will be used across the application. If the user wants to edit the list of emotions or activities, they can use the hamburger menu and then select which list they wish to edit, as shown in screen 9.

Table 25 – View diary entries

Screen 13

This screen is where the user can view a list of their past diary entries. It is accessed through the "View Diary Entries" button from the home page (screen 3). At the top is the search bar and 'filter & sort' options; pressing the 'filter & sort' drop down will bring up the menu shown to the right. If the user clicks on an entry, they will be taken to a page showing them all the details of that entry. Selecting edit will take them to the same page with editable inputs, and selecting delete will bring up the confirmation dialog shown on the right.

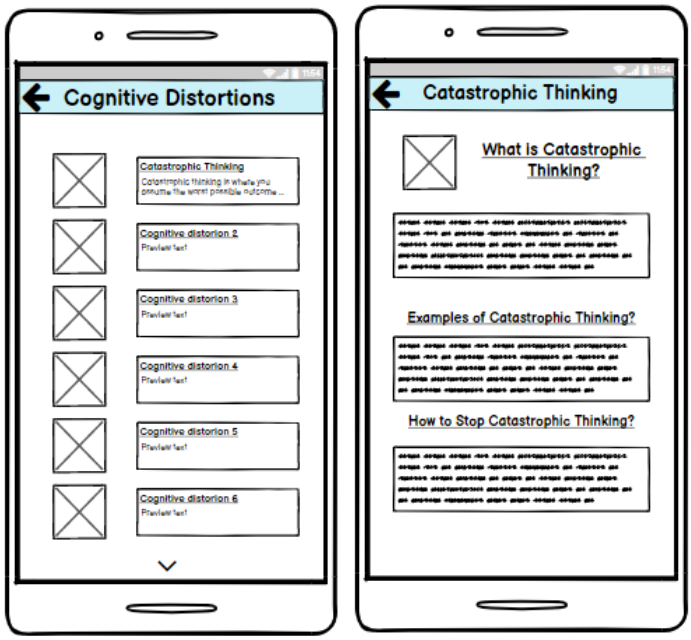
Table 26 – Statistics screen



Screen 16

On this screen the user can view statistics generated from their diaries and check-ins. It is accessed through the "View Statistics" option on the home page. As with other screens, the arrow at the bottom indicates to the user that they can scroll down to view more content.

Table 27 – Cognitive Distortion screens

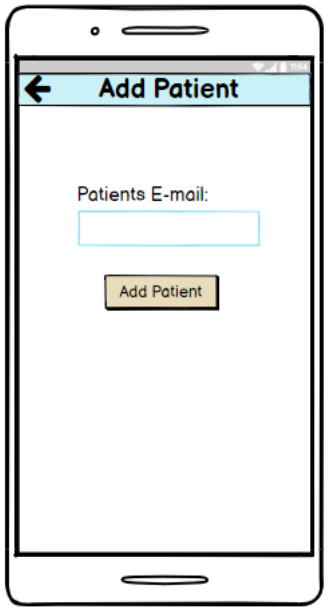


Screens 14 and 15

These are the screens where users can learn about cognitive distortions. Clicking on each of the distortions on screen 14 will take the user to screen 15, where they will be provided with further information about that cognitive distortion.

Therapist screens

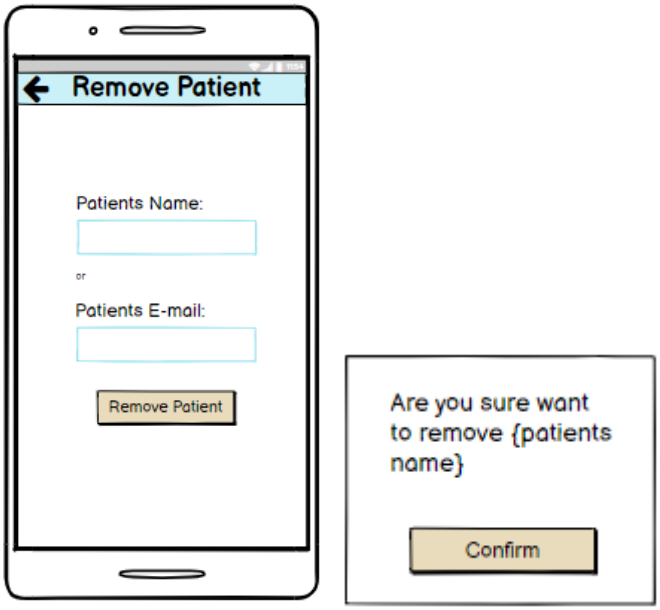
Table 28 – Add patient screen



Screen 17

This is the screen where a therapist can add a new patient to their list, by entering their e-mail. This screen is accessed through the “Add Patient” button on screen 4. If there is no patient registered with the e-mail entered, the user will be prompted to check the e-mail and try again.

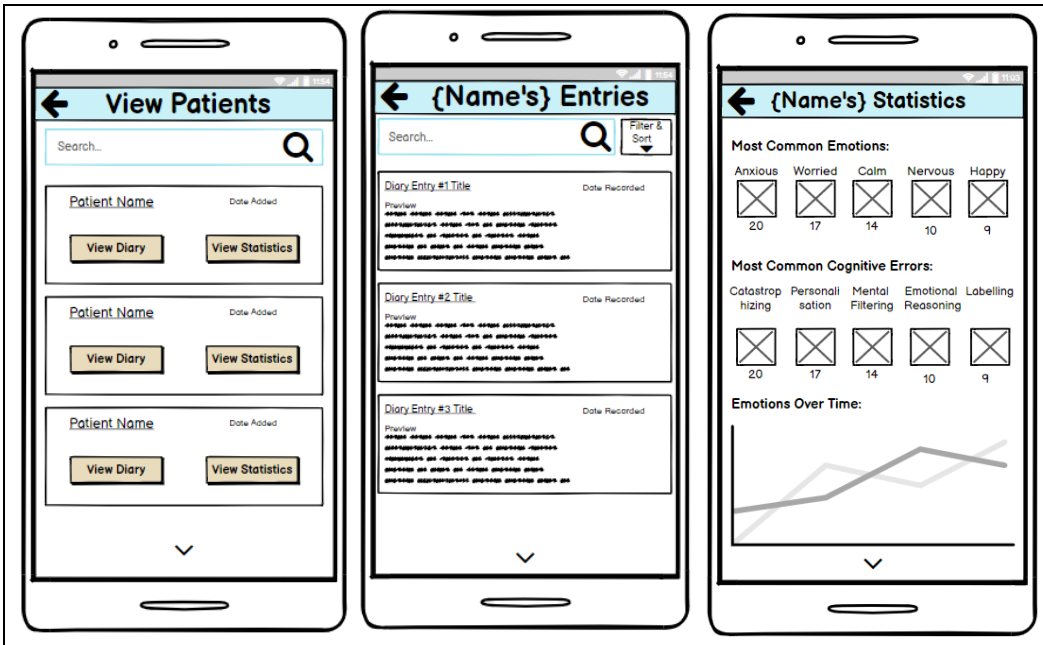
Table 29 - Remove patient screen



Screen 18

This screen is where therapists can remove patients from their list that they are no longer seeing. Entering the desired patients name or e-mail, and clicking the 'remove patient' button, brings up the dialog on the right where the user can confirm their action.

Table 30 – View patients screen



Screens 19, 20 and 21

These screens are where therapists can view their list of patients (screen 19), and each patient's diary entries and statistics (screens 20 and 21). They are accessed through the "View Patients" button on screen 4. On the "View Patients" screen the user can search through their list of patients using the search bar at the top, and choose to view a patient's diary or statistics using the buttons next to each patient. The "View Diary" button takes the user to screen 20, which is the same as the patients' diary entries view (screen 13), minus the edit and delete buttons. Clicking on

the “View Statistics” button takes the user to screen 21, which is again the same as the patients’ statistics view (screen 16).

6.3 – Heuristic Evaluation

While the initial wireframes described in the previous section cover all of the functional requirements of the system, it is important that they also fulfil the third non-functional requirement stated in section 4.2, “The system must be easy to use, navigate, and understand”. Nielsen’s 10 heuristic principles will therefore be used to evaluate the wireframes. These principles are (Nielson and Molich, 1990):

1- Visibility of system status

2- Match between system and the real world

3- User control and freedom

4- Consistency and standards

5- Error prevention

6- Recognition rather than recall

7- Flexibility and efficiency of use

8- Aesthetic and minimalist design

9- Help users recognize, diagnose, and recover from errors

10- Help and documentation

These principles will be rated on Nielson’s 5-part severity scale:

0- I don’t agree that this is a usability problem at all

1- Cosmetic problem only: need not be fixed unless extra time is available on project

2- Minor usability problem: fixing this should be given low priority

3- Major usability problem: important to fix, so should be given high priority

4- Usability catastrophe: imperative to fix this before product can be release

Table 31 – Heuristic evaluation 1

Heuristic Evaluation No	1
Screen No	1 and 2

Principle(s) violated	Help user recognise, diagnose, and recover form errors.
Problem	There are no error messages related to the form input; for example, if the user gets their password wrong, or tries to sign up with an email that is already registered. This could be very confusing to users as they wouldn't be aware of what was causing the issue.
Severity Rating	3
Solution	Include a text area for errors to be displayed which describes to the user what they have done wrong, e.g. "Password incorrect", or "E-mail already in use".

Table 32 – Heuristic evaluation 2

Heuristic Evaluation No	2
Screen No	3
Principle(s) violated	10
Problem	The purpose of each section of the app that is accessed from this screen is not explained.
Severity Rating	2
Solution	Next to each button, add a help icon which brings up a dialog box that explains what each section of the app is and what its purpose is.

Table 33 – Heuristic evaluation 3

Heuristic Evaluation No	3
Screen No	5,6,7,8
Principle(s) violated	Visibility of system status, User control and freedom, and consistency and standards.
Problem	As the user is going through the diary entry process there is no indication of their progress. The button at the bottom simply says "Next Page" on the first three screens and "Submit Entry" on the final screen. Further, if the user wanted to go back through the form, they would have to use the back arrow in the top right corner. This is usually used to go back to the main menu rather than back through a functions screens, so this could be confusing to the user. Finally, if the user wished to exit without saving, they would have to go back through each of the screens and then back once more to the main menu.
Severity Rating	3
Solution	Change the next page buttons to tabs; this way the user can see their progress and easily skip through the form if they wish to go back. Also, make the back arrow in the top bar exit the function from any of the screens - this will show a dialog message saying "Would you like to exit without saving".

Table 34 – Heuristic evaluation 4

Heuristic Evaluation No	4

Screen No	9, 10, 11 and 12
Principle(s) violated	Visibility of system status, User control and freedom, and consistency and standards.
Problem	As with evaluation 3, this series of screens (through the check-in process) does not display the progress to the user and they have to use the back arrow to go back through the screens of the individual functions. These screens also don't have a way to exit the function from any of the screens.
Severity Rating	3
Solution	As with evaluation 3, change the next page buttons to a series of tabs and make the back arrow exit the function. Also add a save and exit option into the hamburger menu.

7 – Implementation

7.1 Class Diagram

Figure 5 below shows a UML class diagram for the back-end of the system, in order to give an overview of the architecture.

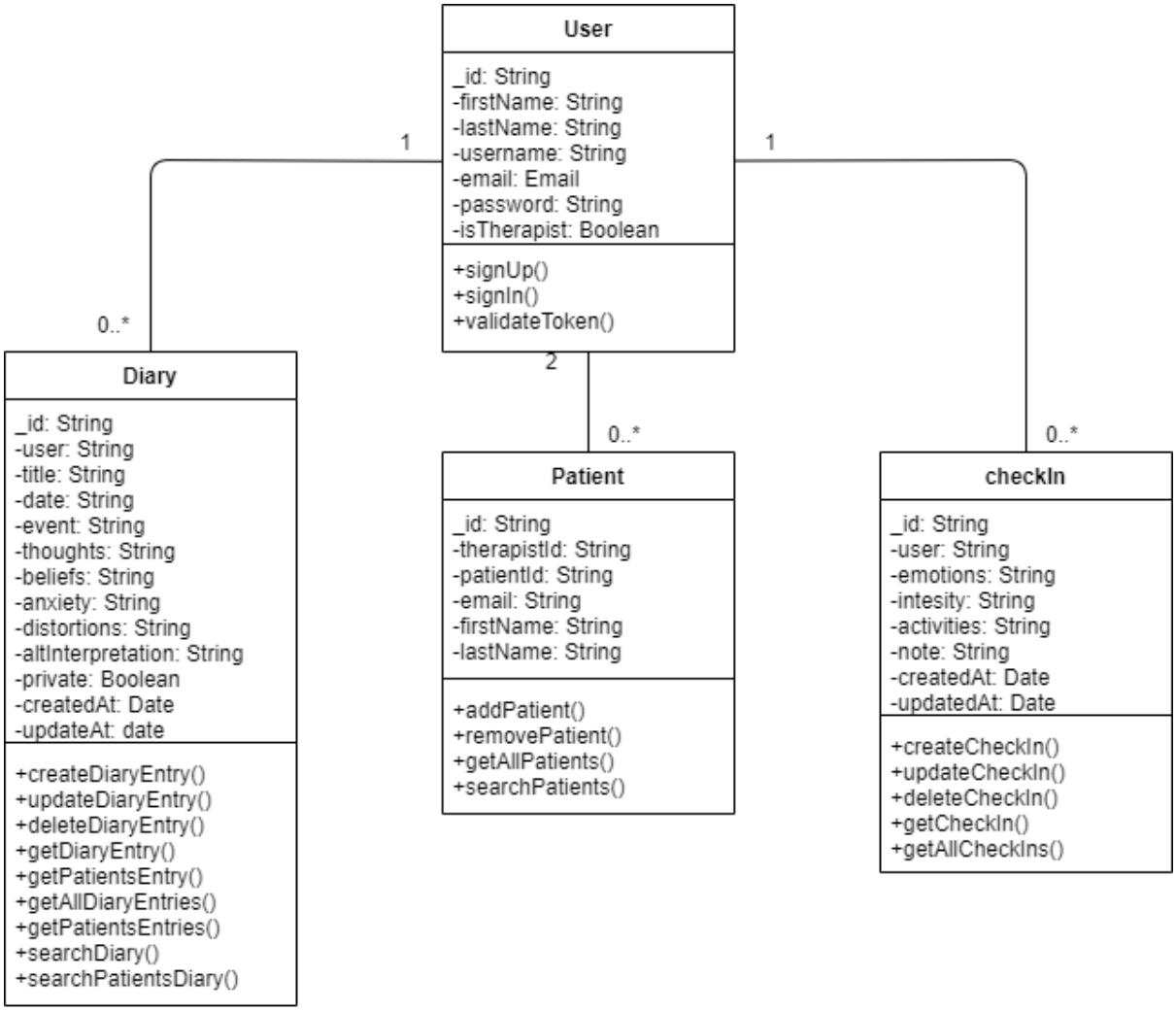


Figure 5 - UML class diagram of the application

The original design idea for the structure of the back-end had been to simply have one "User" collection which contained one document per user, making use of arrays to store the users' entries (and patients if they were a therapist). While this approach would have simplified the structure of the database and meant that all of a user's information was stored together, it also came with drawbacks. To the best of my knowledge, it is not possible to retrieve a single element from an array in a MongoDB document, only to retrieve whole documents that contain elements matching a search criteria (MongoDB [no date][a]). This would have led to inefficiencies when performing operations such as retrieving a single diary entry, having to retrieve all of the user's information and then filter out the desired entry. Having arrays within

a document also increases the complexity of searching for and retrieving a single item from the array, and as the arrays get bigger they can impact the performance of the application (MongoDB [no date][a]).

For these reasons a structure more akin to that of a SQL database, where foreign keys are used to link separate collections, has been used. The "_id" field of each collection was used as the key to create these links; for example, the "user" field of a diary entry would contain the "_id" field of the user that created the entry.

7.2 Back-end

At the start of the implementation phase I decided to focus my efforts on completing the back-end of the system before starting on the front-end. There were two main reasons for this decision; firstly, I was more confident in using the back-end frameworks (Node.js, Express.js and MongoDB) than the front-end ones (React Native), so I could start work on the back-end while continuing to learn React Native. Secondly, it was more efficient to work in this way. As I had defined the methods that the back-end needed in my class diagram, it was an obvious decision to produce all of these at once rather than working on the front- and back-ends at the same time.

To test the back-end's functionality the program Postman was used. Postman is an API (Application Programming Interface) platform that allows users to easily send http queries and view the server's responses. This is a useful feature when building the back-end of an application as it allows for quick testing of routes without having to implement any front-end features.

7.2.1 – File Structure

The structure of the back-end is simple and adheres to the principle of separation of concerns, which is important for large projects like this as it provides a clear code structure that is easy to expand upon in a modular fashion. The structure was inspired by JavaScript Mastery (2020a).

In the root folder there is an index.js file where all the other files are imported to, the connection to the database is made, and the server is run. The root folder also contains the four other folders, which contain the code for the back-end: controllers, middleware, models, and routes.

Models

Within the models folder there is a JavaScript file for each model used in the database: checkIn.js, diary.js, patients.js and user.js. In each file a mongoose schema is created following the attributes shown in the UML class diagram. From this a mongoose model is created which is then exported. These models define the fields of each collection in the database and also the attributes of these fields, such as the data type, if the field is required, and if the field is unique. For example, in figure 6, the schema for check-ins defines the five fields as “user”, “emotions”, “intensity”, “activities” and note. These fields are a mixture of strings and arrays, and only the “user” field is required.

```
import mongoose from "mongoose";

const Schema = mongoose.Schema;

const CheckInSchema = new Schema({
  user: {
    type: String,
    require: true,
  },
  emotions: {
    type: Array,
  },
  intensity: {
    type: Array,
  },
  activities: {
    type: Array,
  },
  note: {
    type: String,
  },
});

CheckInSchema.index({
  note:"text"
})

const CheckIn = mongoose.model("CheckIn", CheckInSchema);

export default CheckIn;
```

Figure 6 - Model for check-ins

Controllers

For each model is a corresponding file in the controller’s folder. These contain the logic for the ways in which the user will need to interact with each of the models. For example, the user’s controller contains methods for creating a new user and signing into an account, and the diary controller contains methods for creating, updating, and deleting a diary entry.

Routes

For each controller, a corresponding routes file defines the routes that are used to access each function and the http method for each route. For example, figure 7 below shows how

the methods defined in the therapist controller's file are bound to their corresponding routes and http verbs.

```
import express from "express";
import verifyTherapist from "../middleware/verifyTherapist.js";

import therapist from "../controllers/therapist.js";

const router = express.Router();

router.use("/", [verifyTherapist]);

router.post("/addPatient", therapist.addPatient);
router.delete("/removePatient/:email", therapist.removePatient);
router.get("/getAllPatients", therapist.getAllPatients);
router.post("/searchPatients", therapist.searchPatients);
export default router;
```

Figure 7 - Routes for the methods defined in the therapist controllers file

Middleware

The middleware folder contains functions that are bound to routes using the express.js middleware feature. When a middleware function is bound to a route, it is executed each time the route is accessed, before the controller function for that route is executed. I have used this feature for token verification; all routes other than signing in and signing up require a valid JSON web token in the header of the request.

7.2.2 Example Back-End Features

Following from the architecture of the back-end, a few examples of key functions will be described. These are password encryption, signing in, and creating a diary entry.

Password encryption

```
// encrypt user password before saving to database
const saltRounds = 10;
UserSchema.pre("save", async function (next) {
    if (this.isModified("password")){
        this.password = await bcrypt.hash(this.password, saltRounds);
    }
    next()
})
```

Figure 8 - User password encryption

In the user model a mongoose "pre" middleware is used to encrypt the users' passwords. Specifying "save" as the first argument in the pre-ware function means that the callback function is executed before the data is saved into the database. Within the callback function, the if statement checks if the password is different to the previous saved password, and only hashes the password if this is the case. This means that if another part of the user's information was being edited, the password would not be re-hashed. To hash the password,

the npm library bcrypt was used. The library implements the bcrypt password hashing function, which is based on the Blowfish cipher. Bcrypt is very secure as its complexity can be scaled to match the increase in computing power, meaning it is resistant to brute force cracking attacks (ref). Bcrypt provides a hash function with two parameters - a string and the amount of salt rounds - salt being a small piece of data that is randomly generated and added to the given string before hashing.

```
password: "$2b$10$Dx7EVWMIUBL44mip7pw8jeBgfyXnpCwvHiHV6rf8gkwWiIavX8jry"
```

Figure 9 - example of how a password is stored in the database

If passwords were not encrypted in the database, a data leak would mean that anyone would be able to access users' personal information. For example, the recent leak of the data of 8.3 million plain text passwords from the website DailyQuiz (Cimpanu 2021). Using salt when hashing provides an additional security measure in the case of this happening. Without the salt, hash lookup tables could be used to identify common passwords, i.e. "password", "qwerty123". This works as hash functions always produce the same output from the same input; adding salt changes the output and eliminates this possibility.

Sign In

Figure 11 shows the signIn() function that is called when a user tries to sign in. The parameters req and res are objects passed to the function by Express.js, and are the request and response objects. The request object contains all the information about the request, such as query strings, body, and HTTP headers, and the response object is the response that will be sent by express. The method response.status() sets the HTTP response status code, and the method response.send() sends the response, with the body of the response being defined as an object in the brackets.

When signing in, the first action is querying the "User" table for a match with the entered username (accessed through req.body.username). If there is an error or no user is found, then a response is sent back with the HTTP code 500 or 404 respectively. Next, bcrypt is used to check if the user has entered the right password. Bcrypt provides a compare function which accepts a string and a hash and returns a Boolean value of true if the string and hash match. If the password is invalid a response is sent to inform the user. If the password is valid a JWT (JSON web token) that contains the user's id, a Boolean value "isTherapist", and a Boolean value "rememberMe" is computed and the token is sent back to the front-end with a status

code of 200 (successful response). An example of an access token can be seen below in figure 10.

Figure 10 - Example in Postman of the response of a successful request to /signin

```
export const signIn = (req, res) => {
  User.findOne({
    username: req.body.username,
  }).exec((err, user) => {
    if (err) {
      res.status(500).send({ message: err });
      return;
    }

    if (!user) {
      return res.status(404).send({ message: "User not found." });
    }

    bcrypt
      .compare(req.body.password, user.password)
      .then((passwordValid) => {
        if (!passwordValid) {
          return res
            .status(401)
            .send({ acessToken: null, message: "Invalid password." });
        }

        const token = jwt.sign(
          {
            id: user._id,
            isTherapist: user.isTherapist,
            rememberMe: req.body.rememberMe == "checked" ? true : false,
          },
          process.env.jwtkey,
          { expiresIn: 1209600 }
        );

        res.status(200).send({
          accessToken: token,
        });
      });
  });
};
```

Figure 11 – signIn() function

JWTs have two main uses, authorization and information exchange (Agarwal 2018), , both of which are utilized in this example. After the token is sent to the user it is stored on their device and attached in the header of any subsequent requests. The verifyToken() middleware (figure 12 below) is applied to all routes other than logging in and signing up; this ensures that only logged in users can access the routes. Additionally, some views on the front-end need to render different content based on the type of user, which can be checked using the "isTherapist" Boolean contained in the token. The token also contains the information of whether the user wants to have their details remembered, so they don't have to log in on the

next use, in the “rememberMe” Boolean. As JWTs are signed based on the contents of the token, you can be sure that the content of the token is correct as it would be invalid when verified if it had changed (Auth0 [no date]).

```
import jwt from "jsonwebtoken";

const verifyToken = (req, res, next) => {
  const token = req.headers["x-access-token"];

  if (!token) {
    return res.status(401).send({ message: "Token not provided" });
  }

  jwt.verify(token, "top-secret", (err, decoded) => {
    if (err) {
      return res.status(401).send({ message: "User not authorized" });
    }
    req.userId = decoded.id;
    next();
  });
};

export default verifyToken;
```

Figure 12 - Middleware to verify JSON web tokens

Creating and retrieving a diary entry

Figure 13 shows the code used to submit a diary entry into the database. After checking the “title” field is not blank, as this field is required in the database, a “Diary” object (from the mongoose model) is instantiated containing the “userId” and the request body spread using ES6 spread notation. Finally, the “Diary” object is saved using mongoose’s save() function which submits the object into the database. The ease of saving the data to the database demonstrates how useful mongoose’s object modelling feature is.

```
const createDiaryEntry = (req, res) => {
  const userId = req.userId;

  if (!req.body.title) {
    res.status(400).send({ message: "Title must not be blank" });
    return;
  }
  const diary = new Diary({ user: userId, ...req.body });
  diary.save((err, entry) => {
    if (err) return res.status(500).send({ message: err.message });
    res.send({ message: "Diary logged" });
  });
};
```

Figure 13 – createDiaryEntry() function

Figure 14 shows the getEntry() function used to retrieve a specific diary from the database. The mongoose find() method is used on the diary object to query the database for documents (diary entries) with credentials matching the “userId” and “diaryId” passed from the front-

end. If there is no data found or an error occurs the appropriate response is sent, and if the data is successfully retrieved it is sent to the front-end. (Note: this function is in a slightly different format to “createDiaryEntry” in figure 13 as it called from two routes, “/getDiaryEntry” and “/getPatientsEntry”, which configure the “userId” and “diaryId” variables before calling the function)

```
async function getEntry(userId, diaryId, res) {
  Diary.find({ user: userId, _id: diaryId })
    .then((data) => {
      if (data.length == 0) {
        return res
          .status(404)
          .send({
            message: `Error: No entries found with id ${diaryId}`,
          });
      } else {
        return res.send(data);
      }
    })
    .catch((err) => {
      res.status(500).send({
        message: `Error while trying to retrieve diary entry with id ${diaryId}`,
      });
    });
}
```

Figure 14 - getDiaryEntry() function

After developing any of the methods in the back-end they must be bound to a route, shown in figure 15. The first step in the process is creating an express router object. A router object has methods for each of the HTTP verbs (e.g. GET, POST, PUT) which are called with two arguments, the URL and the callback function. For example, in the third line of figure 15 an HTTP post method is being defined on the URL “/createDiaryEntry”, when a request is sent to this URL the function “createDiaryEntry” is called.

```
const router = express.Router();

router.post("/createDiaryEntry", diaries.createDiaryEntry);
router.get("/getDiaryEntry/:diaryId", diaries.getDiaryEntry);
```

Figure 15 - Example of binding methods to routes

To expose these routes to the front-end, the router is imported into index.js (as diaryRoutes) and applied using the use() method of the Express app object.

```
const app = express();
const port = process.env.PORT || 8080;

app.use(cors());
app.use(express.json());

app.use("/", userRoutes);
app.use("/", diaryRoutes);
app.use("/", checkInRoutes);
app.use("/", therapistRoutes);
```

Figure 16 - Applying each of the sets of routes to the server

Conclusion

These two examples have been used as they display the typical structure of the back-end and the patterns used for interacting with the database.

To summarize this structure, using the diary as an example:

1. A model is produced in the models folder which defines the attributes that can be stored in the diary database table
2. In the controller's folder diaries.js defines all of the methods for interacting with the database
3. In the routes folder diaries.js a router is created which binds these methods to URLs
4. In index.js the router is applied to the Express app, which exposes the routes to requests made from the front-end

The typical pattern of database interaction is as follows:

1. When putting information into the database there may be a check to make sure the correct information is present, then the information is saved and either a successful or error code is sent back to the front-end.
2. When retrieving information from the database there may be some check as to whether the user is a patient or therapist and an assignment of variables accordingly, then a mongoose function such as find(), findOne(), updateOne() or deleteOne() is used to match the correct entry in the database. If the operation is successful, or there is an error, the corresponding code and information are sent back to the frontend.

7.4 Front-end

7.4.1 Structure:

Figure 17 below shows an overview of the file structure of the system. This structure takes ideas from Habilelabs (2021) and aims to keep the code in an organised and easy to understand manner, making the development and any future expansion of the app organized and efficient.

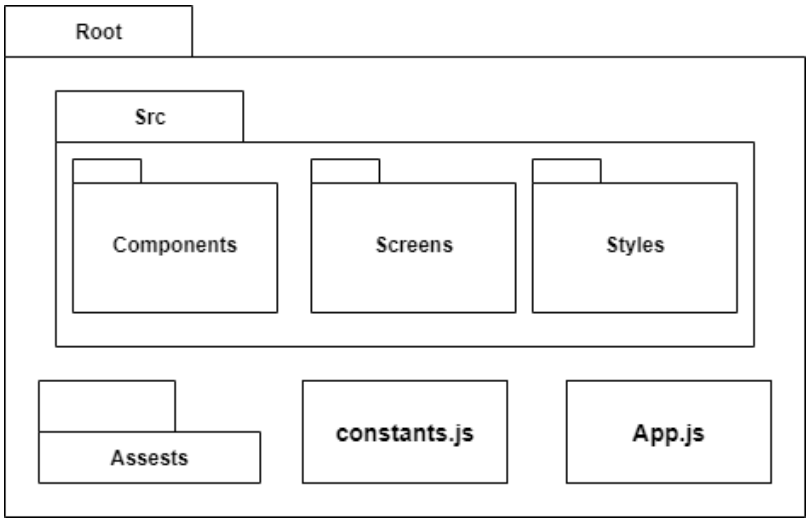


Figure 17 - File structure of the front-end of the application

The assets folder and constants.js are not fully utilized in this project, with the assets folder only containing default pictures used by Expo and constants only containing one constant. Therefore, they are included mainly for completeness and future expandability of the application.

7.4.2 Expo CLI:

The first choice made in developing the front-end was between expo CLI or React Native CLI, the two ways to build a React Native app.

Expo is a group of tools built on top of React Native that assist in the quick and easy development of an app. Expo has 3 three main advantages over RN CLI (Feroze 2021):

1) You can use any code editor, rather than having to use Xcode and Android Studio. When using RN CLI you have to have a computer running macOS to develop for iOS.

2) Setting up a project is very quick using the “expo init” command. It only takes a couple of minutes to have a project structure setup.

3) Running the app is easy via a QR-code using Expo Go, an app available on the play and ios stores. This will be very useful when carrying out the testing for my app as I can simply share the QR code with my participants and they will be able to run the app on their phones.

There are some drawbacks to using expo, mainly not being able to use native modules (modules written specifically for android or iOS) (Expo [no date]).However, these are concerned with features not used in my application, so for me using Expo CLI was the right choice.

7.4.3 Navigation

As React Native does not have an in-built method for handling navigation, the documentation suggests the use of the community package React Navigation. React Navigation is easy to setup, has good documentation, and provides built-in navigators such as tabs, drawers, and stacks that work on android and iOS, which made it a good choice for use in my app. I have utilised two of the navigation types: a stack, and tabs.

Stack Navigator

A stack navigator operates using a stack data structure where screens can be pushed to or popped from the top of the stack. I have used a stack for the main navigation through the app as it is a very common method that will feel familiar to users. In addition to this, the stack navigator in React Navigation is configured to function in the default way on IOS and Android, providing a consistent user experience.

Within the application there are three sets of screens that can be displayed to the user; auth screens (sign in and sign up), screens for patient users, and screens for therapist users. As can be seen in figure 18, these groups are conditionally loaded into the stack based on the two state variables, “isSignedIn” and “isTherapist”.

Contained in each of the groups of screens are multiple screen components, each configured through custom props. Figure 19 shows the screen component for the sign in screen where three props are provided: “name”, “options”, and “children”. The “options” prop is used here to provide the information for what the display in the header, and the “children” prop provides the screen component that is displayed when this screen is viewed.

```
const Root = createStackNavigator();
const Navigation = ({
  isSignedIn,
  setIsSignedIn,
  isTherapist,
  setAuthToken,
  setIsTherapist,
}) => {
  return (
    <NavigationContainer>
      <Root.Navigator>
        {isSignedIn ? (
          isTherapist ? (
            <>
            {/* Therapist Screens */}
              <Root.Group>···
              </Root.Group>
            </>
          ) : (
            <>
            {/* Patient Screens */}
              <Root.Group>···
              </Root.Group>
            </>
          )
        ) : (
          <>
          {/* Auth screens */}
            <Root.Group>···
            </Root.Group>
          </>
        )}
      </Root.Navigator>
    </NavigationContainer>
  );
};
export default Navigation;
```

Figure 18 - Navigation component

```
<Root.Screen
  name="SignIn"
  options={{
    header: (props) => (
      <CustomNavigationBar
        title={"Sign In "}
        {...props}
      />
    ),
  }}
  children={(props) => (
    <SignInScreen
      setIsSignedIn={setIsSignedIn}
      setAuthToken={setAuthToken}
      setIsTherapist={setIsTherapist}
      {...props}
    />
  )}
/>
```

Figure 19 - Navigation screen component for the sign in screen

To pass props to the screen component, children must be set to a fat arrow function, which returns the component with the props set in the usual way. As React Navigation has the default “navigation” and “route” props, they must be passed through the function and spread, using “{…props}”, for them to be available in the screen component. Using this

method, the custom and default props can be directly accessed within the component (figure 20) using object destructuring syntax.

```
> const SignInScreen = ({navigation, setIsSignedIn, setAuthToken, setIsTherapist}) => {···
  };
```

Figure 20 - SignInScreen component

To navigate between screens, the methods of the navigation object are used. The main methods I used were goBack(), popToTop(), and navigate(). goBack() removes the screen from the top of the stack, popToTop() pops all but the first screen from the stack, and navigate() pushes a screen to the top of a stack if it is not in the stack or pops all the screens above it if it is. Originally I had been using the push() method rather than navigate(), but with push() you can push the same screen to the top of the stack multiple times (e.g. if a user spams a button), which could have led to a confusing experience for users.

Together these methods provide a comprehensive set of tools that comprise all of the stack navigation within the app.

Tab Navigator

In the heuristic evaluation in section 6.3, I identified the need for tabs showing users their progression through the diary entry and check-in processes. Within the diary entry page, the React Navigation material top tab navigator was used, and 3 screens/tabs (event, examination, and interpretation) were defined. The definition of the tabs is shown below in figure 21, and the result in the app is shown in figure 22. This solution allowed for quick and easy navigation through the diary entry process for the users in an intuitive manner (either click the tabs or swipe the screen), whilst also providing visual feedback of their progress through the task.

```jsx
const Tab = createMaterialTopTabNavigator();
return (
  <>
    <Tab.Navigator>
      <Tab.Screen
        name="Event"
        children={() => (
          <EventScreen
            userInputs={userInputs}
            setUserInputs={setUserInputs}
            mode={route.params.mode}
          ></EventScreen>
        )}
      />
      <Tab.Screen
        name="Examination"
        children={() => (
          <ExaminationScreen
            userInputs={userInputs}
            setUserInputs={setUserInputs}
            mode={route.params.mode}
          ></ExaminationScreen>
        )}
      />
      <Tab.Screen
        name="Interpretation"
        children={() => (
          <InterpretationScreen
            userInputs={userInputs}
            setUserInputs={setUserInputs}
            mode={route.params.mode}
            submit={submit}
            update={update}
          ></InterpretationScreen>
        )}
      />
    </Tab.Navigator>
    <AlertBar></AlertBar>
  </>
```

Figure 21 - Definition of tabs in the diary entry screen

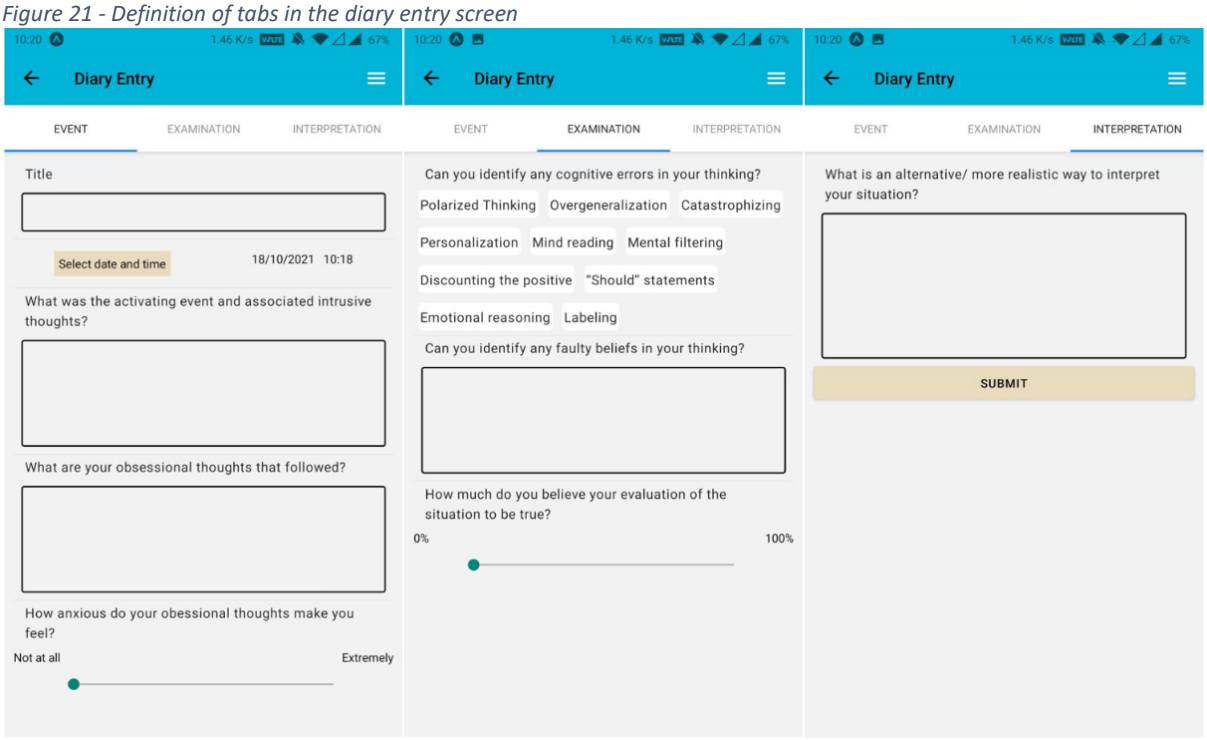


Figure 22 - Diary entry tabs in the app

Navigation overview

The two following figures demonstrate the ways in which patient users (figure 23) and therapist users (figure 24) can navigate the app.

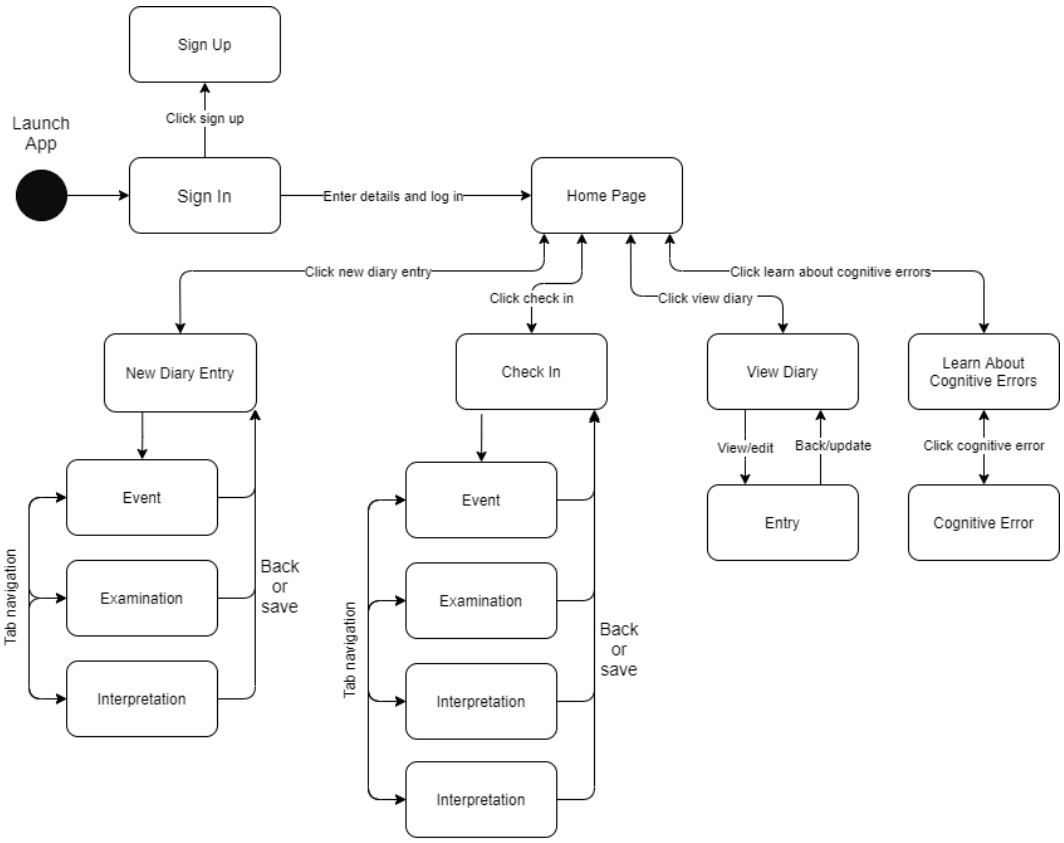


Figure 23 - Map of patient users' navigation around the app

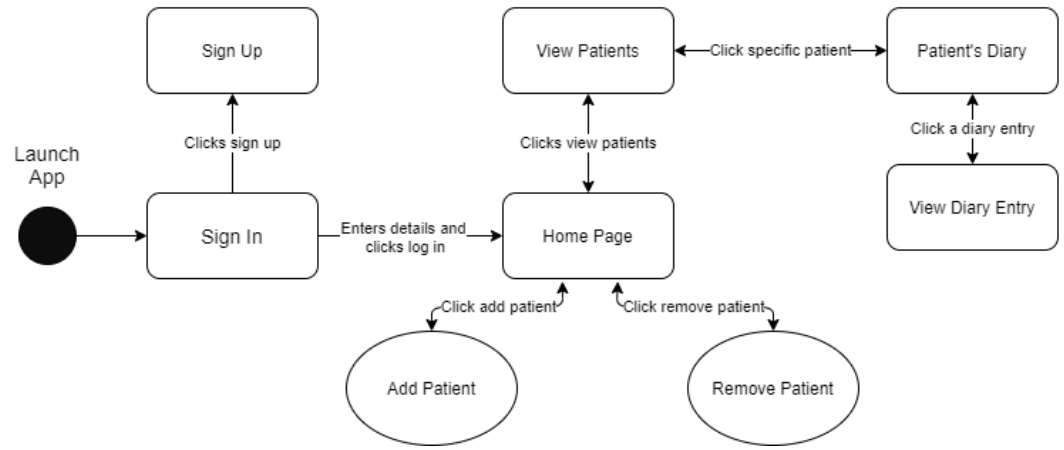


Figure 24 - Map of therapist users' navigation around the app

7.4.5 Communication with back-end

As the application is made of a front- and back-end, communication between the two parts is a vital part of the design. I decided to use the npm package axios, used to handle http requests in the app, as it is a very popular package with good documentation. Axios provides methods

for each of the HTTP verbs such as axios.get() or axios.post(), and allows for easy configuration of the body and header of requests.

Signing-in

The first communication with the back-end occurs when the user signs into the app, figure 25 shows the function that is called.

```
async function signIn() {
  try {
    const response = await axios.post(`${BASE_URL}signin`, {
      username: username,
      password: password,
      rememberMe: checked,
    });
    await SecureStore.setItemAsync("token", response.data.accessToken);
    setAuthToken(response.data.accessToken);
    setIsSignedIn(true);
    const decodedToken = jwt_decode(response.data.accessToken);
    setIsTherapist(decodedToken.isTherapist)
  } catch (err) {
    console.log(err);
    alertBar.handleAlert(true, "Error: Username or password incorrect");
  }
}
```

Figure 25 - signIn() function

The axios.post() method is used to send the details entered by the user to the "/signIn" route, storing the response in the "response" constant. The JWT that is attached in the body of the response (back-end figure) is stored on local storage using the npm package expo-secure-store. This package was chosen as it has an easy-to-use interface and stores date securely using encryption, unlike the default React Native module AsyncStorage, which is not encrypted. Storing the token locally allows the "remember me" function to work, as each time the app loads it checks the local storage for a valid token and automatically logs the user in if one is found. Next, the state variables "authToken", "isSignedIn" and "isTherapist" are set. These variables are used throughout the application to configure the display based on the user's state and type.

As with all axios requests in the application, this one is surrounded by a try-catch block. If the response contains an error response code then axios rejects its promise and throws an error, which is caught by the catch block. Within the catch block the appropriate action can be taken - in this instance, informing the user that their username or password is incorrect.

Request with headers

Once the user has signed in and received a JWT, it must be attached to the header of all subsequent requests to be verified on the back-end (as shown in figure 12). This adds to the

security of the system as only users signed into an account can access the methods of the application.

```
const authToken = useContext(tokenContext);

let axiosInstance = axios.create({ baseURL: BASE_URL });
axiosInstance.defaults.headers.common["x-access-token"] = authToken;
```

Figure 26 - Axios instance definition

To attach the JWT to all requests it is shared throughout the app using context, then in each screen where requests are sent an axios instance is created, with the token set in the default headers (figure 26). This means that each time the instance is used, the token is automatically attached to the header of the request, as shown in figure 27 below. At first, I had tried to create a single axios instance in app.js with the headers attached and share this instance throughout the app using context. This didn't work due to complications with the asynchronous nature of fetching the token from storage and the way React handles updates to objects in state, so multiple instances were used instead.

```
async function submit() {
  try {
    await axiosInstance.post(`createDiaryEntry`, {
      ...userInputs,
      date: userInputs.date.toString(),
    });
    alertBar.handleAlert(true, "Success: Entry added to diary");
    navigation.popToTop();
  } catch (err) {
    console.log(err);
    alertBar.handleAlert(true, err.response.data.message);
  }
}
```

Figure 27 - submit() function

All of the other HTTP requests in the app work in the same manner as the submit() function shown in figure 27.

7.4.6 AlertBar

Within the application I wanted to have a consistent way to show messages to the user. To do this, I used the React Native Paper component Snackbar in a custom component shown below in figure 29

```
import React, { useContext } from "react";
import { Portal, Snackbar } from "react-native-paper";
import alertBarContext from "./Context/alertBarContext";

const AlertBar = () => {
  context = useContext(alertBarContext);
  return (
    <Portal>
      <Snackbar
        visible={context.visible}
        onDismiss={() => context.handleAlert("", false)}
        action={{
          label: "Dismiss",
          onPress: () => {
            context.handleAlert("", false);
          },
        }}
        duration={5000}
      >
        {context.text}
      </Snackbar>
    </Portal>
  );
};

export default AlertBar;
```

Figure 29 - Alert bar component

```
<alertBarContext.Provider
  value={{
    visible: alertBarVisible,
    text: alertBarText,
    handleAlert: (text, visible = true) => {
      setAlertBarVisible(visible);
      setAlertBarText(text);
    },
  }}
>
```

Figure 28 - Alert bar context

In the AlertBar component, the Snackbar's properties are set based on the context "alertBarContext". As the context provides a method to set the text and visibility (figure 28 above), it is very concise to display an error message from anywhere in the app. First, make sure the AlertBar is being rendered, then import the context and use the React "useContext" hook to get the value of the context (figure 29 above). Finally, call the handleAlert() method from the context and pass the message to be displayed(figure 30).

```
alertbar.handleAlert( "Success: Entry added to diary");
```

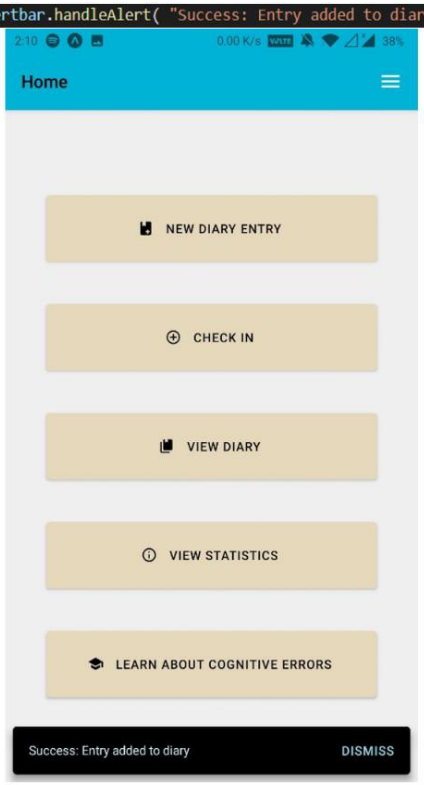


Figure 30 - Code for showing an alert to the user and the resulting alert

This method for displaying messages to the user is optimal for a few reasons: the message is always displayed in the same place meaning it is unlikely for users to miss a message, is not intrusive to the rest of the content on the page, and it is a common approach used in many apps so will be familiar to the users. Furthermore, storing the status of AlertBar in context means that the notifications persist between screens; for example, when submitting a diary entry the user is navigated back to the home page, but the alert still shows (figure 30).

7.4.7 Making a diary entry:

The main functionalities of the app are based around diary entries. To make a diary entry, inputs must be collected from the user in a variety of ways and then be submitted to the database. User inputs are collected through a variety of TextInput, Slider, and custom components. The manifestation of each of these methods in the app can be seen in figure 22.

Text Input

TextInput is a React Native component used to receive keyboard input from the user. The TextInput for the "title" field can be seen in figure 31 below, and exemplifies the typical pattern for text inputs. The "onFocus" and "onBlur" properties are used to update state variables to track which input the user has focused, providing feedback to the user on the state of the application by conditionally including the "input.focused" styles. The "onChangeText" property is used to define the behaviours when the user changes the text of a controlled component. Here, a callback function is passed to setUserInputs() in order to access the previous state. Doing this in this way rather than using the state variable directly ensures the up to date state value is used (React [no date][b]). In the return, "prevState" is spread and just the title is updated. setUserInputs() is used this way as state cannot be directly mutated. Instead, a new object is built and is shallow merged with the previous state (React [no date][b]).

```
<TextInput
  editable={!isTherapist}
  key="title"
  value={userInputs["title"]}
  mode="outlined"
  onChangeText={(text) =>
    setUserInputs((prevState) => {
      return { ...prevState, title: text };
    })
  }
  onFocus={() => {
    setTitleFocused(true);
  }}
  onBlur={() => {
    setTitleFocused(false);
  }}
  style={[
    input.input,
    input.small,
    titleFocused && input.focused,
  ]}
></TextInput>
```

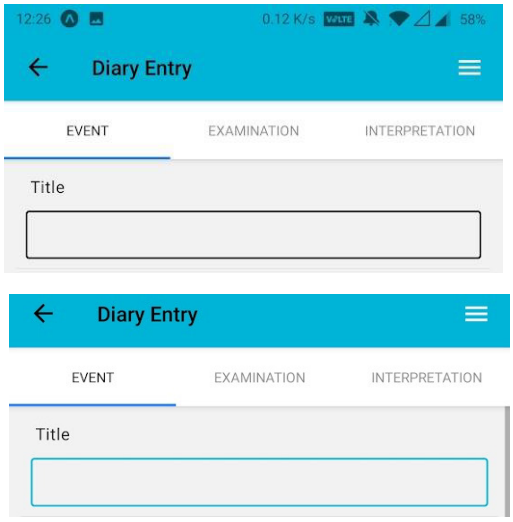


Figure 31 - TextInput component for the title field and the title field shown un-focused and focused

Slider

Sliders are implemented using the npm package, and are a simple component that lets the user select a single value from a range. Figures 32 and 33 below show the implementation of the slider and the result within the app.

```
<Slider
  disabled={isTherapist ? true : false}
  style={{ width: 300, height: 40, alignSelf: "center" }}
  minimumValue={0}
  maximumValue={1}
  minimumTrackTintColor="#FFFFFF"
  maximumTrackTintColor="#000000"
  onValueChange={(value) =>
    setUserInputs((prevState) => {
      return { ...prevState, anxiety: value };
    })
  }
/>
```

Figure 32 - Diary entry slider for users to rate their anxiety

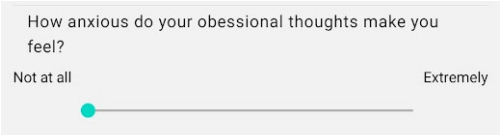


Figure 33 - Slider produced by the code in figure 33 (note: labels added separately)

Selection of cognitive Errors

The cognitive error boxes seen in figure 34 are rendered in a simple View component with "flexWrap" set to "wrap" (figure 35 below). This is an easy way to make sure the boxes fit neatly, independent of the size of the screen the app is being viewed on.

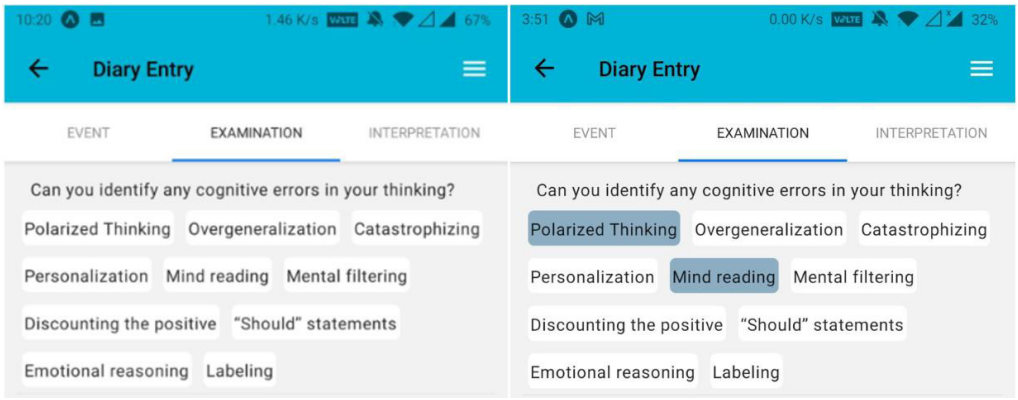


Figure 34 - Cognitive error boxes, boxes highlighted on the right have been selected

```
<View style={{ flexDirection: "row", flexWrap: "wrap" }}>
  {renderCognitiveErrors()}
</View>
```

Figure 35 - View container for the cognitive errors boxes

The cognitive errors boxes are made up of a custom component, SelectionItem, wrapped in a React Native component, TouchableOpacity. SelectionItem is a reusable component for when users need to select items from a list. It displays the title of an item passed to it, in this case the name of the cognitive error, and conditionally changes the styles applied based on whether it has been selected or not (example in figure 34). The TouchableOpacity component makes each of the items clickable through its "onPress" property. A map function is used on

the “Data” variable, to loop over each item and return a component for each. This approach has been used as it is very scalable and easy to edit; for example, in the future if the user could edit the list of cognitive errors none of the code for rendering them would have to change. Having simple components like SelectionItem that are re-used throughout the app also leads to less repetition of code, and makes the project easier to maintain.

```
const renderCognitiveErrors = () =>
  Data.map((item) => (
    <TouchableOpacity
      key={item.title + "touch"}
      onPress={() =>
        isTherapist
          ? undefined
          : handleDistortionClick(item, setUserInputs)
      }
    >
      <SelectionItem
        key={item.title + "item"}
        item={item}
        StateValue={userInputs["distortions"]}
        narrow={true}
      ></SelectionItem>
    </TouchableOpacity>
  ));
```

```
const SelectionItem = ({ item, StateValue }) => {
  return (
    <Surface
      style={[
        selection.item,
        StateValue.includes(item.title)
          ? selection.selected
          : selection.unSelected,
      ]}
    >
      <Subheading>{item.title}</Subheading>
    </Surface>
  );
};
```

Figure 36 - renderCognitiveErrors() function and custom component SelectionItem

When any of the cognitive error boxes are clicked the function handleDistortionClick() is called (figure 37 below), which is the function that handles the update to the state. When the function runs, the first action is to check the previous state to see if the clicked distortion is currently selected or not. If it is currently selected it is removed from the state using the JavaScript array filter method, if it is not selected then it is added to the state.

```
const handleDistortionClick = (item, setUserInputs) => {
  setUserInputs((prevState) => {
    if (prevState.distortions.includes(item.title)) {
      const newDistortions = prevState.distortions.filter(
        (distortion) => distortion !== item.title
      );
      return { ...prevState, distortions: newDistortions };
    } else {
      const newDistortions = prevState.distortions.concat(item.title);
      return { ...prevState, distortions: newDistortions };
    }
  });
};
```

Figure 37 - handleDistortionClick() function

Submitting

Finally, once the user has entered all the information they wish, they can press the “Submit” button on the final tab or save and exit in the hamburger menu. This runs the submit() function which submits the information into the database. (Submit() function can be seen in figure 27)

Checking-in:

Alongside a diary entry, checking in is the other means for users to input data into the app. As can be seen in figure 38 below, the same methods of collecting user input (text inputs, sliders, and a list using SelectionItem) are utilized in checking-in and in a diary entry. These methods are implemented in the same way, as has been explained in the previous sections, so will not explained again here.

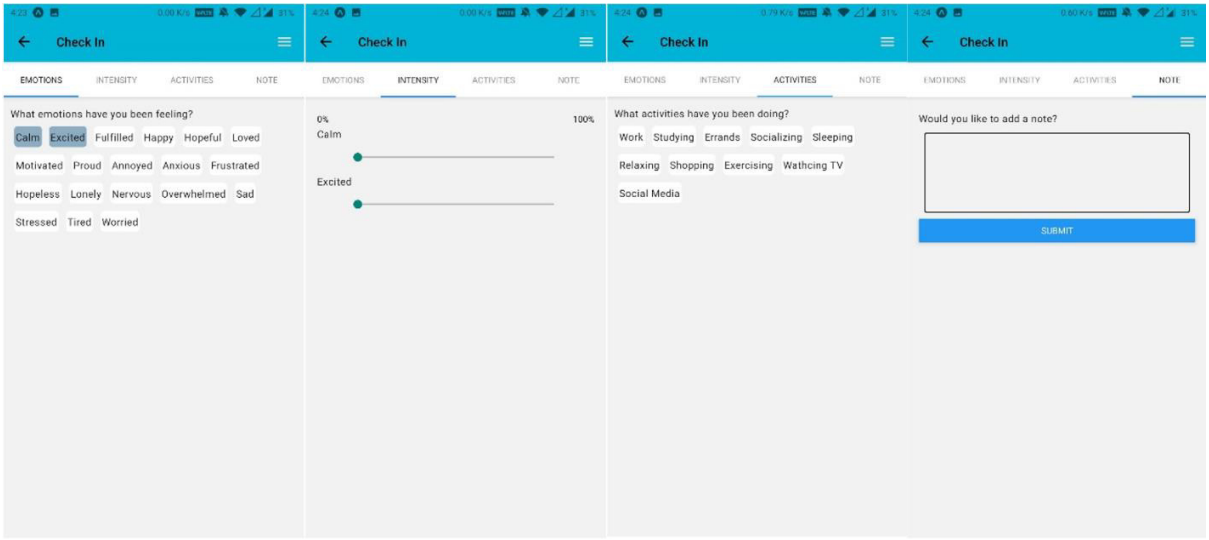


Figure 38 - Check-in screens

7.4.8 Diary

Viewing the diary

When a user has made a diary entry or check-in, it will appear in their diary together with their other entries. The diary screen can be seen below in figure 39. The first time the diary screen loads, the diary entries and check-ins are retrieved from the database, merged, sorted, and stored into the state variable “data”.

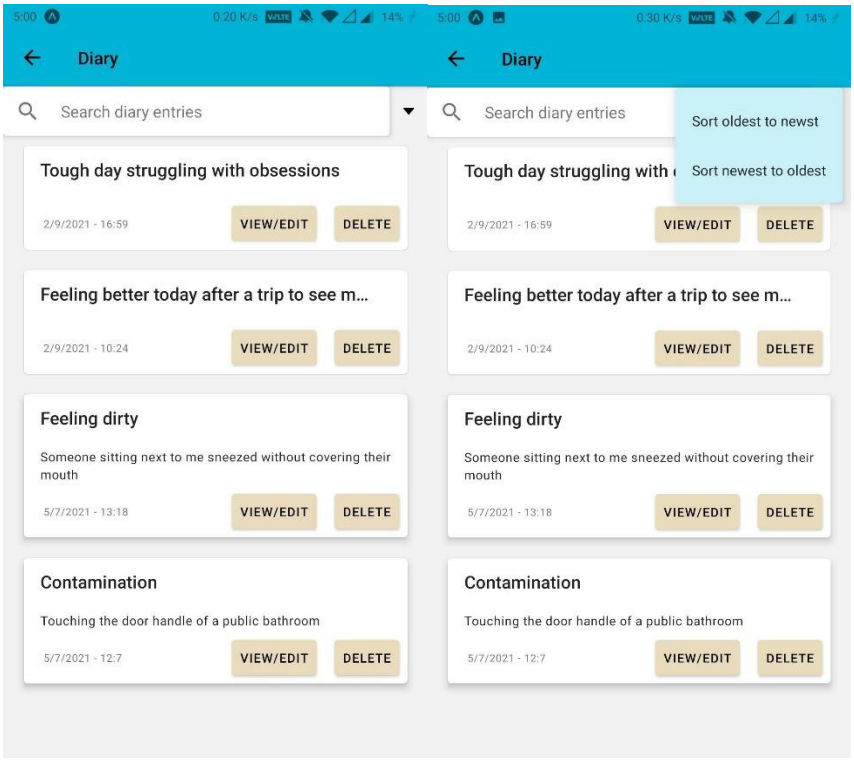


Figure 39 - Diary screen

Each of the tiles seen are a custom component “DiaryTile”. This component wraps the React Native Paper component Card and handles the formatting and passing of data to the Card component. The DiaryTile components are rendered based on the “data” state variable using a FlatList, a React Native component that is used to render a scrollable list of components. FlatList is a very useful component for my purposes, as it can take any number of items and render them into a scrollable list, meaning that however large someone’s diary is it can all be easily accessed. In addition to this it renders the items lazily, meaning that it only loads items that are about to be displayed on the page. This is a great advantage over the similar React Native component ScrollView, which renders all items in a list whether they are being displayed or not, which could lead to performance issues for a user with a large diary.

Sorting the diary

In figure 39 above, within the hamburger menu, the sorting options can be seen. Unfortunately, I didn’t have time to fully implement this feature. As a proof of concept, the diary is sorted by the “createdAt” timestamp that is automatically applied to diary entries and check-ins when they are submitted to the database. The diary is sorted by sorting the array that the FlatList renders the DiaryTiles from; as the array is a state variable, the view automatically updates with the new sorting.

In a fully developed feature, diary entries should be sorted by the date field entered by the user, and more sophisticated features should be implemented as defined in the requirements, such as being able to select a range of dates.

Deleting an entry

If a user wishes to delete an entry, they can click the delete button on the diary tile. This brings up a confirmation dialog where they can confirm their action. When the user confirms, an HTTP delete request is sent to the back-end, where the entry is deleted from the database. If the entry is successfully deleted, a message will be displayed to the user. The entry is also removed from the state variable “data”; as the FlatList renders the DiaryTiles from “data”, the page updates without having to refresh.

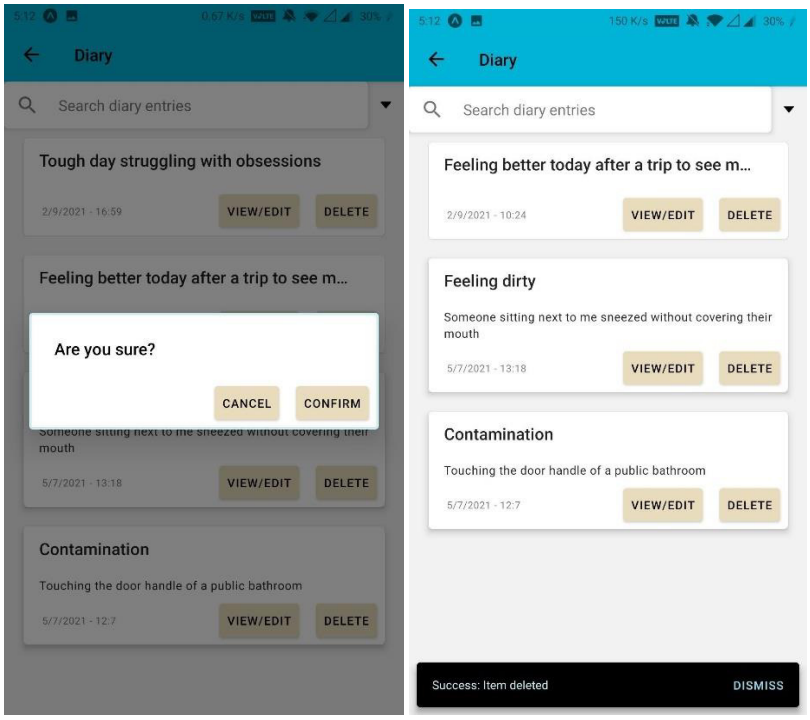
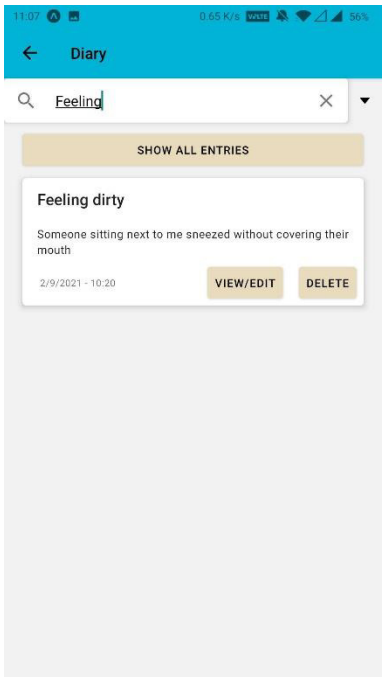


Figure 40 - Deleting a diary entry

Searching the diary

Users can search their diary entries using the search bar at the top of the page. As shown in figure 41, when the search button is clicked, a request is made to the back-end with the search term, and if the user is a therapist, the ID of the patient whose diary is being searched. If there are no results then a message is displayed to the user saying so, and if there are results then they are displayed to the user by setting the state variable “data”, with the FlatList automatically updating the DiaryTiles shown to the user. To see their diary again, the user can

click the “show all entries” button, the visibility of which is controlled by the state variable “searchDisplayed”.




```
async function search() {
  if (searchTerm.length == 0) {
    alertBar.handleAlert("Error: Please enter a search term");
    return;
  }
  try {
    let path;
    isTherapist
      ? (path = `searchPatientsDiary/${searchTerm}/${route.params.patientId}`)
      : (path = `searchDiary/${searchTerm}`);

    const response = await axiosInstance.get(path);
    console.log(response.status);
    if (response.status == 204 || response.status == 404) {
      alertBar.handleAlert(
        "Error: No results found for that search term"
      );
    } else {
      setDATA(response.data);
      setSearchDisplayed(true);
    }
  } catch (err) {
    alertBar.handleAlert("Error: error while searching");
    console.log(err);
  }
}
```

Figure 41- Example diary search and code

7.4.9 Therapist mode

To be efficient and avoid the repetition of code, many of the same screen components are used for patient and therapist users, with conditional rendering to slightly change to contents. The context variable “isTherapist” holds the user type and is used for this conditional rendering. Context was used for this variable, rather than state, to avoid having to manually pass the variable down through many layers of components to where it was needed, therefore making the app much easier to maintain and more scalable. Context should not be overused as it makes it harder to create re-useable components. However, for variables such as “isTherapist” that are used across many components, its use is of great benefit (React [no date][a]). An example of conditional rendering can be seen below in figure 42; when the user is not a therapist a “Delete” button is displayed on the diary tiles, when the user is a therapist the “Delete” button is not shown.

Figure 42 - Conditional rendering based on isTherapist context

Another example of this method can be seen in figure 31, where text inputs in the diary entry are not editable if the entry is being viewed by a therapist user.

7.4.10 Styling

When calling components in React Native, styles are passed to components using the style prop. In a small project it might make sense to simply define each style wherever it is, as this is simple to do and means the style and the component are found in the same place. However, in a larger project this becomes very inefficient, as there will be a lot of repetition of code and searching through code to find where styles are set. Therefore, all of the major styles for the app are contained in src/styles, with a separate file for each category, for example button.js, input.js (show in figure 43). Each of the groups of styles are imported into index.js and then exported again (figure 44), meaning that all of the style objects can imported from one file (figure 45). This structure was taken from Schoeman and Larsson (2019).

```
export const input = {
  color: "black",
  margin: "2%",
  borderWidth: 1.5,
  borderRadius: 3,
  padding: 10,
};

export const small = {
  height: 40,
};

export const medium = {
  height: 110,
};

export const large = {
  height: 150,
};

export const dialog = {
  backgroundColor: "white",
  borderStyle: "solid",
  borderWidth: 2,
  borderColor: "#CAF0F8",
};

export const focused = {
  borderColor: "#00B4D8",
};
```

Figure 43 - Styles for inputs

```
import * as button from "./button";
import * as card from "./card";
import * as input from "./input";
import * as selection from "./selection";
import * as text from "./text";
import * as surface from "./surface";
import * as layout from "./layout";
export { button, card, input, selection, text, surface, layout };
```

Figure 44 - index.js

```
import { selection, input, text } from "../../Styles/index";
```

Figure 45 - Example of importing styles

This method makes it easy to manage styles and is suitable for a large project to avoid repetition of code. Having styles defined in small sections also allows for the composition of multiple styles. For example, when styling a TextInput component, you include “input.input”, then choose the size, e.g. “input.small”, and then conditionally add “input.focused”. This keeps each of the styles simple and makes it easy to customize the appearance of components.

7.4.10 – Additional front-end features

Screenshots and brief explanations of the front-end features not yet described (signing in, singing up, viewing cognitive error information, adding a patient, removing patient, viewing list of patients) have been included in the appendix A. As these features have all been implemented as, or very close to, described in section 6 and use similar methods as have already been described no value would be gained by describing them thoroughly.

8 – Evaluation

8.1 – Test Cases

To assess the functionality of the system's main features, test cases covering all of the implemented requirements have been produced; these can be found in appendix B. Test cases are a useful tool for system evaluation as they help to remove bias in product testing by defining a strict set of steps with specific data to input into the system. Three types of data were used in the test cases: valid data, invalid data, and absent data. With each data type, the test cases check that the system correctly handles the given input and provides an appropriate response to the user.

Of the 17 test cases, there were 13 passes and 4 partial passes (cases 8, 11, 12, 13). Two of the partial passes are related to submitting diary entries and check-ins - if the user spams the submit button then the document will be saved into their diary multiple times, which is not an intended feature. The other two partial passes are related to the sliders within diary entries and check-ins - when viewing a past diary entry or check-in, the sliders all show a zero value rather than the value previously selected.

Overall, however, these test cases outline the fact that the system is robust in its handling of invalid and absent data. Of course, as the test cases were only written for implemented requirements, this result does not reflect a 100% complete program. This will be discussed section 8.2.

8.2 - Acceptance criteria results

To further evaluate the system, the two tables below judge the functional and non-functional requirements by a Pass/Fail metric. Of the 25 "must have" functional and non-functional requirements, there were 6 failures and one partial failure, giving a pass rate of 74% (counting the partial failure as a half). While implementing the app I focused my efforts on the more vital requirements (e.g. creating and editing diaries and check-ins, therapists viewing patients' diaries) first, leaving less important features until later on in the process. This approach was required for two reasons. Firstly, a lack of knowledge of the complexity of each requirement at the time of writing meant that some requirements, such as statistics, ended up being out of the scope of the project, and secondly the limited time available for the implementation of the requirements. Therefore, while this success rate leaves room to improve, the

requirements that have been implemented represent the most important functionalities of the app.

8.2.1 Functional Requirements

No.	Requirement	Pass/Fail	Test case(s)
Must have			
1	The system will allow patient users to write a diary entry	Pass	03-08
2	In the diary entry the patient can include information about an activating event, associated intrusive and obsessional thoughts, and their related anxiety	Pass	05
3	In the diary entry the patient can include information about cognitive errors and their faulty beliefs	Pass	06
4	In the diary entry the patient can include information about alternative ways to interpret their intrusive thoughts	Pass	03
5	In the diary entry the date and time should be automatically filled, although the user should be able to edit them.	Pass	04
6	The system will allow patient users to search through their previous entries	Pass	10
7	The system will allow patient users to view all the details of a previous diary entry	Pass	11
8	The system will allow patient users to edit their previous diary entries	Pass	11
9	The system will allow patient users to view their previous entries and filter and sort by date	Partial pass	09
10	The system will allow user to select entries to become private, which their therapist can't see	Fail	-
11	The system will require all users to create an account with an e-mail and password	Pass	01
12	The system will allow users to change their password	Fail	-
13	The system will allow patient user to perform a check-in	Pass	12
14	In the check-in the patient can choose and rate their emotions from a list	Pass	12
15	The system will allow patient users to edit the list of emotions	Fail	-
16	In the check-in the patient can choose what activities they have been doing	Pass	12
17	The system will allow the patient users to edit the list of activities	Fail	-
18	In the check-in the patient can add a note	Pass	12
19	The system will allow therapist users to view their patients	Pass	
20	The system will allow therapist users to add/delete patients to their list	Pass	15, 16
21	The system will allow therapist users to see and search through all of the non-private entries of each of their patients	Pass	17
Should have			
22	The system should have a function to provide information about cognitive errors to the patient users	Pass	14
23	The system should have a function to display summary statistics to the users	Fail	-
24	The system should have a function to display useful comparisons to patient users	Fail	-

Could have			
25	In a diary entry, the patient user could be able to tag their location	Fail	-
26	The system could send a notification to the patient user if they have not done an entry in an amount of time specified by the user	Fail	-
27	The system could let patient users save common items such as people and places so they can be referenced between diary entries	Fail	-
28	In a diary entry, the patient user could be able to add a photo	Fail	-

8.2.2 Non-Functional Requirements

No.	Requirement	Pass/Fail
Must have		
1	The system must provide example inputs for each section of the diary	Fail
2	The system must explain the purpose of each section of the diary	Fail
3	The system must be easy to use, navigate, and understand	Pass
4	The system must be reliable	Pass
Should have		
1	The system should be fast to respond to user input	Partial Pass

8.3 - User testing

To property evaluate my product it is vital to gain feedback from prospective users. As the developer of the application, I have my own biases about it and need to gather the thoughts and opinions of others, to make sure that I have made a product suitable for the intended audience (as laid out in my 'user personas' in section 3.3).

Two methods of user testing were incorporated into this project. Firstly, prospective users of the app were questioned using the System Useability Scale (SUS) and three written questions (what they liked about the app, what could be improved, and additional comments). As the intended user population for the application is quite specific, it was not feasible to try and carry out the SUS testing in person. Separate observational testing was therefore carried out with participants of the general populace, in order to directly observe the participants using the app.

8.3.1 – System Useability Scale Testing

The System Useability Scale (SUS) was created by John Brooke in 1986, for the purpose of measuring the useability of terminal applications of the time. The SUS consists of ten statements, such as "I found the system unnecessarily complex" and "I found the various functions in this system were well integrated", which the user rates on a scale of 1-5 from strongly disagree to strongly agree. The full list of questions can be found in appendix C.

While originally being described by Brooke as “quick and dirty”, the SUS has since been shown to be a very useful tool in useability evaluation. Sauro (2011) summarises that the SUS is:

- Reliable: Results of a SUS are repeatable as users respond in a consistent manner. SUS is also reliable in small sample sizes (as small as two participants), although small samples do not necessarily accurately predict the SUS score of the whole user-population.
- Valid: Useable and unusable systems can be successfully identified using the SUS. Answers to the SUS correlate highly with other usability questionnaires.
- Not diagnostic: The SUS can tell you what is wrong with the system, only that something is wrong.

Users were provided with dummy accounts and instructed to complete a series of tasks in the application, covering each of the use cases of the app. For patient users the tasks were: complete a diary entry, view and edit a diary entry, complete a check-in, and read about a cognitive distortion. For therapist users these tasks were: add a patient to your list, remove a patient from your list, View the diary of a patient, and search through diary entries of a patient. Participants were not given detailed instructions for the tasks, in order to replicate an authentic first-time experience of the app.

SUS scores for each participant are calculated by standardising the score for each question to 0-4, with 4 being a positive response, totalling the scores for each user, and then multiplying this by 2.5 so the scale is 0-100. To calculate the overall SUS score, a mean average of the scores of each participant is calculated.

In total five participants were recruited; four people with OCD, and one therapist. The complete results for each question can be found in appendix C. The calculated SUS score for the application was 87.5, a grade of “B”, or description of “Excellent” according to the metrics in Bangor et al. (2009). This is a very positive result and indicates that the application does not have any major useability issues.

While this is an encouraging result, and the SUS is reliable with small sample sizes, I was only able to recruit five participants for my study, which may not be enough to be able to reliably draw conclusions for the whole user population.

8.3.2 - Long answer questions

Alongside the SUS, the participants were given the opportunity to give written feedback on the app. As the SUS is not diagnostic this is a valuable addition, with participants being able to identify and talk about specific areas of the app. Three questions were asked:

1. “Are there are any areas of the application that think are particularly useful? If so, why?”
2. “Are there any areas of the application that could be improved? If so, how”
3. “Do you have any additional comments about the application?”

Whilst full text of the participants’ answers can be found in appendix C, the feedback is summarized below.

Design

Aligning with the SUS scores, multiple participants praised the general design of the app, stating that it was simple and easy to use. Three participants commented on the use of tabs for the diary entry screens; two stated that they weren’t particularly obvious, and one mentioned that when looking at a previous diary entry they would prefer it to be shown on one page, as opposed to over three tabs. Finally, one participant highlighted the fact that with a relatively large diary, flicking through a physical journal would be much easier than scrolling and loading entries in the app.

Diary content

One participant praised the questions in the diary as “provoking”, and the therapist specifically liked the inclusion of “cognitive errors identification”, although suggested the use of the term “unhelpful thinking styles” instead as it is more accessible. The therapist also stated that gathering evidence for and against their thoughts is often what patients find the most helpful, and while this is implicit in the last question of the diary, it may be useful to make it more explicit or even include a separate thought gathering exercise. Two users identified that for the multiple-choice questions, it would be useful to be able to edit the lists of emotions, cognitive errors, and activities, a feature included in my functional requirements that I didn’t have time to implement. Finally, the therapist and two other participants recognised the need for examples/guidance for the questions of the diary, another feature I had originally planned to include.

Functionality

Three participants, one being the therapist, expressed that they liked the ability for the therapist to directly view the diaries of their patients. The therapist said this feature would be extremely useful to them as currently they have to talk through their patient's diaries on the phone, are lucky to get a document sent to them, and sometimes get patients who lie about how much work they have done, which is a barrier to their treatment and recovery but would not be possible if they were using the app.

Two participants commented that the app was sometimes slow to respond to their inputs, one stating that when submitting an entry they were able to click submit multiple times and get repeat entries in their diary.

Finally, the therapist suggested that it would be useful to be able to search through the diary by date/time period, a requirement which I intended implement.

8.3.3 – Observational testing

As previously stated, the SUS testing could not be carried out in person, so observational testing has also been undertaken. This is important to show directly how users interact with my app, which parts are intuitive, which are not, and how my expectations of the UX differ from reality. Four participants were recruited for the testing to cover the four following categories: younger (<35) with good IT Skills, younger with poor IT skills, older (>35) with good IT skills, and older with poor IT skills. These categories allowed me to observe a wide range of users, covering my user personas defined in section 3.3.

The full observation sheets can be found in appendix D, and the findings are summarised below:

- When completing a diary entry, three participants tried to scroll down on the page with the keyboard open when they had finished one textbox in order to click on the next one. They all then realised they could not scroll and have to close the keyboard to click the next textbox.
- The same three participants took a moment to recognise the tabs after they had completed the first page of the diary entry. Subsequently when completing a check-

in, they did recognise that the tabs worked in the same way as the diary and used them straight away.

- When updating a diary entry, all four participants scrolled to the last tab to press “Update”, rather than using the “Save & Exit” button in the hamburger menu. This was probably due to the fact that the “submit” button when creating a diary was in the same place, so this is where they knew to go.
- When instructed to edit the diary entry they had made, all participants realised they had to click “View diary” and then go on the specific diary entry.
- When the app is communicating with the database, i.e. when submitting a diary entry, there is a small (~1-2 second) delay. In this gap, all of the participants paused waiting for a response for their input. Some instant feedback here such as a loading symbol could be useful.

8.3.4 – Testing summary

Overall, the testing has been very positive and informative. In the SUS testing the participants voted in favour of statements such as “I thought the system was easy to use” and “I think I would use this system frequently”, and the overall score of 87.4, although from a small sample size, indicates a very useable app. When suggesting improvements some participants identified features that I had intended to include in the app: the ability to edit the lists of emotions, activities, and cognitive errors, the need for examples and extra information for the questions in the diary, and the ability to search through the diary by date/time period. Participants also highlighted the fact that the tabs in the diary entry were not immediately obvious, something that the observational testing corroborated. Additionally, two SUS participants commented that the app was sometimes slow to respond to their inputs; when observing users I could see that this was when they were performing actions such as submitting a diary entry which require the app to interact with the database. Finally, it is very encouraging that participants made comments that align with the problems with physical journals outlined in section 3 (e.g. security of journal on phone, ease of use having journal online, easily sending entries to therapist).

9 – Conclusion

This project aimed to produce a thought journal mobile application to assist patients and therapists in the process of treating OCD though CBT. Reviewing the existing literature highlighted issues with physical journals; for example, it can be difficult for the therapist to view their patients' journals and inconvenient for the patients to have to carry a physical journal with them. Existing solutions for these problems were reviewed and it was found that while there were many general mental health journal apps, there were none specifically for OCD and none with the ability to easily share a journal with a therapist.

The main objectives for the project were:

1. To allow users to add a journal entry describing a triggering event along with all relevant aspects of the event and the subsequent thought processes
2. To allow therapists to view and edit their list of patients, and view and search their patients' journal entries
3. To allow users to view/edit/delete their past journal entries in a searchable list, which can be filtered in ways such as date or time period
4. To allow users to perform a quick check-in of their emotions and activities
5. To allow users to read and learn about cognitive distortions

The application meets all of these objectives, although there are some shortcomings in relation to the finer details set out in the requirements. Due to time constraints, there were some "must have" requirements which I was unable to implement: a statistics feature, the ability to edit the lists of emotions, activities and cognitive errors in the diary entry and check-in, the ability for users to change their password, filtering the diary by a time period, and including example inputs and additional information with the questions in the diary. In retrospect these would have been better classified as "should have" features, as the core functionality of the app works, and the five main objectives have been met.

Throughout the design and implementation stages, user experience and my user personas were kept in mind. Choices such as minimizing the number of required fields, using standard, recognisable material design components, and keeping a consistent styling throughout the app, all contributed towards making the app as user friendly as possible.

User testing highlighted some areas for improvement in the app, but overall produced very positive results which indicated that the app has no major useability issues and is fit for purpose. The positive feedback demonstrates the value of online journaling for OCD patients. The importance of the findings of this dissertation is that CBT can be enhanced, both for people with OCD and their therapists when technological solutions are designed with users at the centre of the development process.

10 – Future work

As it stands, the application is functional and achieves most of the requirements set out at the start of the project. There are, however, some missing features and some improvements that could be made to the implemented features. These would greatly improve the standard of the app, and there are some that could be achieved in a short amount of time.

Firstly, I would like the refactor some of the code. As I was learning while coding, there are some sections of the code which could be made more efficient and some sections where repetition could be avoided.

Two of the failed non-functional requirements were to provide example inputs for, and explain, the purpose of each section of the diary. Not including these requirements means the app doesn't cater very well to my first user persona (Section 3.3), a user who doesn't have much previous in journaling. Adding these small bits of information would make the app much more useable for those without much knowledge of CBT, increasing the potential user base. Another easy improvement would be to finish the section of the app containing information about cognitive errors; as the framework is already in place, this would only require typing out the information. Additionally, being able to view the relevant cognitive error information page straight from the diary when selecting it would be a nice addition.

Through the user testing it became apparent that the use of tabs in the diary entry and check-in screens was not optimal, with many users taking some time to recognise how to get to the next page. This could be improved in a number of ways; changing the colour of the tabs so they stand out more, placing them at the bottom of the screen where users' attention is after they have filled out the last field of each page, or adding "next page" and "previous page" buttons to each screen. There were also a few other smaller UI issues that would need to be fixed and put back to user testing for further evaluation.

Another issue discovered in the user testing is the delay in response when the app is communicating with the database. This may be due to the fact that I am using free resources to host the back-end of the program, and by upgrading these to paid versions the issue may be solved. Another way this could be resolved is to simply have some instant feedback in the app, such as a loading symbol, when a user initiates one of these actions.

One of the requirements which I was unable to implement was the ability for users to edit the lists of emotions, activities, and cognitive errors. This is an important feature as its impossible to predict all the different things a user may want to record in the app, and including it therefore greatly increases the utility of the diary.

Additionally, a more secure way for therapists to add patients would be needed; in its current format, the app only requires someone's e-mail to be able to view their diary. To make this process more secure patients could be given a unique code that is generated when they create their account, which they would then provide to their therapist to be able to add them. There could also be a way for patients to generate a new code in order to re-set the therapists that have access to their diary.

Further to these improvements of existing functions, there are some additional features which could be added to the app on larger timescale. Firstly, a statistics section. This was a requirement of the app and a common feature of the reviewed existing solutions, but could not be implemented within the timeframe of the project. A second additional feature is an offline mode. Currently, using the app requires an internet connection, which could be inconvenient in the real word. If users were able to save entries on their device when they didn't have an internet connection, and sync up to the database when they did, this would be a very helpful feature.

Finally, there are many more activities involved in CBT than journaling, such as exposure and response prevention. Expanding the app to include these could greatly increase its utility in a therapy setting.

11 – Refection of learning

At the start of the project, I had no knowledge of React Native, node.js, express.js, or MongoDB. While this challenge was exciting as it gave me the opportunity expand my technical knowledge, it was also quite intimidating to take on a big project using frameworks that I had never used before. While learning these frameworks at the start of the project I found that I wasn't always learning things that I was going to be able to apply to the project. For this reason, I started the implementation once I had learnt the basics and simply did further research where required, an approach which I found to be very useful in streamlining my learning. Taking on this challenge has increased my confidence in my ability to quickly learn and apply new skills, and I'm very happy to have chosen to do so.

My lack of knowledge of these frameworks at the start of the project did produce some issues during the project, particularly during the planning stages. Firstly, when writing the requirements, I found it hard to gauge the complexity of each of the features that I wanted to include in the application. This led to an overly ambitious set of requirements which could not all be implemented in the time available. Secondly, particularly during the implementation of the front-end in React Native, I found it difficult to approach the development in an efficient manner. My lack of knowledge sometimes led me down the wrong path when trying to solve problems or implement features in ways which then had to be changed later down the line. While learning through lectures and guides before a project teaches you some things, nothing beats the experience gained through undertaking a large project such as this. With my gained knowledge I feel much more confident going forwards to future projects in my ability to define requirements that match the timescale of a project and plan the implementation in a more methodical manner.

Another area in which I have learnt a lot is UI design, an area in which I previously had very little experience. I learnt the importance of iterative design and critical analysis of prototypes using the Nielsen's 10 heuristic principles, and gained insights from this which were influential in the final design of the app. During the evaluation, the observational study was an eye-opening experience. I had never before seen someone use an interface that I had created. Most interestingly was the comparison between the way I expected people to interact with the app, and the reality that I observed. Therefore, retrospectively, I should have included user testing in the project at an earlier stage to give me the opportunity to incorporate the

feedback into the project, something that I was hesitant to do due to worries about time constraints, as well as an underestimation of its importance.

Towards the end of the implementation phase I found it hard to stop coding and move on with the project. This issue stemmed from two reasons; firstly, a desire to fix all the small issues with the app, not wanting to leave bugs unresolved. Secondly, not having a strict definition of a minimum viable product or a definite date to stop at. Unfortunately, this meant that I was pushed for time at the end of the project, and has highlighted the importance of a stricter time schedule for large projects.

The process of this project has been an experience full of learning and personal and academic growth. The project has been an order of magnitude more involved and demanding than any of the other work completed throughout the MSc, and the technical and management skills acquired will be invaluable in my future.

12 – Reference List

Agarwal, M. 2018. What is JSON Web Token. Available at: https://www.loginradius.com/blog/async/jwt/ [Accessed: 2 November 2021].

Arias, D. 2021. Hashing in Action: Understanding bcrypt. Available at: https://auth0.com/blog/hashing-in-action-understanding-bcrypt/ [Accessed: 2 November 2021].

Auth0 [no date]. JSON Web Tokens. Available at: https://auth0.com/docs/security/tokens/json-web-tokens [Accessed: 2 November 2021].

Billings, D. 2006. Journaling: A Strategy for Developing Reflective Practitioners. Kowalski, K. ed. The Journal of Continuing Education in Nursing 37(3), pp. 104–105. doi: 10.3928/00220124-20060301-02.

Cherry, K. 2019. The Color Psychology of Blue. Available at: https://www.verywellmind.com/the-color-psychology-of-blue-2795815 [Accessed: 2 November 2021].

Cimpanu, C. 2021. 8.3 million plaintext passwords exposed in DailyQuiz data breach. Available at: https://therecord.media/8-3-million-plaintext-passwords-exposed-in-dailyquiz-data-breach/?__cf_chl_jschl_tk__=pmd_qQlE5P6gIsY5BZ8u8EZtuhLQzylhsYhkvBtjhTMihBQ-1635164583-0-gqNtZGzNAnujcnBszQi9 [Accessed: 2 November 2021].

Diduh, A. 2021. Mobile app vs. website: What to choose for the business in 2021. Available at: https://www.cleveroad.com/blog/mobile-app-vs-mobile-website [Accessed: 2 November 2021].

Expo [no date]. Already used React Native? - Expo Documentation. Available at: https://docs.expo.dev/workflow/already-used-react-native/ [Accessed: 2 November 2021].

Feroze, U. 2021. React Native CLI vs Expo CLI — Which one do I choose? Available at: https://levelup.gitconnected.com/react-native-cli-vs-expo-cli-which-one-do-i-choose-bdf02ea457bf [Accessed: 2 November 2021].

freeCodeCamp.org 2021. Node.js and Express.js - Full Course. YouTube . Available at: https://www.youtube.com/watch?v=Oe421EPjeBE&t=7096s [Accessed: 2 November 2021].

Get self help 2010. OCD / Perfectionism - Thought Record Sheet. Available at: https://www.get.gg/docs/OCDThoughtRecordSheet.pdf [Accessed: 2 November 2021].

Habilelabs 2021. Best Folder Structure for React Native Project - Minds Verse - Medium. Available at: https://medium.com/habilelabs/best-folder-structure-for-react-native-project-a46405bdba7 [Accessed: 2 November 2021].

JavaScript Mastery 2020a. Full Stack MERN Project - Build and Deploy an App | React + Redux, Node, Express, MongoDB [Part 1/2]. YouTube . Available at: https://www.youtube.com/watch?v=ngc9gnGgUdA [Accessed: 2 November 2021].

JavaScript Mastery 2020b. Full Stack MERN Project - Build and Deploy an App | React + Redux, Node, Express, MongoDB [Part 2/2]. YouTube . Available at: https://www.youtube.com/watch?v=aibtHnbeuio&t=2553s [Accessed: 2 November 2021].

King, F. and LaRocco, D. 2006. E-journaling a strategy to support student reflection and understanding. Current Issues in Education 9(4)

MongoDB [no date][a]. Avoid Unbounded Arrays. Available at: https://docs.atlas.mongodb.com/schema-suggestions/avoid-unbounded-arrays/ [Accessed: 2 November 2021].

MongoDB [no date][b]. Query an Array. Available at: https://docs.mongodb.com/manual/tutorial/query-arrays/ [Accessed: 2 November 2021].

Patel, P. 2018. What exactly is Node.js? Available at: https://www.freecodecamp.org/news/what-exactly-is-node-js-ae36e97449f5/ [Accessed: 2 November 2021].

React [no date][a]. Context – React. Available at: https://reactjs.org/docs/context.html [Accessed: 2 November 2021].

React [no date][b]. React.component – React. Available at: https://reactjs.org/docs/react-component.html#setstate [Accessed: 2 November 2021].

React [no date][c]. Tutorial: Intro to React – React. Available at: https://reactjs.org/tutorial/tutorial.html [Accessed: 2 November 2021].

React Native 2021. Introduction · React Native. Available at: https://reactnative.dev/docs/getting-started [Accessed: 2 November 2021].

Shaffer, J. 2016. 5 tips on designing colorblind-friendly visualizations. Available at: https://www.tableau.com/about/blog/examining-data-viz-rules-dont-use-red-green-together [Accessed: 2 November 2021].

Stone, A., Shiffman, S., Schwartz, J., Broderick, J. and Hufford, M. 2002. Patient non-compliance with paper diaries. BMJ 324(7347), pp. 1193–1194. doi: 10.1136/bmj.324.7347.1193.

Taylor, D. 2021. What is MongoDB? Introduction, Architecture, Features & Example. Available at: https://www.guru99.com/what-is-mongodb.html.

The Net Ninja 2020. Node.js Crash Course Tutorial #9 - MongoDB. YouTube . Available at: https://www.youtube.com/watch?v=bxsemcrY4gQ&t=304s [Accessed: 2 November 2021].

Think CBT 2017. OCD Thought Record. Available at: https://thinkcbt.com/images/Downloads/Thought_Records/OCD-THOUGHT-RECORD-THINK-CBT-V-09.07.18.pdf [Accessed: 2 November 2021].

Tull, M. 2021. Managing Catastrophic Thinking in PTSD. Available at: https://www.verywellmind.com/managing-catastrophic-thoughts-2797222 [Accessed: 3 November 2021].