Imperial College of Science, Technology and Medicine
Department of Electrical & Electronic Engineering
Final Year Project Report 2020

Attention-Preserving Technology

Student:	**Darrick Lau**
CID:	**01112514**
Course:	**EIE4**
Project Supervisor:	**Prof. Jeremy Pitt**
Secondary Marker:	**Prof. R. Spence**

Submitted in part fulfilment of the requirements for the degree of
Master's of Engineering in Electronic & Information Engineering
Imperial College London, 1st February 2021

Abstract

Smartphone overuse and internet addiction is strongly correlated with a sharp rise in mental health problems and reduced productivity. Social media and gaming applications ("apps") earn money off of users' screen time and thus leverage techniques from behavioural science to cause users to form unconscious habits which lead to increased usage. This greatly detracts from an individual's productivity and ability to focus on difficult tasks.

Many attempts have been made to develop technologies which help individuals reduce these effects in the form of well-being apps. The most effective well-being apps aim to use the same techniques from behavioural science used to hook users to reverse their own effects. This project seeks to continue the work of one such well-being app, closing the gap in product quality and feature offerings by leveraging recent advancements in application development methods.

The new app will be developed in collaboration with a consultant neuroscientist, and will be a novel effort in exploring the translation of behavioural "hooks" into a set of software engineering requirements that can be implemented in an app. The resulting work is a highly interdisciplinary blend of psychology, neuroscience, software engineering and user experience design.

On top of enhancing the original product and its applications in behavioural rehabilitation with new features, the resulting work facilitates developer on-boarding, reduces development time and implements analytical methods for developers to quantitatively assess how effective new features are at helping users improve their focus and productivity.

Acknowledgements

I would like to express thanks to:

- My supervisor, Prof. Jeremy Pitt, who has provided essential guidance and encouragement throughout the scope of this project, and also did a wonderful job of coordinating all the collaborations across experts in different disciplines.

- Our neuroscience consultant, Dr. Thomas Dannhauser, whose efforts are a significant driving force in the successful implementation of all the features in this project.

- My second supervisor, Prof. Emeritus Robert Spence, whose class I was unexpectedly dropped into in my first year of university and will carry with me for the rest of my career as a developer.

- Leon Wiederkehr, who worked on this project before me and created the initial android release of the app on top of the literature review and motivation for the original project, creating a starting point for the fascinating exploration into multiple fields that this project seeks to undertake.

- Olly, Ethan, Naim, Seb, Yusuf, Hardik, LH and Ollie, who are the last things holding up my mental health while undertaking this project during the COVID-19 pandemic.

Dedication

This work is dedicated to all my fellow procrastinators who set out on a a productive day only to lose time to an unplanned YouTube binge. It happens to us all, and we owe it to ourselves to be aware of how the subtle ministrations of powerful organisations are designed to influence our lifestyles for their profit.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

One does not need to quote academic research to know that the smartphone has been one of the most beneficial additions to modern society. The smartphone and the rich array of features it brings has changed the way many industries operate and increased speed of communication and efficiency. Banks and utility companies no longer need to waste resources sending letters, colleagues can have video conferences at the drop of a hat and friends and family are just a message away.

However, some of the features that bring us such joy are the very same features shown to have a very detrimental effect in some users. Social media, emails and games are a constant distraction that exist on the very same device that is used for work and productivity, be it a smartphone or computer. There is much compelling scientific literature suggesting that the overuse of smartphones and the effects of its addictive applications is highly correlated to increased levels of sleep disorders, depression, anxiety and lower life satisfaction [1, 2, 3].

In this report a particular focus is given to the negative effects it has on the individual's attention capacity, motivation and focus. While some references will be made to extreme cases of highly-addicted audiences, the general demographic under study is the average adult, ranging from college students to working professionals.

1.1.1 Problem Statement

The presence of a smartphone or otherwise connected device has shown to have adverse effects on one's productivity and ability to stay focused [4]. This is especially so for individuals whose work involves many hours at at desk, working closely with a connected device, unable to decouple their personal devices from doubling-up as a productivity and communication tool for work.

This digital dependence requires the individual to constantly exercise some level of self-restraint to prevent distraction from the priority task at hand, and this phenomenon is not at all coincidental. This report will show that the most successful modern smartphone applications are addictive by design and have very successfully secured a place for themselves in their users' lifestyles. The user is therefore, without external education and intervention, quite effectively helpless to identify and remedy the negative effects of smartphone use that have crept into their lifestyle. In extreme cases, this may result in what is generally known as an Internet Addictive Disorder (IAD).

1.1.2 User Attention as a Commodity

Social media and gaming applications (henceforth abbreviated as "apps") generate profits by retaining the attention of a user. Social media apps might monetize this by generating advertising revenue off of a user's screen time, or offer the user additional premium features requiring an in-app transaction to unlock.

Gaming apps have evolved to follow this model of monetizing continued user engagement: games in the early 2000's delivered a full product on an upfront purchase, seeking to deliver an experience that would encourage users to buy a sequel, but modern games now attempt to monetize continued user playtime by offering "microtransactions" in the form of in-game cosmetics or bonuses on top of the base game purchase.

As of June 2020, of the 5 companies to have ever been valued over a trillion dollars, 4 of them are technology companies that profit off of user engagement with their products: Apple ($1568b), Microsoft($1505b), Amazon($1337b) and Google($953b), leaving the winning business models of the decade before such as Walmart($337b) and Johnson & Johnson($366b) in the dust [5]. Even Facebook($629b), whose business revenues are almost purely from monetizing user attention, easily doubles the value of the winning business models of the past.

Twitter (a social media company whose revenue comes primarily from advertising) even goes so far as toting the metric of "monetizable daily active users" (mDAU) in investor press conferences [6], referring to users which the company may show ads to. Investors of such business models focus so heavily on such metrics that, despite beating traditional financial health indicators (such as earnings estimates) by a sound margin in the third fiscal quarter of 2020, Twitter lost 20% of its stock value in a single day simply on the announcement that they had only gained 1 million new users instead of the projected 9 million [7] [6].

Therefore, the new hot commodity of the 21st century is no longer gold, silver or oil, but user attention. The attention, engagement and growth of an app's user base is so lucrative for these companies that they have completely changed how their business models are evaluated and priced.

1.1.3 Addictive App Design

In her book "Addiction by Design", anthropologist Natasha Dow Schüll [8] illustrates the concept gamblers who are in the "Machine Zone", a tracelike state which is an almost perfect state of escapism. Players in this zone are no longer playing for the original thrill of winning, but to simply extend their time in that state. Worse still is the fact that this state may be induced with careful design of the user experience, using concepts such as flow[9] which we shall later explore. The optimal state of profitability for a company is if they can achieve this trance-like state in their users, who will then seek the user experience offered by their app voluntarily instead of the app having to find ways to pull the user's attention.

Social media and gaming apps are therefore carefully designed in line with key concepts from behavioural science to be highly addictive, aiming to capture the user's attention and keep them coming back to interact with the application. This is loosely referred to in the industry as a user's level of "engagement" within the application. To increase and retain user engagement, techniques best described as "addictive software design" [10] are employed, including but not limited to:

- Variable rewards, as in gambling slot machines, maintaining an element of unpredictable positive reinforcement that keeps users coming back for more [10].
- "Bottomless" content with infinitely scrolling lists, experimentally proven to increase consumption [10].
- Ease of use and consistent user-interface elements across apps, allowing users to conveniently feel a sense of familarity when accessing their apps across all platforms (PC, smartphone, tablets and wearables).

Such mechanisms are deliberately crafted to appeal to the primitive parts of the brain responsible for unconscious habitual compliance (the limbic system), rather than those associated with rational, forward and deliberate thought (the prefrontal cortex). This enables the tendency to act on habit and impulse rather than exercise conscious thought. We will see later in greater detail just how these apps exploit and reinforce this tendency to a point which can be extremely destructive.

With the onset of COVID-19 and the rise of the remote workplace, the line between work and play has grown even more vague. Individuals report higher screen time and video game participation which has been correlated to decreased mental health [11]. Smartphones and computers are key pieces of technology that are essential for connectivity and productivity in the age of remote work. The devices themselves do not pose a problem, but the access they give users to the attention-grabbing applications they contain have significant effects on the individual's ability to remain focused at work. The exercise of willpower alone, while possibly yielding some results in the short term, is not the solution to the problem.

1.1.4 Present Solutions

Therefore, many efforts have been made in the fields of neuroscience and psychology to raise awareness of the dangers of digital addiction. Similar addiction-based problems such as gambling and drug use may employ a “cold-turkey” approach, but depriving a modern worker of a smartphone and computer almost completely robs them of function in today’s digital age. One approach to remedy device overuse is, paradoxically, the use of so-called well-being apps, which attempt to address problems caused by internet addiction and overuse.

There are many more well-explored examples of insidious software design and behavioural manipulation to exacerbate internet overuse [12], but the arguments proposed so far sufficiently motivate the creation of a solution that does not require the complete abstinence of smart-device usage.

Many examples of so-called well-being apps have been made, in a paradoxical attempt to design apps that would enhance users’ productivity or state of mental health. Apps such as BlockSite and Hold incentivise users to put their phones away and offer features like blacklisting certain websites and apps. However, [12] highlights that the such apps do not employ the same behavioural “hooks” that social media and gaming apps use. They often yield a short period of initial use, propelled by motivation, and then fall out of favour of their users within a few months.

Therefore, the work previously done in [12] proposes the concept of attention-preserving technology, which leverages the same behavioural science and software engineering techniques from addictive apps to help users stop using them and reverse their detrimental effects. The authors also produced a proof-of-concept: a well-being app called Sage. In summary, their work uses the development of Sage to highlight the failings of the well-being apps that came before it, proposing a set of design principles for future effective attention-preserving app design.

The Sage application helps users regain their focus and productivity through task and time-tracking features. However, given the short development time, it currently exists only as an Android application with limited functionality. Despite this, an initial round of evaluation on the alpha release yielded positive reviews from surveyed testers, claiming that it improved their awareness of their own productivity and that they would continue using the application.

1.2 Objectives

The work done before in [12] achieved a deep literary understanding and formal documentation of the scope of the problem that attention-grabbing apps create. However, the resulting implementation is lacking in features in finesse due to time and developer talent constraints. This report documents the second run of the Sage project, which is much more engineering and implementation-focused. The following objectives are briefly outlined here, and will be elaborated in the implementation documentation later to come.

- Engineering objective: Making a brand new app developed for legacy, with methods that would be more suitable for future lean development teams to continue work on the project. The result will be a high-quality application that is performant, enjoyable to use and built on infrastructure which conforms with the latest development best-practices used in industry.
- Academic / Exploratory objective: Contrary to Sage, this run of the project involves the cooperation of a neuroscience subject-matter expert through the development process, which will highly increase the interdisciplinarity of the resulting work and explores the relation between neuroscience and software engineering in addictive app design.
- Paradoxical social objective: This project seeks to further explore how to further leverage the technologies and methods that cause the problem of digital addiction to reverse the problem. To achieve this, the app has to deliver valuable, attention-preserving features to the user such as to be capable of attracting a loyal, active user base. The previous product implemented only a small set of basic features which are unlikely to keep the app installed on a user's phone for long. The new implementation should implement a much richer feature set such as to warrant continued use from users for long-term iterative development.

In summary, this second run of the Sage project aims to elevate the current product to a level which allows it to compete with the likes of production-grade, attention-grabbing apps built by large enterprise teams. The application will be made available to a much larger user base by re-implementing the code base using cross-platform development frameworks. The app itself will also be re-designed from scratch to deliver a higher-quality, engaging user experience. Improved and automated deployment infrastructure will allow for the rapid monitoring, feedback and iteration of new features for future development. Finally, an attempt will be made to create and implement infrastructure for quantitative evaluation methodologies that allow future product teams to identify which features are most effective at helping users improve productivity and reduce screen time.

Chapter 2

Background

With the goals and objectives of the project clearly laid out, it is now necessary to explore the factors which drive digital dependence. This chapter explores the concepts in biology and neuroscience which cause “hooked” users to become unfocused and unproductive despite their best efforts to do otherwise. It will then highlight a few clear examples of how the design of modern apps leverage these concepts and then do a short review of the existing product.

2.1 Biology Primer

To develop an app to fix the problem of attention deficiency, a short exploration into the mechanics of attention preservation is necessary. Within this scope, our focus will be on two different parts of the brain: the limbic system and the prefrontal cortex.

2.1.1 The Limbic System

The Limbic System carries out many functions that are considered “primal urges”. Emotional responses to food, smell, social cognition, emotional memory and sexual behaviour [13]. All these functions can be summarised and grouped by one common concern: self and species preservation.

The limbic system (henceforth referred to as the “chimp”), is adept at remembering emotional responses to previously experienced situations and inducing the optimal response for self-preservation, should it find itself in a similar one. These responses can be associated with negative emotions, such as the fear one experiences when spotting a predator, or positive, such as the satisfaction and drive upon finding a food source. This is a gross over-simplification of the capabilities of the limbic system, but the message is clear: it is the part of the brain responsible for learning and inducing impulses that it deems will result in the highest chances of evolutionary survival and comfort.

However, with respect to self-control and achieving one’s long term goals, this may be undesirable. The stress hormone (cortisol) produced when working under tight deadlines with complex problems is not unlike that produced in life-or-death situations in the wild. The chimp has no way of differentiating this and therefore the stress response induced by work is biologically registered as an existential threat, which encourages impulses of procrastination or indulging in escapism as part of a “fight-or-flight” response.

2.1.2 The Prefrontal Cortex

The Prefrontal Cortex (PFC) is the part of the brain largely associated with conscious thought and rational self-control. An extremely well-cited study by Miller and Cohen [14] concluded that in general, actions that require critical thought against our base instincts always involve the engagement of the PFC, and that test subjects with PFC impairment performed poorer on such tasks. One such example was the sorting of cards by the colour verbally printed on them, even when the card itself was another colour (e.g. a blue card with the word “red” printed on it would be classified as red).

One prevailing theory is therefore that the PFC and the limbic system are often at odds in tasks involving self-regulation [15] and that successful self-control is a balancing equation of the strength of so-called “bottom-up” impulses and “top-down” conscious thought. This sets the stage for our narrative: that the “chimp” within us and the “human” are always at odds when it comes to work and procrastination. While the PFC may wish to act in favour of our long-term goals, it often loses to the "chimp" and we are left with only a running commentary on our actions, despite our best wishes to do otherwise.

2.2 Self-regulatory Failure

Resisting the urge to check one's phone or play video games is a prime modern example of self-regulatory failure that reflects much in common with addiction-like behaviour in dieters, smokers and substance abusers. App-fueled procrastination may not be as lethal as substance abuse or obesity, but there nonetheless exists a very slippery slope into internet addiction that may ensnare the average person. Heatherton and Wagner elaborate in [15] that there are a few common causes for self-regulatory failure, which can be separated into two categories. "Top-down" factors are associated with reduced capacity or strength of conscious self-control such as negative moods and self-regulatory resource depletion. "Bottom-up" methods refer to unconscious or natural impulses such as cue exposure or lapse-activated consumption.

2.2.1 Resource Depletion & the Strength Model of Self-Control

Baumeister and Heatherton [16] initially proposed in 1996 that conscious self-control against one's impulses draws from a global resource pool that depletes as one uses it further. Since then, many studies have released results in support of this hypothesis. The strength model dictates by construction that it is possible to exhaust one's self-control and must be replenished like any other resource.

This suggests that as long as a distracting alternate stimulus is present, self-regulatory failure is not a question of "if", but "when". For traditional office work settings, this effect is ameliorated by the social norms enforced by the office culture. However, for a person working from home, the distraction unfortunately exists on the same device that enables productivity and there are no sources of social pressure in play to keep the chimp in check.

2.2.2 Cue Exposure

A cue in this context is defined as a piece of sensory stimulus which has an association to a certain consumption-related behaviour. Take for example he smell of food for a dieter or the sight of a beer for a heavy drinker. Cues have been shown to increase cravings, draw attention and increase likelihood of consumption [17]. Furthermore, multiple studies [18] have shown that individuals are often unaware of how cues affect them on a conscious level, and so it becomes difficult for a struggling individual to pinpoint what it is exactly that causes their self-control to lapse.

The first example of addictive software engineering is the push-notification, one of the core tools of user-experience (UX) design. The premise seems simple: software engineers needed a way of bringing a user's attention to key information. This is not a "new" phenomenon, as pagers and message alerts have been around for as long as phones have existed. However, the capabilities of the smartphone take these alerts to a new level. The modern push notification engages many senses: a kinetic vibration (which is audible even if the phone is silent!), an audio alert and a visual alert window.

Both Android and iOS have robust application programming interfaces (APIs) that enable developers to exercise a very high level of control over how their app sends and handles notifications [19]. In the example of a social media message from a friend, this allows for extremely alluring design: an avatar of the friend and a short preview of the incoming message. Users are even empowered to make the experience of receiving a notification as enjoyable for themselves as possible. Facebook Messenger allows users to set nicknames for their friends. Most phones allow users to set specific alert sounds on a per-contact basis, allowing a user to know from audio perception alone who they are receiving a message from. While this feature feels like one that helps the user "filter out the noise", one might argue that it in fact facilitates distraction, by allowing the user to have a stronger, more focused response to a self-set cue.

Cue exposure need not even be as explicit as a push notification. The experience of using a social media app or playing a game is extremely pleasurable. Beautiful colour palettes, highresolution artwork and buttery-smooth animations are a staple in successful apps. All these factors enable the “chimp” to register a positive emotional response to act of phone use in what we call an “internal trigger” (more on this later). Therefore, the very presence of the phone is a form of cue exposure that constantly forces the user to exercise self-restraint.

2.2.3 Lapse-activated Consumption

A 1975 study [20] showed that the consumption of a small amount of an addictive substance (in this case, a milkshake on a test audience of dieters) paradoxically caused dieters to consume more food afterwards, in contrast to the control group of non-dieters who ate less. Similarly, many of us find that picking up our phones for a short break or otherwise directed task can often snowball into a longer-than-intended session which comprised of all manner of actions unrelated to why we originally picked up the phone. Playing “just one round” of a video game often ends up being anything but.

Apps offer extremely dense experiences that engage visual, auditory and even kinesthetic senses that are highly appealing to our physiologies. Levy and colleagues [21] showed that not only is the occurence of a disturbance a problem, but the “richness” of that disturbance was also detrimental to subjects’ performance on cognitive tasks. This explains the unconscious effect that distraction has, and why people find it difficult to go back to work once they have been distracted by a form of smartphone media.

The exact reason as to why lapse-activated consumption occurs is not fully clear even with current research, but the implications of its existence are. Humans are unfortunately predisposed to “falling off the wagon” once an attractive enough stimulus is presented. We will soon explore how smartly designed apps leverage this as an entry point to their user experience and place the user into a continuous loop of habit-forming app use.

2.3 Motivation and Persistence

We have seen so far that the outcome of a self-restraint task comes down to whether the strength of bottom-up impulses exceeds that of our top-down intentions. Most of the time, this tends to be the case. Having explored all the ways in which self-control can fail due to how strong our in-built tendencies are, perhaps then there is a way in which we may use these natural tendencies to our advantage, or at least increase the strength of top-down control.

2.3.1 Strengthening Self-control

Muraven, Baumeister and colleagues [22] conducted a study in which subjects performed consistent amounts of self-enforced exercise over two weeks. While this caused slightly decreased self-control capability for a short refractory period after, this "strength drain" diminished rapidly by the day and eventually resulted in increased success in other completely unrelated self-restraint tasks. Muraven [23] took these results again in a later study and found that "smokers who squeezed a handgrip or avoided sweets for two weeks before quitting cigarettes remained abstinent longer and had fewer lapses overall as compared to smokers who practiced tasks that did not require self-control".

These studies, and many others like it, strongly support the previously mentioned strength model of self-control. Metaphorically, self-control behaves very much like a muscle: its capabilities are slightly diminished following immediate use and while it is possible to fatigue it significantly, it is also possible to increase its capabilities with consistent amounts of progressive load.

2.3.2 Impairing Impulses

Cinciripini and colleagues [24] investigated the effects of schedules and gradual consumption reduction in smoking cessation. Participants were all given uniform education on the physiology of nicotine addiction and methods of coping. Control over consumption was exercised in two forms: scheduling, where participants were allowed to smoke at designated and progressively lengthened intervals, and reduction, where consumption was reduced by a third of the subject's baseline consumption each week. The scheduled and reduced group unsurprisingly performed best, with the greatest reduction in mean cigarette consumption and reduced frequency of urges and severity of withdrawal symptoms. Reduction alone contributed most to the reduction of urge frequency, and scheduling seemed to have an effect on mean consumption.

Results from studies such as these suggest that the controlled appeasement of urges may be a key factor in helping to diminish the effects of bottom-up impulses and serve as a basis for productivity schedules such as the pomodoro technique, which is based on using 25-minute focused intervals of work with short breaks in-between to maximise the extent of a given individual's attention and focus.

2.4 Addictive Design

The principles of behavioural science examined so far dictate that a successful well-being app has to not only help the user form healthy habits that help them avoid self-regulation pitfalls, but also offer new "hooks" to help to keep their inner chimp from becoming overly restless. As a first step in this endeavour, we shall study in this section a few detailed (but not exhaustive) examples of how apps create an addictive experience for their users.

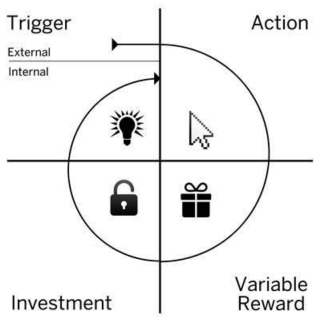


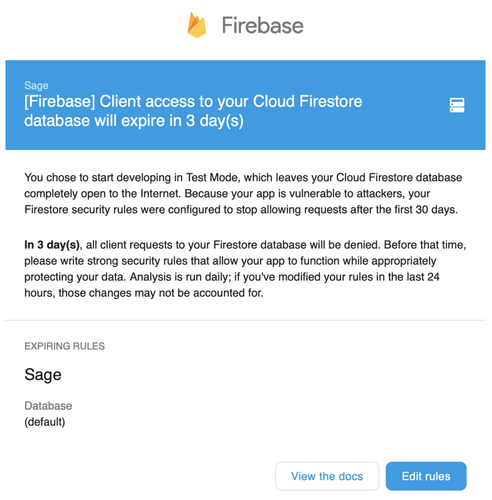
Figure 2.1: The 4-stage Hook Model.

This project uses the concepts laid out by Nir Eyal [25] in his book about habit-forming software design, “Hooked”. He lays out the hook model, which consists of 4 phases: Trigger, Action, Variable reward and Investment.

2.4.1 Trigger

Bearing the exact same meaning as it does in the case of substance addicts, the trigger is any form of cue exposure that is deliberately aimed at getting the user to perform a desired behaviour. In the world of user interaction studies, we call this a “call-to-action”. Eyal broadly separates these into two categories: external and internal triggers.

Anyone who owns a smartphone immediately relates to the concept of an external trigger. A push notification, message, email, or toast-style pop-up within an app all are perfect examples. They bring to our attention a piece of often enticing information, causing us to react. In fact, in the process of trying to make a minimum working example for this project, the wonderful engineers at Google have sent us a perfect example of an external trigger, shown in Figure 2.2.



Firebase

Sage
[Firebase] Client access to your Cloud Firestore database will expire in 3 day(s)

You chose to start developing in Test Mode, which leaves your Cloud Firestore database completely open to the Internet. Because your app is vulnerable to attackers, your Firestore security rules were configured to stop allowing requests after the first 30 days.

In 3 day(s), all client requests to your Firestore database will be denied. Before that time, please write strong security rules that allow your app to function while appropriately protecting your data. Analysis is run daily; if you've modified your rules in the last 24 hours, those changes may not be accounted for.

EXPIRING RULES

Sage

Database
(default)

Figure 2.2: An external trigger from this very project’s cloud service provider.

The logo at the top immediately reminds us of who we are receiving this email from. A calming light blue panel greets us in the middle of a surrounding of off-white, providing high contrast and yet being easy on the eyes. The message is extremely clear, highlighted within the accessible, high-contrast colour panel and enlarged font. The detail text in the middle satisfies the more discerning individual, but is almost completely unnecessary, as the panel below it tells the entire story in a beautifully minimalistic fashion. It tells us *exactly* what is happening, to which database, and even provides clearly described high-contrast elements that are clearly interactable, allowing us to take action immediately. Each element in this panel is given its own generous amount of real-estate on the screen, helping the user to hone their focus and drawing attention to the correct elements.

This email is directed at someone deep in the user journey, not even at "hooking" a new user into the platform. Even then, the amount of attention to detail and incredible design cannot be overstated and the trigger achieves its goal perfectly. Through careful planning, Firebase has successfully anticipated the exact courses of action that a developer might want to take with regard to this piece of information, and has made it as painless as possible for the user to actuate it.

Internal triggers are much more subtle, and are the result of an emotional coupling (remember the limbic system!) of a specific thought or feeling with the product. One can argue that the reason for the massive success of addictive apps is that, similar to drugs, they provide a euphoric escape to the otherwise negative emotions that their users face on a daily basis. Childress and colleagues [26] found that hypnotically-induced negative moods such as anxiety and depression produced significant increases in self-rated cravings in detoxified opiate abuse patients. Similarly, loneliness, anxiety, boredom and depression can be temporarily escaped with the hit of dopamine that the app experience provides (more on this in the section on variable rewards). This conditions an internal trigger in the user: "When I am sad or lonely, talking to friends on Facebook makes me feel less so".

This is the power of the trigger. It is the first step in a continuing feedback loop which motivates or otherwise plant an intent within the user. Triggers may be coordinated and put in place all over the user journey for a variety of different outcomes.

2.4.2 Action

Fogg's behaviour model [27] summarises the mechanisms of user action elegantly: "Behavior (B) happens when Motivation (M), Ability (A), and a Trigger (T) come together at the same moment". To increase the odds of a user performing an action we desire, all three elements must be present, and in sufficient quantity to cross a so-called "decision threshold".

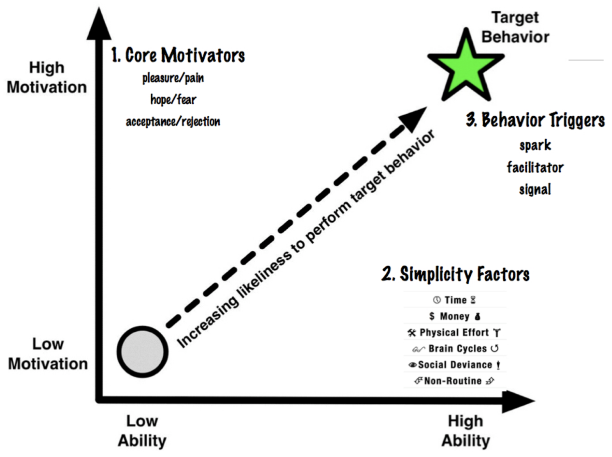


Figure 2.3: The Fogg Behaviour Model as illustrated in the original paper.

Having already covered triggers and how they may be used to spark motivation, this section mainly discusses ability. The example in the previous section is an excellent example of how interactive elements can be carefully placed to make action as easy as possible for the user. This highlights the fact that motivation and ability can trade off, as a highly motivated user may be willing to go through more work in order to achieve a desired outcome. Ease of use is much more within developers' control than user motivations and intentions. If a developer can design the interaction such that all the "simplicity factors" expended (in figure 2.1) are minimised on behalf of the user, the product feels intuitive and users can simply act on instinct rather than constantly deplete precious reserves of cognitive effort.

A quick moment of reflection will reveal that many of the features that we consider to be excellent are the ones that do precisely this. The widely adopted "Sign up with Facebook" feature completely removes the barrier of entry that new users usually have to go through to access a site. Most modern smartphones have access to their cameras without requiring an explicit unlock. E-commerce retailers often allow users to subscribe to email updates to know when a specific product is back in stock. The list goes on and is endless, but it is clear that usability is a key factor in app design, allowing users to spend increased amounts of time in a given app without feeling the strain of prolonged cognitive effort or other resources.

2.4.3 Variable Reward

We have known since the times of Pavlovian conditioning that rewards condition a learned response in animals. Science has since then come a long way in helping us understand the mechanisms responsible for motivated action in humans and animals. The neurotransmitter responsible in many of these motivating behaviours is Dopamine. Dopamine travels through various parts of the brain through well-known pathways, and each of these pathways is responsible for feelings of pleasure and reward such as during sexual intercourse and eating food (particularly foods high in sugar).

However, dopamine is not only released upon the actual experience of a positive experience, but in anticipation of one. A study by Salimpoor and colleagues [28] conducted a study on dopamine release and music. It showed that contrary to belief at the time, dopamine was released distinctly in different pathways at two moments: in anticipation of a moment of peak emotional arousal, and during the actual moment itself. One of Schultz's [29] many extensive works on dopamine-reward mechanics also builds upon long-standing models in neuroscience which state that the brain uses dopamine to learn emotional responses by comparing the anticipatory and actual response upon receiving the reward. Schultz showed that the dopamine neurons showed the greatest activity when the received reward exceeded what was anticipated, therefore resulting in the greatest learned behaviour. If a reward was anticipated but an underwhelming (or negative) result was actually received, the latter dopamine response is practically nonexistent and the brain associates this to a negative experience.

Variable rewards in apps exploit this in one of two ways: either providing a reward when one was not expected, or by having a low-calibrated expected reward and providing a much larger one (eventually). The addictiveness of this variability has been shown to be extreme. The classic experiment conducted on mice by Olds and Milner [30] connected electrodes directly to the pleasure center in test mice and placed them in a chamber where they could press a button to stimulate themselves. When the reward was made variable (given after a random number of button presses), mice lost interest not only in food and drink, but even sexual reproduction. A later study [31] confirmed with fMRI scans that humans had higher brain stimulation levels when the potential reward was variable, and not predictable.

Taking this knowledge in the context of primitive human biology, Eyal [25] describes that rewards fall into three general categories: the tribe, the hunt and the self. Simply put, humans crave rewards from social acceptance, reward-for-effort and self-accomplishment and improvement, respectively. One does not have to be a genius to see how these exist in the design of social media apps. Facebook and Instagram give users social acceptance in the form of statistics on their content, such as the number of "likes" from friends, and even have social interaction built in to features such as comments and in-app messengers. Such updates happen completely randomly from the perspective of the user, as they are actuated by other users. It is therefore very common for social media app users to feverishly open the app, even when no explicit notification has been received. Even if no direct response from a friend was expected, apps still employ machine learning algorithms to place a tailored content feed in front of the user when they open the app, providing a reward even when none was expected. Over time, this results in a conditioned response of opening the app in expectation of some unknown abstract reward, and an internal trigger is formed that tethers the mere opening of the app to a positive, rewarding experience.

2.4.4 Investment

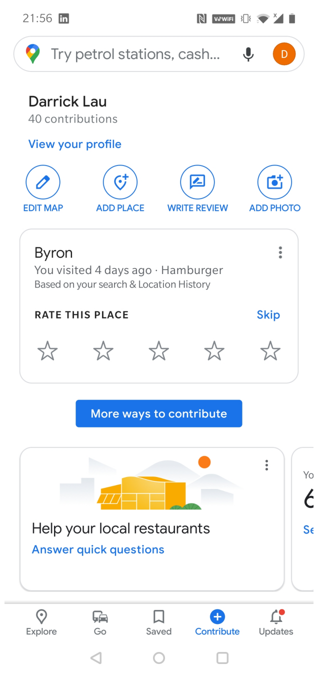
The final phase of the 4-step hook model requires the user to perform an investment in the product. This is not necessarily monetary, but rather refers to the user placing some form of effort into the system. This is slightly contrary to the principles of action mentioned before, in which effort on behalf of the user was minimised. In order to get users to willingly put in this investment, it is usually prompted after the user has received a variable reward. This, according to Eyal [25], plays on the human tendency to reciprocate to kindness. Other motivating factors for a user to contribute may be similarly built up by the app. Humans enjoy consistency with past behaviours and dislike cognitive dissonance, where one's actions may be in conflict with one's beliefs. We will see in the following example how these can be exploited.

Investment can come in many forms, and often apps find a way to make users feel a sense of ownership over their investment, even if it offered no direct benefit to them. Many of the most successful apps have some form of recommender system within them. Google Maps, for example, is essentially a large geographical recommender for a user's desires. Apart from its explicit use as a navigation tool, it allows the semantic search of locations, such as when one types "Japanese restaurant" or "hairdresser". Clearly, Google does not deploy an army of employees to log each and every establishment in the world. Google Maps is only as good as the data it collects, which comes from its users!

For those who are unfamiliar, Google Maps offers a comprehensive navigation experience to a location of your choice. If we were to use it to get to a restaurant for a date, Google would later leave us a notification that brings us to the "contribute" screen of the app, pictured in 2.4a.

This is a precise example which highlights the concepts of user investment. Having received the benefit of the wonderful navigation services Google kindly offered (for free!), the user is now given a chance to "Help your local restaurants". The request is made after having received some benefit, and offers the user a chance to not only reciprocate in kind to the app, but to perform an action that identifies them as a helpful member of the community (which, for most people, lines up with the image they would want to have of themselves).

The review process is exceedingly simple. Users may go out of their way to write an open-ended review with pictures of the establishment, or simply answer a set of pre-determined questions that Google has set, all not requiring open-ended answers (Figure 2.4b). In either case, the user is thanked with colourful confetti explosions upon completion, and the congratulatory screen is shown (Figure 2.5b).

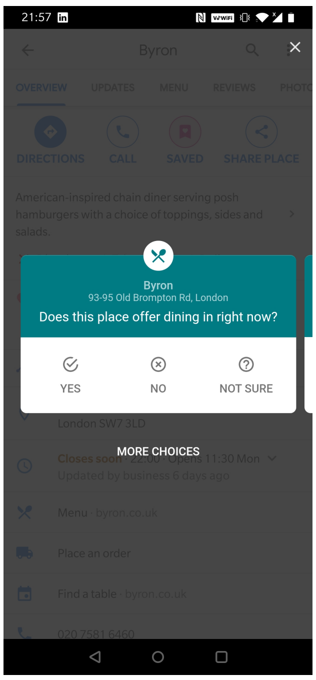


Darrick Lau
40 contributions
View your profile

Byron
You visited 4 days ago · Hamburger
Based on your search & Location History

Help your local restaurants
Answer quick questions

(a) The home page.



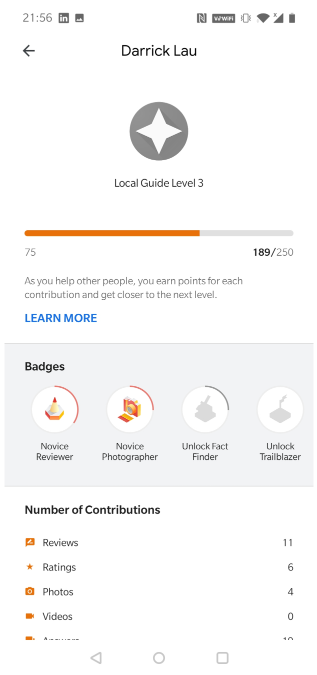
American-inspired chain diner serving posh hamburgers with a choice of toppings, sides and salads.

Byron
93-95 Old Brompton Rd, London
Does this place offer dining in right now?

London SW7 3LD
Closes soon · 22:00 · Opens 11:30 Mon
Updated by business 6 days ago
Menu · byron.co.uk
Place an order
Find a table · byron.co.uk
020 7581 6460

(b) A survey question about a given establishment.

Figure 2.4: The “Contribute” Section of the Google Maps App.

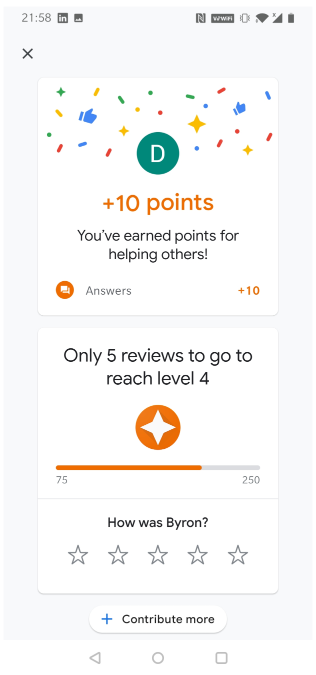


As you help other people, you earn points for each contribution and get closer to the next level.

Number of Contributions

Reviews	11
Ratings	6
Photos	4
Videos	0

(a) Google Maps contributor profile



+10 points
You've earned points for helping others!

Only 5 reviews to go to reach level 4

How was Byron?

(b) A congratulatory message upon contributing.

Figure 2.5: The gamification of user contribution.

This brings to mind another concept: ownership and gamification. Much like classic role-playing games such as World of Warcraft, the user is bestowed a sense of ownership by building up a profile of "achievements" by contributing to the app, and their efforts are celebrated with colourful badges and titles (Figure 2.5a). The user is now not only increasingly attached to the product, but actively takes pride in helping to improve it on behalf of the developers.

2.4.5 Concluding Statements on Addictive Design

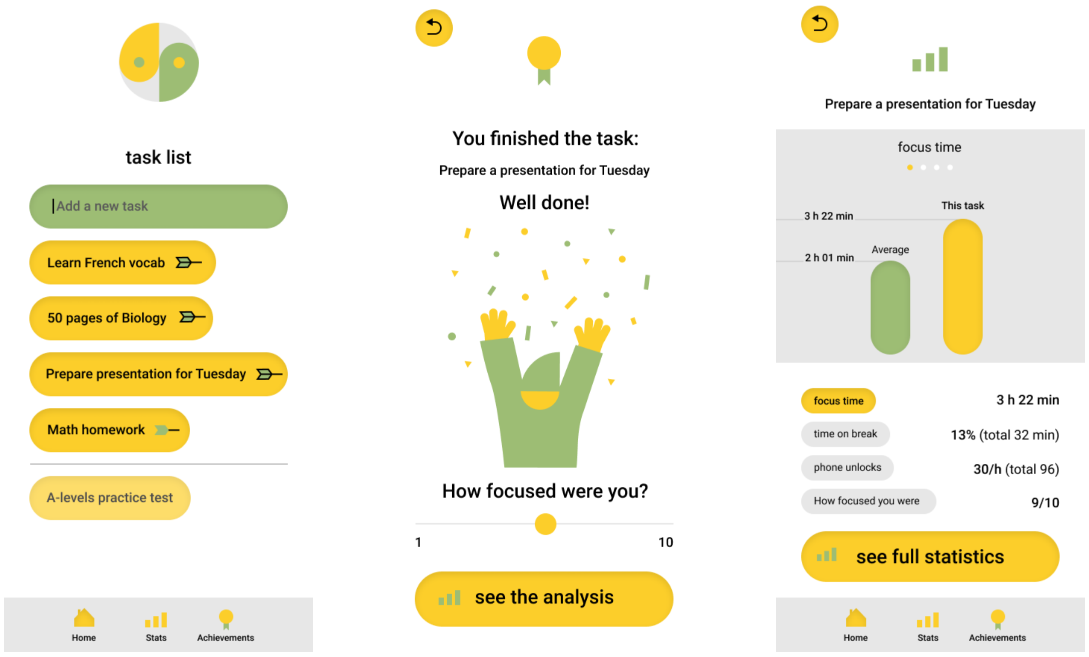
It is concerning that terms such as "habit-forming products" have been thrown around so freely and only ethically questioned in the past few years. Admittedly, none of these concepts are insidious in and of themselves. One would argue that Google Maps, despite starring in these examples, has largely been an extremely beneficial product for modern society (Not many people suffer from addiction to Google Maps). However, it is clear that larger social media apps have iterated many times upon their original product and achieved a very successful application of these principles in their software design. Large amounts of money are dedicated to User Experience research in universities and internal departments in modern companies in pursuit of furthering the art of product design.

Ethics aside, one cannot expect a corporation not to take every action possible to optimise its revenues. To date, no legislation exists to govern the principles of addictive software design, and we argue that it would be very difficult to do so even if lawmakers wanted to do something about it. The best solution in the interim would be for this information to be widely available such that users at least are made conscious of how apps on their phones may be driving their unconscious behaviours.

2.5 The Existing Sage Application

Having now briefly covered the mechanics of human attention and addictive app design, a critical inspection of the current platform is necessary in order to determine the ways in which we might go about improving it.

2.5.1 Functional overview



(a) Home page (b) Completing a task (c) Usage statistics

Figure 2.6: A few screens from the existing Sage app.

After granting the app permissions for notification access in an onboarding sequence, the app starts up and offers its core functionality: a task list. Users may create and edit tasks. Each task may be recurring daily, and may optionally block notifications. Users may then at any time start working on a task, triggering a running timer which may block all incoming notifications while active. A task can be completed, or returned to any time later with a break. All this activity is stored within the client and can be displayed to the user through graphs as a form of feedback on their productivity.

2.5.2 Limiting issues

While the legacy app passes as a starting effort, the concepts explored so far reveal many potential points for improvement. This naturally may have been the result of there only being a single developer at work. This is likely to occur again with this run of the project, and so this section serves as a potential guide for future work in identifying deficiencies in the app's user experience and building upon them. That said, there are several issues and limiting features with the current implementation of Sage, from both a UX design and software engineering perspective which we wish to prioritise in this run of the project:

Design & User Experience

- The selected colour palette is dull, with no possible customisation on behalf of the user. Furthermore, colour contrast and shadows are not employed to draw attention to interactive elements within the app, reducing the enjoyability of the experience.
- All the assets and images are static, with no animation or movement to create the aforementioned dense and enjoyable app experience.
- Sizing of interactive elements is not well-standardised, and in most cases are oversized.
- Interface interactions are often not animated, which is not only potentially very jarring for the user, but also may result in a well-known concept known as change blindness [32, 33], in which a user does not notice that an element of the interface has indeed changed. This directly translates to a higher amount of cognitive effort expenditure on the part of the user.

Software Engineering

- The notification blocking feature is faulty. The current implementation dismisses the notification using a callback function which is initiated when the notification is fired. This code is asynchronous to the actual firing of the notification, and therefore there exist edge cases in which it can fail.
- Currently implemented only as an Android app, with no extensibility to iOS, desktop or mobile web users. The features offered by native code APIs in Java and Kotlin are almost unused, and offer a lot of potential for attention-preserving features of the app.

- Completely local: no data backups or export, no online cloud synchronisation.
- No documented data model for the application, making continued development harder for newly-onboarded engineers.

2.6 Concluding Remarks

This chapter has explored the main concepts of neuroscience and user-experience design involved in app development. The existing app has been reviewed with a few starting points of critique, which is by no means exhaustive. The next step is to take these concepts back into a software engineering context and develop a set of requirements with an implementation plan for this run of the project.

Chapter 3

Project Planning

3.1 Project Respecification

This project seeks to create a user experience that leverages the concepts explored so far, such that minimal deliberate exercise of self-discipline is necessary for the user to remain focused while working. As in any other software engineering problem, the first step is to gather an initial list of requirements that will form the specification of what the final product must achieve. The re-implementation of the Sage application aims to complete with the following set of features:

Functional Requirements

- Goal and To-do list functionality.
- Focus timer functionality, implementing 25-minute work periods as per the pomodoro technique [34].
- Improve the current self-feedback system. The current statistics offered to the user are not visualised in a meaningful way over time, and some of them feel like they might not even be helpful.
- Implement notification blocking functionality with the use of *Do not Disturb* mode, instead of the current implementation.

- Offer online cloud storage, identity management and synchronisation of user data.
- Feedback survey built-in to the app and send to an online repository for evaluation.
- Offer an in-app educational experience on the mechanics of attention and addictive app design.
- Offer a consistent app experience across mobile and web, so that users do not have to be in the presence of any one particular distracting device.

Non-functional Requirements

- Must be performant without excessive jank, jitter or loading delay.
- In the event of a crash, the application should recover gracefully and inform the user of any inconsistencies in their data caused by the crash (interrupting a focus period, etc).
- The application should be fully GDPR compliant and users should have a large degree of awareness and control over their data and how it is collected.

Some further stretch goals include other features like an in-app social experience, but these ideas are still pending feasibility evaluation from a behavioural perspective and will be documented at the time of implementation, if applicable.

3.2 DevOps

An excellent starting point for planning this project is to survey the latest development practices used in industry. DevOps, as it is now known in the industry, is a set of practices dedicated to speeding up code delivery while ensuring that faulty code never reaches users, while allowing developers to iterate and experiment behind the scenes. The following sections outline some basic DevOps principles whose practices will be implemented in the project.

3.2.1 Agile Software Development

Agility is a concept often used in business strategy and operation, referring to an organisation's ability to adapt to quickly changing business demands and put themselves in the most competitive position possible. This concept has naturally spread to software development practices, and is largely referred to under the umbrella term "Agile software development".

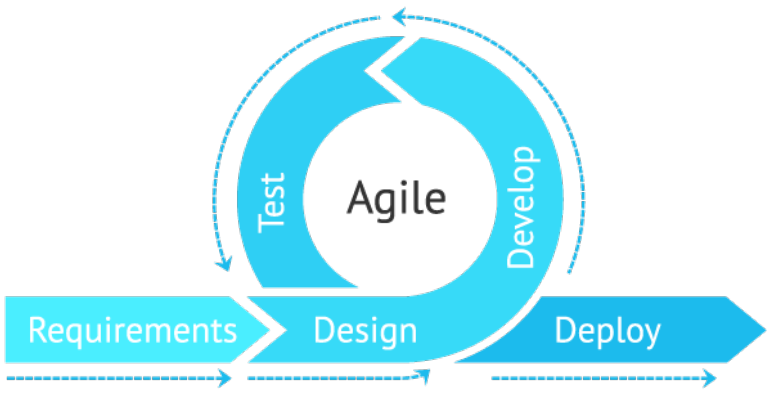


Figure 3.1: Agile software development flowchart for a single sprint.

Agile comes in many flavours and frameworks for teams to follow such as Scrum, Lean software development and Kanban. However, they all seek to solve the problems which bottom-up uni-directional software development (called "waterfall") failed to address. Agile incorporates feedback-driven revision into the development process which allows quick pivoting based on all manner of contingent circumstances such as changing requirements, fickle clients and unforeseen roadblocks.

Regardless of flavour, Agile development incorporates many improvements into the software development life cycle. The client is involved as an active member the development process and does not end up with a potentially nasty surprise at the end. Short development (called "sprints") cycles and tight, frequent deliverables reduce the effects of decision paralysis, trying to find a "perfect" implementation and excessive feature creep where developers think up of new additions as they develop, resulting in a product that is never delivered.

Within this project, a relaxed version of Agile will be implemented which caters for the limited meeting time for the team. Normal Agile development requires daily standups and feature ownership of individual developers. However, since this is a solo development project and meetings only occur weekly, a more reasonable arrangement is to set sprint goals for delivery and review in short cycles.

We therefore structure work into variable 2 or 1-week sprints. Each sprint will begin with a meeting, consisting of a progress update of work done in the previous week and review of improvement points. Important points of critique will be added to the backlog if they are actionable. A selection of tasks will then be selected from the backlog to serve as the priorities for the upcoming sprint, based on developer bandwidth for the week.

Ideally, if manpower and time constraints could be ignored, the project would require a front-end developer, a back-end and database engineer, an asset designer and a product manager in order to correctly execute the development of an app that correctly applies the principles of addictive app design. Unfortunately, the former 3 roles are squashed onto a single developer. This report will thus fully document in later sections the actions taken to account for this shortage and the compromised features that may have had to be pushed onto the backlog in favour of developing more urgent features.

3.2.2 Continuous Integration & Development

Developers often break their own code. Facebook, one of the world's largest social media giants, is known for the quality of their engineers. Yet, one of their key tenets is "Move fast, break things", for fear of causing a malfunction in code is one of the largest inhibitors of progress. Yet, system downtime and app crashes often shake customer confidence in a product. DevOps practices under the Continuous Integration and Deployment (CI/ CD) umbrella refer to practices while automate and remove human error in the deployment and release process. Such practices include automated test runners built-in to version control, and automated tasks to build and test the product before automatically deploying it and releasing it to users.

Future developer manpower on this project will consist of a fluctuating student-manned team, each member of which may carry different levels of engineering competence. Implementing unopinionated DevOps infrastructure upfront therefore allows students to focus on developing and improving the customer-facing product without worrying about breaking the project, regardless of their level of engineering competence. Later sections will explore and detail the implementation of such infrastructure.

3.3 Choosing A User-Interface Framework

A long standing problem in the development of mobile applications has been that of having to maintain multiple codebases in order to increase user reach. Especially since different platforms almost always use completely different workflows and programming languages, this means having to employ multiple teams just to maintain separate codebases, leading to features being rolled out on one platform before another, or an inconsistent user experience between both platforms.

Cross-platform development seeks to solve this problem, often by creating a workflow in a given language which may then be transpiled into native platform code. While this solves the aforementioned problem, it also creates problems of its own. Cross-platform frameworks usually offer a reduced set of features compared to their native counterparts, since they effectively have to duplicate the API that the native platform exposes using different language semantics, which may not even be possible in some cases.

Despite this, cross-platform frameworks have found their niche in helping undermanned teams of developers bring a concept to market quickly. It is for this reason that this project will use a cross-platform framework in order to bring the product to iOS in a way that is extensible for future work, especially since no feature in this app requires the complexity that native development offers. The following subsections will evaluate the current two most popular choices of cross-platform frameworks.

3.3.1 Flutter

Flutter is a recent framework released in 2017 by Google, aiming to provide a set of out-of-the-box interface elements to enable quick development of beautiful user interfaces. As of the time of writing, Flutter has full support for iOS 14, Android 11, macOS, Windows and web [35].

However, the primary criticism of Flutter is not only its age, but that it runs in Dart, another language made by Google. This means that not as much libraries and packages for flutter are limited, which means likely having to write much more first-hand code and less stable releases of libraries.

3.3.2 React Native

React Native was released by Facebook in 2014 and continues to be one of the most developed open source projects on Github (alongside Flutter). It builds off of the Model-View-Update architecture originally implemented by React and shares many domain-specific semantics. It runs on JavaScript, which is by a large margin the world's most popular programming language [36] and therefore has full access to the open-source library distribution channels that it offers. This mean stable support on well-developed and community endorsed packages that enforce a standardised implementation of many common features that developers use.

The result is a smooth cross-platform experience that offers close-to-native performance and a fully-functioning native bridge to give developers full flexibility in writing native code where necessary. The success of React Native has attracted many reputable companies to build their platforms with it, such as Walmart, UberEats, Bloomberg and Pinterest [37].

3.3.3 Comparison

The primary cause of concern regarding framework choice in this project is which one best enables speed of development. This project is scoped, designed and executed by a single engineer. The scope proposed so far is usually the work of multiple full-time software engineering teams. The age and reduced developer community size for Flutter is therefore a deal-breaker in this case. The NPM registry for JavaScript currently boasts over 1 million packages in comparison to its Flutter equivalent, pub.dev which currently has around 21000. Developer documentation and community FAQ pages are also likely to have answers to issues on JavaScript rather than Dart, which means much more time can be spent focusing on higher level application design instead of “re-inventing the wheel”.

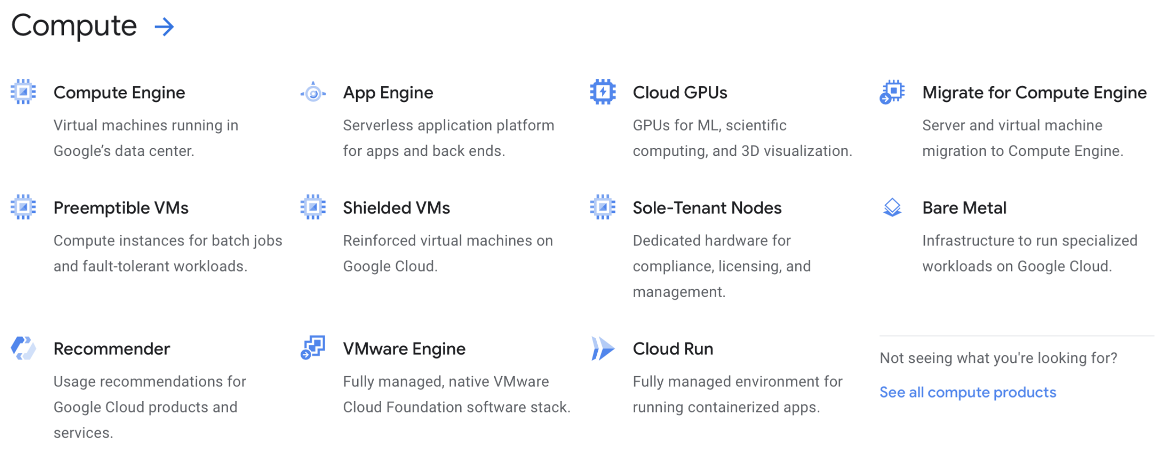
3.4 Choosing a Cloud Hosting Vendor

3.4.1 Cloud Infrastructure

Based on requirements so far, the application will also require a back-end with a database for persistent storage of user data. This, along with what will potentially be the web version of the app, will require hosting on a server. This section briefly evaluates a selection of popular cloud service vendors and how their offerings are relevant to the requirements of this project.

The offering of a Cloud-service provider can be thought of as the balance of two philosophies: “bare-metal” providers, and managed solutions. Bare-metal refers to the act of simply provisioning computing resources on a machine, the specifics of which are left up to the user to implement. The state-of-the-art nowadays would be for a developer to specify a containerised environment using a tool such as Docker or Kubernetes, effectively creating an isolated “container” which contains all the environmental requirements needed to run a desired program. A bare-metal service would then simply bill the developer based on pre-defined billing criteria such as CPU runtime, memory usage and network bandwidth consumption.

Managed solutions are built on top of the bare-metal philosophy. Cloud service providers may have specific offerings which deliver value to their customers in the form of pre-defined APIs which achieve a commonly desired functionality, such as user authentication. The market-leading solutions often provide a modular product selection that gives a developer a large degree of freedom to build suitable solution. One example of this is Google Cloud, which



Compute →

Compute Engine
Virtual machines running in Google's data center.

App Engine
Serverless application platform for apps and back ends.

Cloud GPUs
GPUs for ML, scientific computing, and 3D visualization.

Migrate for Compute Engine
Server and virtual machine migration to Compute Engine.

Preemptible VMs
Compute instances for batch jobs and fault-tolerant workloads.

Shielded VMs
Reinforced virtual machines on Google Cloud.

Sole-Tenant Nodes
Dedicated hardware for compliance, licensing, and management.

Bare Metal
Infrastructure to run specialized workloads on Google Cloud.

Recommender
Usage recommendations for Google Cloud products and services.

VMware Engine
Fully managed, native VMware Cloud Foundation software stack.

Cloud Run
Fully managed environment for running containerized apps.

Not seeing what you're looking for?
See all compute products

Figure 3.2: Google Cloud compute product offerings

offers different degrees of control within its compute offerings, ranging from fully managed to completely bare metal. Solo-developer projects thus benefit greatly from the conveniences that managed workflows offer, and only employ bare-metal products where the flexibility offered is strictly necessary. In large enterprise teams, the reverse tends to be true, as maximal control over the codebase means less problems with software agility in the long run.

3.4.2 Firebase

Firebase began as a startup that was later acquired by Google and now stands as one of the most popular backend-as-a-service platforms for developers looking to get small projects off of the ground. Firebase currently offers an unparalleled out-of-the-box offering for implementing safe user authentication, identity management, database, on-demand cloud compute and many more essential features in a single workflow.

However, all these benefits come at the cost: Firebase is closed-source and has vendor lock-in built into its design. Firebase is much more opinionated than the other back-end solutions that we have explored, in the following ways:

- Databases are accessed directly by clients instead of a back-end endpoint through the Firestore or Realtime Database APIs.

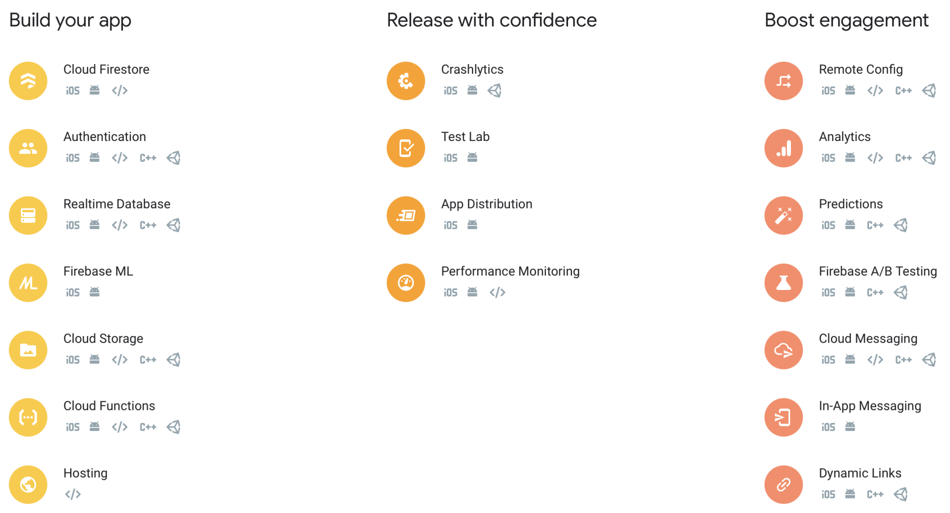


Figure 3.3: Firebase product offerings

- Firebase-provided databases are strictly non-relational (noSQL). Firebase offers no "bare-metal" products for developers to spin up their own databases using other paradigms.

- On-demand compute comes in the form of Firebase Cloud Functions, which is designed for small bursts of computation instead of long-running loops. The latter is still possible, but the resulting bill would be unfeasible.

- Cloud Functions currently only supports REST APIs and only supports a limited range of runtime environments.

Despite these, Firebase provides key functionality which may prove extremely valuable in continuing development for this project. Even if not implemented in this run of the project, Firebase offers out-of-the-box solutions for file storage, app crash analytics, search engine optimisation, A/B testing and even (limited) machine learning tasks. Therefore, it is undisputably the platform that will enable the fastest development of a workable product for this project.

In summary, the features that it provides will service the needs of future development on this project for a long time to come. Even if it should scale beyond that point, there should be enough resources to facilitate a migration away from Firebase anyway.

3.5 Evaluation Plan

Instead of the typical academic approach of conducting individual user studies on a select sample, this app hopes to build feedback in to the app as part of the user experience. The app is thus hosted on a cloud provider with a generous “free tier” allowance, allowing us to experiment with a sizeable user base of what we estimate to be up to 500 users before cloud computing costs actually start to kick in.

For this study, the concerns are not to develop the world’s best-selling productivity app or to gain traction with a massive user base. This preliminary phase of the project is to find a way to measurably determine how effectively the Sage platform does its job of helping users stay on task. The refinement, re-selection and validity of these metrics will have to be determined as part of future work when the app is released in beta to the test audience. As an example, we may wish to measure a given user’s productivity on a day-to-day basis by capturing a “productive window” in which a user continues to execute tasks without a significant (15 minute) disruption.

The app will be released to students in Imperial College London through the Android and iOS distribution channels, with its intentions clearly stated in both publicity efforts and as part of the in-app onboarding process. The goal is to complete development on the alpha release in time for the Easter holiday, when students will be focusing on studying for exams.

3.6 Ethical, Legal and Safety Plan

This project has no actionable safety concerns, since there is no physical product in which the user might harm themselves with. The primary concerns are with the use of third-party intellectual property and the safe and ethical use of user data for analytics and product improvement.

3.6.1 Software Licensing

This project will leverage the use of open-source software libraries and frameworks, the largest example of which is React Native. Open-source, by definition, allows the unrestricted redistribution and modification of a given piece of software for any purpose. As a secondary precaution, all packages used will be derived from the Node Package Manager (NPM) registry, which effectively serves as a central distribution point of open source packages pertaining to the JavaScript ecosystem (in which this project resides).

Most, if not all packages registered on NPM have an open-source approved license selected for their project. That is, a license which is in accordance with the open-source philosophy of freely useable, modifiable and shareable code. The list can be found at the open source initiative organisational website. Should the exceptional case arise in which a package is used which does not use one of these licenses, it will be explicitly justified and disclosed in its implementation documentation. However, this should technically never arise.

3.6.2 User Data Legal Compliance

This study will use the General Data Protection Regulation (GDPR) as a basis of rules to follow in the ethical collection and use of user data. The following subsections will outline the parts of the GDPR deemed relevant to the nature of data used in this study, and perform a risk assessment of where grey areas may occur.

In the interest of brevity, the GDPR checklist can be found here. In addition to having completed this checklist, the Sage application will adopt the following policies for the collection of user data:

- Once implemented, users will be explicitly informed via a transparent and detailed privacy policy what data we collect for evaluation metrics and how we will use it. The usage of the app requires the user to consent to this, as there is otherwise no point in offering the service which could otherwise incur cloud computing costs.

- The application does not capture any data defined by the GDPR as “personal”, except for the user’s email. Evaluation metrics, while potentially changing over the course of the project, should not allow the de-anonymisation of a given user.
- Data will be sent and stored using secure channels, using third-party GDPR compliant vendors where applicable.
- In the unlikely event of a data leak, users will be immediately informed via email what was leaked. However, none of this data should be particularly harmful to a user even if this should happen.
- Users may at any time erase all their usage data within the settings menu.

3.7 Concluding Remarks

A full implementation plan has been developed. However, the primary cause concern for this project is the shortage of developer manpower. Successful social media apps are usually developed across large enterprise teams, each of which “owns” a specific part of the product. Even then, a polished product typically emerges only after many cycles of product iteration. A realistic goal for this project is therefore to implement an excellent product to the best of our ability and have infrastructure in place to guide future development and iteration.

Chapter 4

Implementation

4.1 Foreword

This section serves as engineering documentation for the project. All data models in the codebase are thoroughly decorated with in-editor documentation using `JsDoc`. React's tree structure also naturally guides new contributors through the app's component hierarchy, and thus this chapter focuses on rationalising the design and infrastructure choices made throughout the project. A working familiarity with a few technologies is implicitly assumed:

- Source and version control (Github)
- Front-end web development: HTML, CSS and JavaScript
- Back-ends, databases and remote cloud services

For the sake of brevity, sections will contain links to documentation pages of relevant libraries. Since they are not academic references, documentation of relevant libraries will be referenced using footnotes or in-text links, which further facilitate easy, immediate reference.

The goal is to create a high-quality app platform that is built for legacy and continued work. The first step is therefore to implement previously non-existent development infrastructure, especially since the online nature of the new app involves secret keys which can be monetarily exploited if compromised. The previous version of the app is henceforth referred to as Sage, and the new app is referred to by its new name, Within.

4.2 DevOps Implementations

The following sections outline the focal points of the infrastructure and tooling choices made for Within from a DevOps engineer's perspective.

4.2.1 Developer Workflow

The code is currently hosted open-source on Github. Source control systems often are the first line of defence in code deployment and Github offers features to allow developers to implement secure rules on their repositories. The rules for the Within project are listed below, and will be elaborated in further sections.

- Secret keys are stored in non-readable repository secrets.
- Non-collaborators cannot push changes directly to the repository.
- Changes cannot be pushed to the `main` branch without all tests passing and an approving review from an approved author, who cannot be the one who submitted the pull request.
- The build on the `main` branch is automatically deployed as the latest release whenever a change is effected.

In order to correctly maximise the utility of these features, a developer workflow is outlined in the project `README`. Github unfortunately does not offer strict branch protection like other version control systems, and thus developers must adhere to these guidelines, as they are not strictly enforced by the system. Invited collaborators may clone the repository directly and develop features on a separate branch, which will be merged into a staging branch (`development-staging`) before being approved to `main`. Non-collaborators who wish to contribute may fork the repository and submit a pull request from their fork, as per open-source norms.

4.2.2 Continuous Integration / Continuous Deployment Infrastructure

Automated CI/CD tasks are run by commissioning remote machines to run scripted actions (often called "jobs") for a fee whenever the codebase is updated. For this reason, many CI/CD solutions are offered as add-ons or plugins to popular version control systems, so that they may be run as post-commit hooks. All jobs in this project are implemented with Github Actions. Each action is implemented in YAML syntax under the `.github/workflows` folder in the root of the project repository. Github actions provide unlimited compute minutes to open-source repositories and is very well-integrated into the Git ecosystem, making it the superior choice. Currently, the jobs take a total of about 20 minutes on each code change, which would easily exceed the quota for a private repository. If the project is to go closed-source in the future, it may be of interest to change to an alternate, paid solution such as Travis CI or CircleCI.

4.2.3 Regression Testing and Runtime Stability

A key tenet of the software development lifecycle is to test code before deploying it. To this end, this project uses the open-source Jest testing library by Facebook. Jest implements a configurable test suite which recursively searches an entire JavaScript project directory for any files ending in the `.test.js` extension, and provides a declarative syntax for writing and running tests. Developers can write tests right alongside the code that it tests. Code regression is henceforth detected by running a single command, and Jest displays a full breakdown of any test that failed as a result of recent code changes. The test suite is automatically run with every commit on the `main` and `development-staging` branches of the repository, and will cause the jobs running them to produce an error whenever the test suite fails.

JavaScript as a language is dynamically typed and thus highly prone to runtime errors. This motivates the use of TypeScript and immutable data. We will later see examples of how using strongly type-checked immutable data structures to track the state of the app greatly reduces the occurence of runtime errors that would have otherwise have been very difficult to detect.

4.2.4 Build Stability

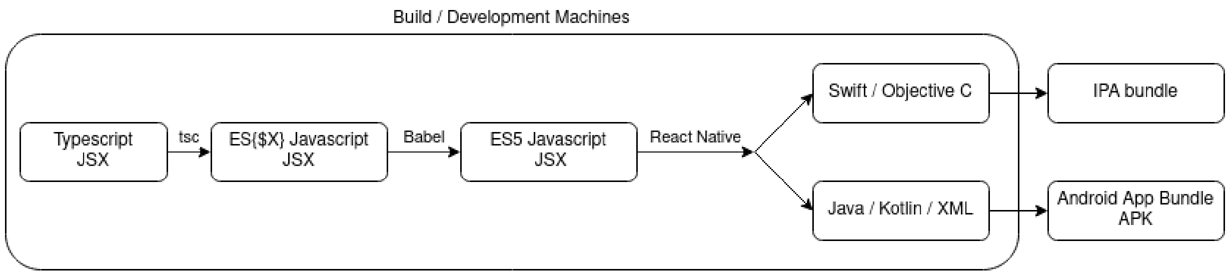


Figure 4.1: An example of the application build process.

React Native transpiles a React-syntaxed JavaScript codebase into an Android Kotlin project or an iOS project in Objective C. Figure 4.1 shows the build process of the toolchain involved in this application. Each step in the build process in this case involves a source-to-source compiler, which has to bridge semantic equivalents and substitutes between languages and frameworks. Having a long toolchain of different open-source components results in an unreliable build process which can fail if any dependencies along the toolchain fail for any reason.

True to form, throughout the development of this project, there were many instances in which the project would suddenly fail to build without any change in the codebase. Most frustratingly, the project involves writing platform-specific code, which adds another dimension along which a developer could cause the project build to fail. The long toolchain involved in the project build results in enormously long stack traces, which result in a situations where a developer cannot possibly know if the project is now failing to build as a result of their own changes, or a dependency breaking along the chain. Almost every time, the cause of build failure was a version dependency change in one very small library along the toolchain.

A naive solution would be to require the developer to run a script which builds app bundles for both platforms before pushing code to source control. However, building app bundles often takes a few minutes for each platform, which is extremely disruptive when developing small, rapid changes.

To address this issue, two automated jobs are defined in `android.yml` and `ios.yml`. The two jobs build the installation binaries for both native operating systems on cloud-connected build machines and throw an error if any of them fail, storing full trace logs in the job history on Github. Building the app requires the firebase configuration file, which contains secret keys and are not included in the source code. The firebase configuration files are first compressed into a tar archive, passphrase-encrypted with GPG and then uploaded to the repository root as `services.tar.gpg`. The passphrase is then kept as a repository secret, and loaded into an environment variable to decrypt and extract the configuration files on the build machine.

Developers can sign up for email-updates via the Github repository, which allows them to be immediately notified of a build failure. They can then immediately work on understanding if the failure was due to the most recent change, or an unexpected change in the React Native dependency chain.

4.2.5 Product Deployment

Presently, the app is not deployed on any public stores due to the release process requiring a significant investment which is outside the scope of the project. Releasing the app on either store requires a monetary investment for a developer account (USD$125), a privacy policy hosted on an owned domain and a collection of assets for a store page listing. As a proof-of-concept and temporary workaround, the job implemented in `releasedraft.yml` automatically builds a release binary of the app whenever a change is successfully pushed to the `main` branch.

Building a release APK requires an app to be signed using a consistent set of keys as proof-of-work from the developer. These keys are contained in a keystore, which is encoded as a base64 string and then stored as a repository secret. The decryption passphrase is similarly stored as a repository secret, and both are loaded into environment variables for decryption on the build machine. Once the app is built and signed, the resulting build artefact (the release APK) is released on the repository with a new tag. In the future, this should be updated to use the API of the respective app stores to automate the release process.

4.3 Developer Documentation

Instead of exhaustively documenting screen flow which is best left to a live demo, the documentation in the sections to follow will detail the implemented features of the app in an order which is conducive to onboarding a new developer. New developers should adhere strictly to the contributing guidelines in the Github repository README. The documentation contained herein is by no means exhaustive, and will not fill in the gaps of knowledge for a new developer attempting to understand the repository. All linked documentation therefore serves as a starting point for further reading, and unfamiliar terms should be researched and understood before proceeding, if encountered at any point.

4.3.1 Background Information

A brief tour of React

React.JS is a library which introduces a declarative syntax for making responsive user interfaces on the web. Readers familiar with web technologies will be familiar with the Document Object Model (DOM), which simply describes the layout of a webpage in a tree-like structure. React allows a developer to declare 'what' should appear on screen, which is 'rendered' as a virtual DOM. This virtual DOM is then compared with what is presently on the user's screen (the actual DOM) using a process called reconciliation, in which only the differing elements between the two trees are identified and re-rendered. This results in minimal computation to achieve the desired change in the user interface without requiring a page refresh on behalf of the user, and the result is a single-page application (SPA), which continues to be the dominant design for parts of many modern websites.

The paradigm of 'thinking in React' has now become so popular that React is a dominant presence in modern web stacks, and a highly in-demand skill for developers. Reactive paradigms have been further extended to new libraries such as React Native, which has an almost identical developer experience to React.JS, but transpiles down into native code for multiple platforms. React Native reads and largely feels like working in React.JS, but introduces a set of components which act as wrappers around native APIs to produce a codebase which has one consistent syntax which renders an app across multiple devices, and enables business logic to be entirely contained in JavaScript. There are naturally exceptions to this, such as when some APIs are only avaiable on one native OS and not the other, as we will see later.

As mentioned before, Within also makes extensive use of TypeScript (TS), which greatly increases code clarity and reduces runtime errors using compile-time type-checks, at the cost of writing slightly more upfront code. Since TypeScript is a typed superset of JavaScript (JS), this means that all valid JS code is valid TS code, and the result is a natural learning curve which a given develop can lean into as much as they prefer.

The root React node is contained in `App.tsx`, which contains other setup code for libraries which require initialisation outside of the standard React component lifecycle. Before proceeding, developers should read the documentation links below, as these concepts are heavily used in this repository. At minimum, a thourough understanding of the React component life cycle, as well as the `useEffect` post-render hook are required to understand the sections to come.

- React Docs - the starting point for any React developer. Read all main concepts.
- A guide to React's rendering behaviour
- React Context documentation. Context is heavily used to pass data downwards through the component tree without having to bloat an entire chain of component props.
- React Hooks documentation. Hooks in React are a concise way of using component lifecycle methods and other React utilities without having to create a verbose class-based component.
- TypeScript Docs

App Navigation

Navigation loosely refers to the practice of directing the user to different parts of the app using a visual experience. All navigation is implemented with the React-navigation[1] library, which implements conditionally rendered fullscreen pages based on named routes. All routes are contained and enumerated in `navConstants.ts`.

Global App State

Within is a write-heavy application, with state that changes on every interaction. Such state needs to be accessible in different parts of the component tree, motivating the use of a single, global state provided by context at the root of the tree. However, global states require careful management and needs to be mutated in a stable manner to prevent crashing the app. All of these problems are solved by the Redux paradigm, which promotes the use of a single, immutable state contained in a plain, serializable JavaScript object.

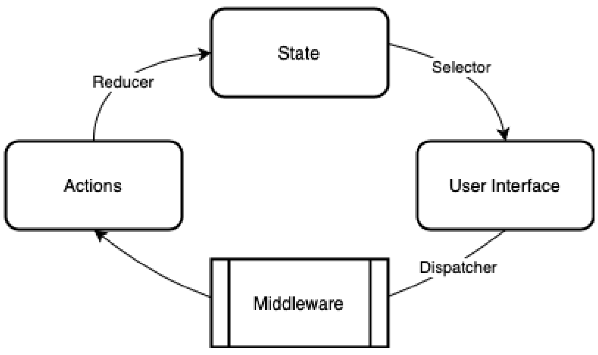


Figure 4.2: The state-component lifecycle with Redux

Redux[2] is a state-management paradigm which generates new states based only on the current state and a descriptive action, the processing of which is handled by a pure function that contains no side-effects (called a reducer). App state is purely contained within a serializable JavaScript plain object, which means the state will not contain any moving or self-mutating parts. The resulting state is always predictable and thus easy to debug. Since states simply

[1]React Navigation documentation
[2]Redux documentation

transform from one to another in a sequence instead of mutating, state-recording debugging methods such as time-travel debugging, crash logging and an undo history are easily implementable.

Since its inception, Redux has developed a rich ecosystem, which has unfortunately resulted in unstandardised documentation and practices. The current best practice for using Redux is to use Redux-toolkit[3] along with library-specific bindings, and thus Within implements global state using Redux-toolkit and React-redux[4].

Figure 4.2 shows the data flow for Within using Redux. Relevant parts of the state are read from the store into React components using selectors. Interactive parts of the user interface dispatch actions, which may or may not be intercepted by middleware to execute side-effects before passing the action on to the reducers, which generate a new state based on the old state and the dispatched action.

The React-redux library allows us to provide a global store through context at the root level of the app. The reference of the store object itself does not change when actions are dispatched, and components down the tree use selectors to select specific parts of the global state to "listen" to. This results in a performant user interface which only re-renders the relevant parts of the component tree when relevant bits of the app state change, instead of the entire tree. Similarly, the dispatch function which is used to dispatch actions for the store is also provided globally at the root using context. It is noteworthy at this point that both `useAppDispatch` and `useAppSelector` are pre-typed wrappers[5] around their base counterparts from the Redux library.

The global state in this case is also split into segmented parts using reducer-splitting, in which each part of the app state is handled by a sub-reducer. All redux and state-related implementations are contained in the `redux` folder, and the `rootReducer` combines the subreducers using ES6 object property-value shorthand. All selectors can be found in the `selectors.ts` file.

[3] Redux toolkit documentation
[4] React-redux documentation
[5] Redux usage with TypeScript

4.3.2 Authentication and Identity Management

Data synchronisation is achieved through a user's unique ID, which is unified across all their logins. The secure login is implemented through the Firebase SDK using the React-native-firebase bindings[6], which provides a single point of unified identity management using either email credentials or one of the supported providers. Users are able to access their data and have the same experience across devices after logging in.

While Firebase offers support for third-party login providers, these are unfortunately unavailable in the scope of this project due to the lack of presence on an app store. Apple devices require any app offering provider sign-ins to offer AppleID sign-ins first, which require a verified developer account and a published app on the App Store for security reasons. However, a traditional email and password style sign-in is fully implemented.

In Within, the user's login status is held in the `appSettings`-related reducers. A single listener in the `StackScreens.tsx` component dispatches an action whenever the user's Firebase login state changes.

4.3.3 Structuring Productivity in Data

Motivation

As discussed previously, a key factor in successful self-regulation is the amount of cognitive effort required to successfully execute a task. Therefore, procrastination is best avoided if users are able to consistently plan their work schedules into small, manageable tasks which can be completed in a single 25-minute focus interval. In Sage, this was entirely the user's responsibility as only to-dos could be created without any form of structure other than daily recurring tasks. Within seeks to introduce a more deliberate structure that emulates real project work, with a more longitudinal time scope and time-sensitive deliverables in-between.

[6] React-native-firebase documentation

Initial Data Model

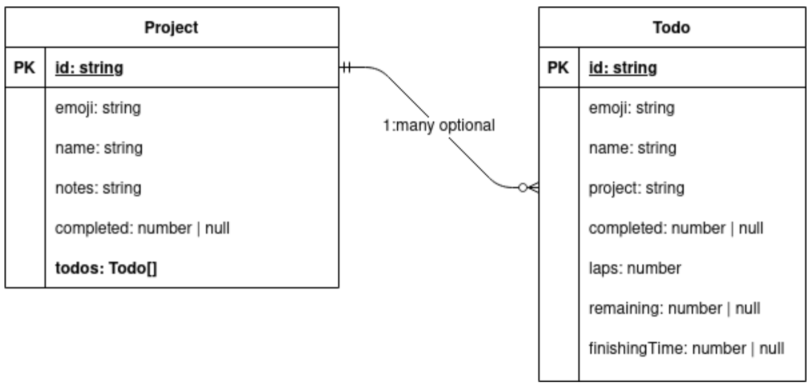


Figure 4.3: The initial data model.

Figure 4.3 shows the Entity-relationship (ER) model to support projects and to-dos. This schema is easy to implement using foreign-key relations in an SQL-like database. However, noSQL stores such as Firestore often use composition to capture relationships. This is reflected in the fact that a Project contains an array of Todos, which are serialised to raw string values for storage in the database. In an SQL environment, this would be captured purely using a foreign-key relation between Projects and Todos. This replication (signified in **bold**) has been left intact, since it does not change any of the factors affecting pricing by any significant amount, and is useful in helping developers understand the relation between the two entities, should the system migrate to an SQL datastore later on.

Revised Data Model

Projects needed to be broken down further into multiple deadlines, and a given deadline in focus had to be further decomposed into individual tasks on a per-session basis. Figure 4.4 shows the ER model for the updated schema. This model allows for a structured breakdown of complex tasks, modelled after a typical student or academic's workload.

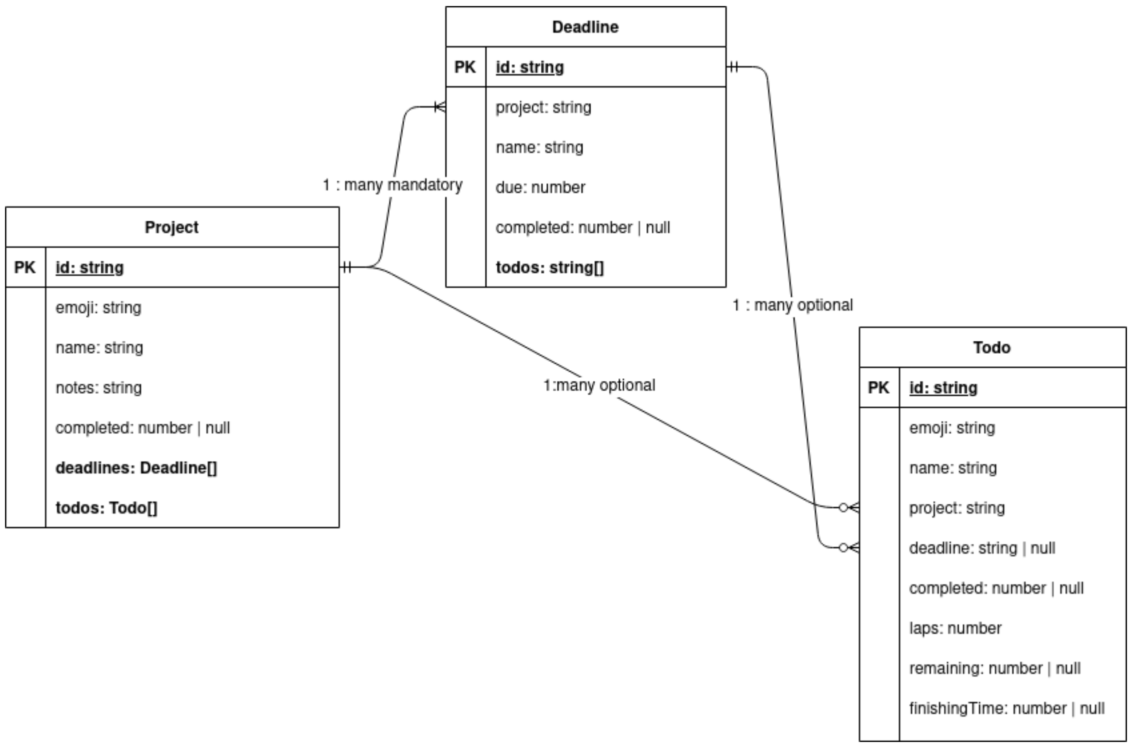


Figure 4.4: The current data model.

Syncing through snapshot listeners

Firebase bills developers off of three key metrics: reads, writes and network eggress[7]. JSON-like objects are stored as documents in named collections, and documents may themselves hold subcollections. Therefore, an obvious was to optimise the schema for minimal reads is to hold more information within a single document. Reads and writes are optimised by serialising small objects into `string` representations, which can then be kept in arrays under larger entities. In Firebase, all deadlines and todos are held under a single project's document. The user's work data is therefore encapsulated in an array of `Projects`. Changes to any entity (Todo, Deadline or Project) also consequently only cost 1 document write under this schema.

Firebase offers snapshot listener functions as part of its JavaScript API. These follow a publish / subscribe (pub/sub) model where a subscription function is passed a callback to execute. Real-time synchronisation is achieved by attaching a function to a listener in `StackScreens.tsx` which dispatches an action to load the updated app state into whenever the database is updated,

[7]Firebase quotas documentation

in a process called "hydration". The app state stored in the database is taken to be the single source of truth, and actions therefore cause changes in the app by writing directly to the database, triggering a hydration. This, however, is an asynchronous action which happens over a network. Since Redux reducers are meant to be pure functions without side-effects, there is a need to "intercept" the action and perform the database write as an asynchronous side-effect before dispatching the action on to the store. The Redux-thunk middleware[8] is built for this purpose. All database-sensitive actions are therefore implemented as thunks in files called `thunks.ts`, which are categorised by their respective slice reducers under the `redux` folder.

4.3.4 Do-not-disturb mode

Preventing notifications from disturbing a user is one of the key functions of the app. The native Android SDK gives a developer the ability to invoke a callback function whenever a notification is fired by the system. The previous notification-blocking functionality thus worked by adding code within this callback to dismiss the notification as soon as it was fired. This implementation has 2 fatal flaws: firstly, the asynchronous nature of the callback invocation meant that there would be non-deterministic edge cases in which running tasks on the main thread would delay the dismissal of the notification, therefore firing too late to dismiss it and allowing the notification through. Secondly, the dismissal of a notification meant that a user would not see them after the work interval is complete, potentially causing the user to miss important messages or calls.

A more suitable implementation that fixes both of these faults lies in the Do-not-disturb (DnD) functionality provided by the Android OS. DnD mode suppresses notifications while active, preventing attention-grabbing features such as sound, vibration and the notification LED on the phone face. This has obvious applications for users who are driving, in important meetings, or otherwise wishing to be undisturbed. All notifications received in this mode are still visible to a user who deliberately picks up the phone and checks the notification inbox after the fact, which means users still have access to any important pieces of information delivered during the

[8] Redux thunk documentation

interval.

The challenge now is accessing native android APIs from within React Native. As Figure 4.1 showed previously, the entire app is written in TypeScript. However, React Native offers a `NativeModules` feature which allows developers to write native code modules which may be accessed across the language gap. Full documentation is available in the React Native guide. From here, turning on Do-not-disturb mode programmatically is a matter of navigating the Android APIs. The full code is available in `DnDMode.java`.

4.3.5 Disruption Detection

Even with notifications disabled, users are likely to be distracted when faced with roadblocks or difficult tasks, and may flee the app to seek momentary relief on social media or other distracting apps. A solution to this would be if the Within app could detect when such distractions occur, and prompt the user's pre-frontal cortex to wrest back control.

Apps on phones typically adopt one of three states: they are either foregrounded, backgrounded or inactive. React Native provides nice utilities around these states [9], allowing the developer to attach callback functions when such state transitions are triggered.

The relevant listener is subscribed in `TodoTimer.tsx`. The relevant code attaches an event handler in a post-render hook using `useEffect`. Since the timer component re-renders whenever the selected task changes, it is necessary to re-subscribe the listener whenever the timer re-renders, and the selected todo is therefore included in the dependency array.

The listener uses the `react-native-push-notification` library to push a notification to the user whenever the app takes on a non-active state while the timer is running. For this to be fully effective, however, users must manually allow the Within app to push notifications through do-not-disturb mode, which would be active when the timer is running. There is no way for the developer to enable this programmatically, and thus the user has to be clearly guided through this process as part of a future onboarding flow.

[9]React Native AppState documentation available here

4.4 User Interface Design

Instead of aiming to develop a fancy user interface full of bells and whistles, Within aims to re-implement a simple UI with minimal clutter and good foundations in interactivity and visualisation. In this section, we will explore how Within incorporates concepts from interactivity experts such as Donald Norman and Niklas Elmqvist into its UI. Since this is not a study in human-computer interaction, we will simply make an attempt to implement the UI in a way which conforms with the guidelines explored below without studying them in-depth.

A fluid interface, as defined by Elmqvist [38], is an interface which promotes flow [9], supports direct manipulation[39] and minimises both of Norman's [40] gulfs of action.

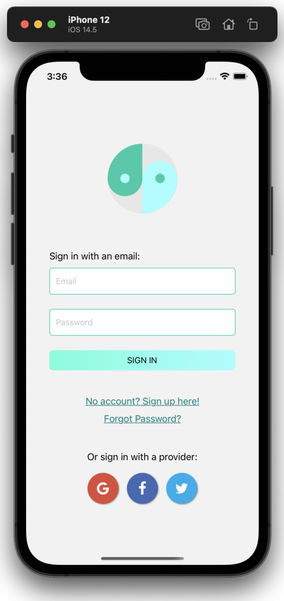
Gulfs of Action Donald Norman defines two 'gulfs' of action: the gulf of evaluation, which refers to the difference between the system state and the user's perception of it, and the gulf of execution, which refers to the difference between what the user perceives they are able to do and what the system permits. A large gap in either of these gulfs results in confusion or frustration for the user, as more cognitive resources are required for them to reconcile the differences between the information presented to them and what is actually happening in the system.

Flow The concept of flow has been highly applied in many user-interaction systems, and has become a key principle of addictive design. Yet, flow itself is a very abstract concept which has no formal definition, but refers generally to the act of a user staying highly engaged and focused on a given activity. Csikszentmihalyi [9] states a few factors for flow which Elmqvist [38] highlights as particularly relevant for the creation of fluid interfaces:

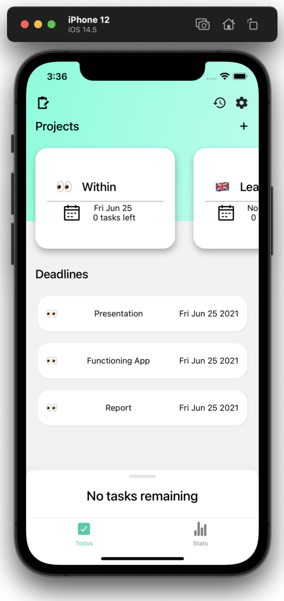
- Balanced challenge - The skill required to actuate an intention and the user's skill should be matched. In modern app design, as we have reviewed previously, often the goal is to completely eliminate any challenge required to actuate an intention.

- Concentration - Structuring the app experience in a way that permits a high degree of focus on a limited field of attention.
- Transformation of time - Users should be able to immerse in the app experience and disregard external stimuli, essentially losing track of time.
- Prompt feedback and sense of control - Instant feedback of the effect of a user's action minimises the gulfs of action, giving the user a sense of control and resulting in what many users will simply feed back as an "intuitive feel".

These concepts are difficult to measure quantitatively and are circularly dependent, but can be intuitively felt in many of today's modern products: consider a slot machine, social media app or modern video game. The challenge in this case would be to reproduce such an experience in an app which takes on a much more boring and less stimulating context (doing work instead of engaging with friends or simulated challenges).



(a) The Within login screen



(b) The Within home screen

A few noteworthy ways in which the implemented UI conforms to these principles are listed below:

- The app focuses itself around a primary color, using slight variations to create contrast to draw user attention. This minimises the gulfs of action, and automates the focus of the user's attention to relevant components.

- White space and margins are used to divide screen real-estate into semantically disjoint groups of information, which provides different fields of focus for different user intentions.
- Interactive elements will "hover" above the screen using drop shadows to convey the concept of elevation[10]. Elevated components are intuitively "pressable", minimising the gulf of execution.
- Interactive elements are interfaced with either a short touch, long press or swipe. These conform to many smartphone users' learned behaviours from other popular apps, removing any form of challenge in actuating an intention. Additionally, the most natural and frequent intentions (as determined by user journeys) are actuated with the simplest action (a short press), and require the fewest actions to execute.
- All interactive elements provide visual and animated feedback, using a transient opacity or highlight. This provides instant feedback and minimises the gulf of evaluation.

The resulting user interface is described by early testers as "clean and minimal", providing a much more compact user experience in which users can perform most of the app's base functionality without switching off of the main screen.

4.5 Limitations and Issues

The mechanisms of the Within project have been clearly documented. However, many parts of the application remain untested. Due to the limitations of solo development, testing was only strictly enforced on business logic and database sensitive operations. Ideally, snapshot tests would be implemented to ensure a consistent user interface and experience across different devices. A few known issues are listed below:

[10] Material UI Documentation

- Many components in the UI have their dimensions determined as a fraction of the viewport height and width. Some UI components may therefore appear odd on very squarish devices as a result.
- Some React components encapsulate a lot of business logic, which may benefit from refactoring.
- No easy mechanism is currently in place to easily switch off of Firebase APIs. Refactoring the reducers to fetch data from different sources using a status flag would be much cleaner.
- No workaround for the lack of notification blocking functionality on iOS is currently implemented.
- Visualisations are not implemented as an active user base has not yet been achieved.

Despite these issues, the app now offers a usable base product and is ready for evaluation and iterative development, as we will explore in the next chapter.

Chapter 5

Evaluation

5.1 Summary of Achievements

5.1.1 Engineering Milestones

Feature	Sage	Within
Multi-platform code		✓
Global, immutable state		✓
Regression testing		✓
Automated test runner		✓
Automated app signing & release		✓

Table 5.1: Engineering / DevOps Feature Comparison

Table 5.1 summarises the infrastructure and DevOps improvements made throughout this run of the project. Within implements many automated processes which were not present in Sage, and has a fully automated push-to-release CI/CD process, which eliminates possibility for the compromise of secure keys and incorrect version uploads. The improved infrastructure allows developers to solely focus on creating new features on the app.

Feature	Sage	Within (Android)	Within (iOS)
Todo list	✓	✓	✓
Pomodoro timer	✓	✓	✓
Daily task scheduling	✓	✓	✓
Visualisations	✓	✓	✓
Notification blocking	?	✓	
Projects and deadlines		✓	✓
Productivity history		✓	✓
Real-time multi-device synchronisation		✓	✓
Cloud-synced data		✓	✓
Distraction deterrence: push notifications		✓	✓
User interaction tracking features		✓	✓
In-app feedback gathering		✓	✓

Table 5.2: App Feature Comparison

5.1.2 Feature Offerings

Within offers a much richer set of features over Sage, as reflected in Table 5.2. A structured productivity framework has been added, which is fully synchronised in real-time to an online datastore. Additional functionalities have been added to the productivity timer while helps the user stay focused above possible distractions. Completion of tasks is also timestamped in the database, which can be used to produce analytics on user productivity habits which may drive future feature development.

5.2 User Feedback

5.2.1 Deployment and Release

An original goal of this run of the project was to deploy the application on the Google Play store for Android and the App store for Apple devices. However, a number of unforeseen circumstances have surfaced to prevent this:

- Each app store requires the purchase of a developer account. In total, this costs $125

USD.

- Each app store requires graphic assets for store listing pages, which require the intervention of a designer.
- Release on the app store requires a long review process, especially for apps which request sensitive permissions (such as this one). A privacy policy must be accessible from a verified-owned domain.
- Increased app store review times at the time of intended submission meant that the app could not be live before the project deadline.

While all the above criteria are technically possible to fulfill with temporary placeholders, the release process involves several rounds of internal testing to invite-exclusive testers. Due to project and time constraints, this internal testing can be manually executed through the automated Github release, since feedback has been built-in as part of the app interface. This allows us to properly prepare the relevant resources for an app store release over the summer, in time for the next run of the project to resume development.

Modern app users have extremely high expectations of apps. Slight amounts of jitter or other frustration through the user experience can be a death sentence, causing a user to uninstall an app. However, iterative development is the only way of producing an app of passable quality. This results in a chicken-and-egg problem, which is solved using incremental releases. The Google Play ecosystem supports this out-of-the-box, encouraging developers to use their internal frameworks which release the app to a gradually increasing audience as follows:

- Internal Testing: The app is first released to an invite-only email list of up to 100 internal testers, chosen by the developer. Feedback must be manually aggregated.
- Closed Testing: The app is released via the Google Play Store to an invite-only group of testers. Feedback can be gathered using the Google Play Store, but is only visible to the developer. Support for multiple testing tracks is available, allowing the developer to push a different version of the app to different test groups.

- Open Testing: Similar to closed testing, but available to any member of the public who signs up via the Play Store. Also supports multiple additional features for pre-release. Open testing often happens in parallel with a production build, where upcoming features are offered to existing users through the open testing initiative.

5.2.2 Qualitative Results

Without access to either app store, conducting internal testing manually is the only option available within this run of the project. 20 Android users were chosen to test the app. Some examples of the initially reported problems were as follows:

- Onboarding screens were not rendering consistently across different Android devices.
- Some users reported uninformed crashes or incorrect dialogs during the sign-in process.
- Some interactive elements did not work consistently on some users' devices.
- The focus timer itself did not offer enough incentive to stay within the app. Users still were distracted and went to use other apps without receiving a push notification "trigger".

This highlights a problem with cross-platform development. Many phone manufacturers develop their own flavour of the Android operating system which uses the underlying open-source Android kernel, but introduces slight differences which are difficult for developers to cater for. As such, features which may perform as expected on an emulator or a given user's phone may misbehave on another. The use of libraries in React Native adds a further level of abstraction which may contain platform and version-specific code, and therefore the reasons why a given bug may exist can be very difficult to track. Fortunately, due to the in-app feedback infrastructure and cloud-sync functionalities of the app, it is possible to add crash reporting functionality to the app which logs the user's device information along with the crash logs.

After a few initial rounds of feedback and bug-fixing, users reported a pleasant app experience and reported that while they enjoyed the app and its concept, they would like to see more features, which will be documented in the future works section.

Chapter 6

Conclusion

6.1 Summary of Work Done

A large amount of literature has been reviewed in pursuit of understanding the mechanics of human attention and addictive app design. A plan for implementation was laid forth and accomplished, with a flexible set of future requirements catering to the nonlinear progression of solo-software development. A minimal app has been developed and positive reviews from initial testers shows promise in the efficacy of attention-preserving features. The app currently offers a fully cloud-synced structured system for planning and breaking down tasks, with full notification-blocking functionality.

6.2 Applications

Internet Addictive Disorders (as generally known by the public) as a term lacks standardisation and is not yet recognised as a disorder in the Diagnostic and Statistical Manual of Mental Disorders (DSM-5). The DSM-5 does, however include Internet Gaming Disorder. The application, if successful, could therefore see use as a tool in rehabilitation of individuals suffering from chronic Internet Gaming Disorders, while simultaneously bringing awareness to the more comprehensive set of problem-causing technologies whose impacts may one day be formalised as Internet Addictive Disorders.

Corporations and individuals are increasingly seeking ways to reduce distraction from productivity, and so this app could see commercial or retail flavours to cater to those audiences. Once a good privacy policy is in place, informed users can consent to contributing usage analytics which may be helpful for ongoing research in neuroscience and human-computer interaction related to Internet Addictive Disorders.

6.3 Future work

Due to many unforeseen deployment issues, much of the effort in this run of the project was dedicated to building a codebase for legacy. Much of the tedium and complexity of building a production-quality app has been automated as part of the project's infrastructure. As such, there is room to add many features on to the existing app, without requiring more than a junior developer's level of expertise.

Creating a privacy policy and releasing the app on public stores. Since one of the ways the app can help the user is by logging usage data, a detailed privacy policy is necessary to provide full transparency to users about how their data is used. While no malice is intended on our end, the privacy policy is also required for app store publication, and is reviewed in line with the sensitive permissions that the app asks users to grant it on the host operating system. This review process typically takes 2-3 weeks, and at the time of writing was subject to more delays, which made app store publication not possible within the scope of the project.

Additionally, the review process for app store publishing requires numerous assets and processes to be in place, which are disjoint with the original engineering-based scope of this project. As such, publishing the app requires the involvement of a graphic designer, as well as considerable effort in creating branding and marketing assets. Furthermore, creation of developer accounts requires a monetary investment which should lie with a central, consistent entity that is owned by the project, and not the people working on it. $125 USD is required for developer licenses, and additional costs may be incurred in cloud-computing provider costs, domain ownership and custom email accounts for customer support.

Migrating to an SQL-based schema. Firebase's Firestore is an opinionated database which only supports a noSQL schema, which favours read-heavy applications. Within is a write-heavy application, and should the user base scale to a reasonably large size (for example, 100,000 daily active users), the cost incurred in using Firebase may be higher than if the app were implemented on a separate SQL database. However, the implementation of a distributed SQL database that supports websocket listeners for real-time synchronisation is non-trivial and would occupy the scope of a full project of it's own, and thus is recommended as a future extension if the app starts to gain a large, active user base.

Adding more blocking functionalities: web-extensions and more. A web-client was originally proposed as as stretch goal for this project. However, after considering multiple rounds of user feedback, a few users mentioned that they enjoyed site and app-blocking functionalities from other apps such as BlockSite and Hold. Additionally, the interface for Within is rather minimal and does not require the space of a full browser window. It thus makes sense to implement Within as a browser extension instead of a web client, which will permit for the replication of site-blocking functionality according to a user's specified blacklist.

BlockSite additionally attempts to add app-blocking functionality to its Android app, which displays a pop-up over blacklisted apps should the user try to access them. Naturally, there are limitations to this; the user can simply click off the pop-up and continue using the blacklisted app. Nonetheless, adding an additional layer of deterrence is useful to strengthen the user's resistance to their own involuntary impulses, and is greatly in line with the principles of neuroscience that will make this app an effective productivity-booster.

Ongoing development. The app currently has a small pool of internal testers who may stop using the app for any number of reasons. It is important to constantly increase the pool of test users to continue iterative development and root out bugs in the application. There lies much potential for improvement of features and the user experience on both Android and iOS. An extensive backlog of fixes and feature requests are available in the project repository, which can be developed and implemented in a stable manner thanks to the automated project infrastructure.

6.4 Final Remarks

Within has been a successful exploration into implementing attention-preserving technology using practices and frameworks that allow even a solo developer to produce a working product. The literature surrounding modern apps and their effect on privacy and attention is ever-growing, and there is much promise for the Within app to expand into a platform, offering more attention-preserving features on multiple platforms and offering richer features. In the long term, the Within platform should improve awareness in users and call other developers to action in to effort to develop attention-preserving technology, putting the choice of when to engage with an app back in the user's hands.

Bibliography

[1] E. Abi-Jaoude, K. T. Naylor, and A. Pignatiello, "Smartphones, social media use and youth mental health," *Cmaj*, vol. 192, no. 6, pp. E136–E141, 2020.

[2] U. Lee, J. Lee, M. Ko, C. Lee, Y. Kim, S. Yang, K. Yatani, G. Gweon, K.-M. Chung, and J. Song, "Hooked on smartphones: an exploratory study on smartphone overuse among college students," in *Proceedings of the SIGCHI conference on human factors in computing systems*, pp. 2327–2336, 2014.

[3] K. Demirci, M. Akgönül, and A. Akpinar, "Relationship of smartphone use severity with sleep quality, depression, and anxiety in university students," *Journal of behavioral addictions*, vol. 4, no. 2, pp. 85–92, 2015.

[4] B. Thornton, A. Faires, M. Robbins, and E. Rollins, "The mere presence of a cell phone may be distracting," *Social Psychology*, 2014.

[5] PwC, "Global top 100 companies by market capitalization." Available at https://www.pwc.com/gx/en/audit-services/publications/assets/global-top-100-companies-june-2020-update.pdf (Accessed 27/11/2020).

[6] The Motley Fool, "Twitter inc (twtr) q3 2020 earnings call transcript." Available at https://www.fool.com/earnings/call-transcripts/2020/10/30/twitter-inc-twtr-q3-2020-earnings-call-transcript/ (Accessed 11/11/2020).

[7] CNBC, "Twitter shares sink amid slowing user growth and and uncertainty." Available at https://www.cnbc.com/2020/10/29/twitter-twtr-q3-2020-earnings.html (Accessed 11/11/2020).

[8] N. D. Schüll, *Addiction by design: Machine gambling in Las Vegas.* Princeton University Press, 2014.

[9] M. Csikszentmihalyi and M. Csikzentmihaly, *Flow: The psychology of optimal experience*, vol. 1990. Harper & Row New York, 1990.

[10] C. J. Neyman, "A survey of addictive software design." Available at `https://digitalcommons.calpoly.edu/cgi/viewcontent.cgi?referer=https://scholar.google.com/&httpsredir=1&article=1127&context=cscsp` (Accessed 28/11/2020).

[11] R. C. Colley, T. Bushnik, and K. Langlois, "Exercise and screen time during the covid-19 pandemic," *Health Rep*, vol. 31, no. 6, pp. 3–11, 2020.

[12] L. Wiederkehr, J. Pitt, T. Dannhauser, and K. Bruzda, "Attention enhancing technology: A new dimension in the design of effective wellbeing apps," *IEEE Transactions on Technology and Society*, pp. 1–1, 2021.

[13] V. Rajmohan and E. Mohandas, "The limbic system," *Indian journal of psychiatry*, vol. 49, no. 2, p. 132, 2007.

[14] E. K. Miller and J. D. Cohen, "An integrative theory of prefrontal cortex function," *Annual review of neuroscience*, vol. 24, no. 1, pp. 167–202, 2001.

[15] T. F. Heatherton and D. D. Wagner, "Cognitive neuroscience of self-regulation failure," *Trends in cognitive sciences*, vol. 15, no. 3, pp. 132–139, 2011.

[16] R. F. Baumeister and T. F. Heatherton, "Self-regulation failure: An overview," *Psychological inquiry*, vol. 7, no. 1, pp. 1–15, 1996.

[17] A. Jansen, "A learning model of binge eating: cue reactivity and cue exposure," *Behaviour research and therapy*, vol. 36, no. 3, pp. 257–272, 1998.

[18] A. W. Stacy and R. W. Wiers, "Implicit cognition and addiction: a tool for explaining paradoxical behavior," *Annual review of clinical psychology*, vol. 6, pp. 551–575, 2010.

[19] Google Developers, "Android developer documentation: Notifications overview." Available at `https://developer.android.com/guide/topics/ui/notifiers/notifications` (Accessed 12/1/2021).

[20] C. P. Herman and D. Mack, "Restrained and unrestrained eating.," *Journal of personality*, 1975.

[21] E. C. Levy, S. Rafaeli, and Y. Ariel, "The effect of online interruptions on the quality of cognitive performance," *Telematics and Informatics*, vol. 33, no. 4, pp. 1014–1021, 2016.

[22] M. Muraven, R. F. Baumeister, and D. M. Tice, "Longitudinal improvement of self-regulation through practice: Building self-control strength through repeated exercise," *The Journal of social psychology*, vol. 139, no. 4, pp. 446–457, 1999.

[23] M. Muraven, "Practicing self-control lowers the risk of smoking lapse.," *Psychology of Addictive Behaviors*, vol. 24, no. 3, p. 446, 2010.

[24] P. M. Cinciripini, L. Lapitsky, S. Seay, A. Wallfisch, K. Kitchens, and H. Van Vunakis, "The effects of smoking schedules on cessation outcome: can we improve on common methods of gradual and abrupt nicotine withdrawal?," *Journal of consulting and clinical psychology*, vol. 63, no. 3, p. 388, 1995.

[25] N. Eyal, *Hooked: How to build habit-forming products.* Penguin, 2014.

[26] A. R. Childress, R. Ehrman, A. T. McLellan, J. MacRae, M. Natale, and C. P. O'Brien, "Can induced moods trigger drug-related responses in opiate abuse patients?," *Journal of substance abuse treatment*, vol. 11, no. 1, pp. 17–23, 1994.

[27] B. J. Fogg, "A behavior model for persuasive design," in *Proceedings of the 4th international Conference on Persuasive Technology*, pp. 1–7, 2009.

[28] V. N. Salimpoor, M. Benovoy, K. Larcher, A. Dagher, and R. J. Zatorre, "Anatomically distinct dopamine release during anticipation and experience of peak emotion to music," *Nature neuroscience*, vol. 14, no. 2, p. 257, 2011.

[29] W. Schultz, "Dopamine reward prediction error coding," *Dialogues in clinical neuroscience*, vol. 18, no. 1, p. 23, 2016.

[30] J. Olds and P. Milner, "Positive reinforcement produced by electrical stimulation of septal area and other regions of rat brain.," *Journal of comparative and physiological psychology*, vol. 47, no. 6, p. 419, 1954.

[31] G. S. Berns, S. M. McClure, G. Pagnoni, and P. R. Montague, "Predictability modulates human brain response to reward," *Journal of neuroscience*, vol. 21, no. 8, pp. 2793–2798, 2001.

[32] D. J. Simons and D. T. Levin, "Change blindness," *Trends in cognitive sciences*, vol. 1, no. 7, pp. 261–267, 1997.

[33] D. J. Simons and R. A. Rensink, "Change blindness: Past, present, and future," *Trends in cognitive sciences*, vol. 9, no. 1, pp. 16–20, 2005.

[34] F. Cirillo, "The pomodoro technique (the pomodoro)," *Agile Processes in Software Engineering and*, vol. 54, no. 2, p. 35, 2006.

[35] Flutter, "Supported platforms." Available at https://flutter.dev/docs/development/tools/sdk/release-notes/supported-platforms (Accessed 11/2/2021).

[36] Stack Overflow Insights, "Stack overflow developer survey 2020." Available at https://insights.stackoverflow.com/survey/2020#technology-web-frameworks-professional-developers2 (Accessed 11/2/2021).

[37] React Native Documentation, "Who's using react native?." Available at https://reactnative.dev/showcase (Accessed 11/2/2021).

[38] N. Elmqvist, A. V. Moere, H.-C. Jetter, D. Cernea, H. Reiterer, and T. Jankun-Kelly, "Fluid interaction for information visualization," *Information Visualization*, vol. 10, no. 4, pp. 327–340, 2011.

[39] B. Shneiderman, "1.1 direct manipulation: a step beyond programming languages," *Sparks of innovation in human-computer interaction*, vol. 17, p. 1993, 1993.

[40] D. A. Norman and S. W. Draper, “User centered system design: New perspectives on human-computer interaction,” 1986.