Getting Things Done: A Context-Aware Android Application for Productivity

Author: Federico Milana
Student number: 1419622
Programme of Study: BSc Computer Science
Supervisor: Dr. Jose Such

Table of Contents

Abstract

Productivity at work and life in general can be improved using different methods of organising oneself. David Allen's methodology Getting Things Done is a popular time management methodology that was published in 2001 when mobile phones were unable to offer a digital alternative to a pen-and-paper approach. 17 years later, this project aims to develop an Android application that fully supports GTD without the need of any papers or physical objects while providing additional features only available on a smartphone. The possibility of automating most of the organisation required by Allen's methodology means that the user is not required to know anything about GTD in order to use the application. Time and location awareness of the system were chosen as ways to complement to the philosophy's workflow in order to enhance the productivity of the user.

Originality avowal

I verify that I am the sole author of this report, except where explicitly stated to the contrary.

I grant the right to King's College London to make paper and electronic copies of the submitted work for purposes of marking, plagiarism detection and archival, and to upload a copy of the work to Turnitin or another trusted plagiarism detection service.

I confirm this report does not exceed 25,000 words.

Federico Milana
April 2018

1. Introduction

Productivity at work is essential in order to maximise the results of our efforts and maintain motivation. Likewise, a productive lifestyle ensures tasks to be accomplished in time and contributes to an overall well-being. However, it can be difficult for some to organise themselves in such a way to revolutionise their way of life just by using an agenda or a to-do list, which is why many time management techniques exist.

These techniques, or methodologies, aim at guiding the user to follow specific guidelines and practices that, according to their authors, will facilitate the transition to a more productive lifestyle. The amount of the required user's effort is expected to be as low as possible so that the technique can be incorporated smoothly into everyday life. This excludes having to worry excessively about following a particular procedure and drastically change habits as a consequence.

The book *Getting Things Done* by David Allen, published in 1st edition in 2001 and revised edition in 2015, presents one of the most popular time management techniques, praised as "the definitive business self-help book of the decade" (Caplan, 2007). Essentially freeing the mind of the responsibility of recalling, it allows the users to concentrate their efforts solely on performing actionable tasks, called Items (Allen, 2015, p. 3). While the user is expected to maintain a physical collection of pieces of paper in one single place, the Items are placed into different lists.

Meanwhile, as the number of smartphone users has reached 2.32 billion in 2017 worldwide and is expected to grow further, it is safe to assume that a large number of people who might be interested in *Getting Things Done* own also a smartphone (Statista, 2018). Moreover, existing productivity applications, referred to as business applications, are currently the second most popular type of application on the market after games (Statista, 2018).

Implementing *Getting Things Done* as a mobile productivity application takes advantage of the fact that smartphones are now not only widespread, but they are used frequently throughout the day, with 76% of users doing it more than 3 hours (Statista, 2018). This allows the user to rely on the presence of her lists of Items at any time while being reminded of which Items are relevant at any specific moment.

It is therefore the aim of this project to offer the organisational GTD-type methods in the mobile environment. The effort required for the user to adapt to Allen's

methodology is also minimized by taking advantage of the target platform features as it will be discussed in the report.

1.1 Motivation

While there are numerous different time management techniques, *Getting Things Done* stands out as one of the most well-known and successful ones. It tackles issues like multitasking, email organisation and task delegation that are as relevant today as they were in 2001.

Allen emphasises various examples of how the philosophy can be adopted by entrepreneurs and businessmen to organise their work life. While it is true that *Getting Things Done* was formulated, tested, and implemented in the training and development of the corporate world, but its concepts are highly compatible with any work environment and everyday life (Allen, 2015, p. *xix*). Lists like *Reference* can contain a recipe just like an email address, while *Maybe/Later* can contain learning the piano just like asking for a raise.

Other than a wide applicability, this particular technique offers considerable flexibility in terms of when to engage with Items and Lists. In fact, the only strict requirement concerns the *Clarification* of every *In Tray Item* once a week. Otherwise, the user is free to perform any of the *Five Phases* of the workflow (*Capture, Clarify, Organise, Reflect* and *Engage*) at any time and in any order, which is a good example of non-intrusiveness of time management techniques.

The digitalisation of *Getting Things Done* can be motivated by considering the numerous advantages over the intended pen-and-paper approach. The first great advantage of using a mobile application for the user is not having to read Allen's book in order to understand how the methodology works. At the same time, this poses a burden to the system, which is expected to guide the user through the basic concepts without an overwhelming amount of instructions and descriptions. However, if this is done successfully, the first barrier of entry to GTD technology is removed as users start using the system straight away.

Another advantage is the ability to keep *Items* and *Lists* as close to the user's needs as much as possible, providing the option of performing every action in the workflow at any time of the day (see Appendix A). While this might be a sufficient reason for somebody to prefer a software over a physical approach, the application includes additional functions that are only possible on a smartphone.

Developing for a mobile operating system allows the implementation of push notifications, which has the potential of increasing the user's productivity. This reduces the effort spent on opening the application to check for items that are due. Notifications work in favour of the methodology's goal by focusing on taking action on tasks instead of recalling.

By using the software, the user is freed not only from recall, but also from the organisational skills that the methodology expects. In fact, the biggest amount of effort that is required from the user by a pen-and-paper approach goes into managing a large number of lists on paper efficiently. These lists can be grouped inside a single application as long as the user prefers without requiring any physical space.

For a similar reason, a digital environment can include user accounts linked to a cloud-hosted database so that user-submitted data can never be lost. In the event of losing the smartphone, it would be simple for the user to log in from a different device and access the same Items and Lists. On the other hand, in a physical environment, misplacing an Item, for example, could possibly mean never getting it back.

1.2 Scope

The project consists of developing a mobile application that facilitates the implementation of the GTD methodology. The application was developed in order to fully replace a pen-and-paper approach in the form of lists, calendars, location-based features, and notifications.

The full implementation of this productivity application will provide various benefits to the user, such as portability and a centralisation of resources and services. Not only will the user be able to perform any of the *Five Phases* of Project Planning (*Capture, Clarify, Organise, Reflect and Engage*) at any time and place, but she will also perform this within a concise and organised environment.

This also allows the possibility of features that are not strictly part of the GTD philosophy, such as reminders and location-based services, which can be implemented in a digital environment in order to enhance the user's productivity.

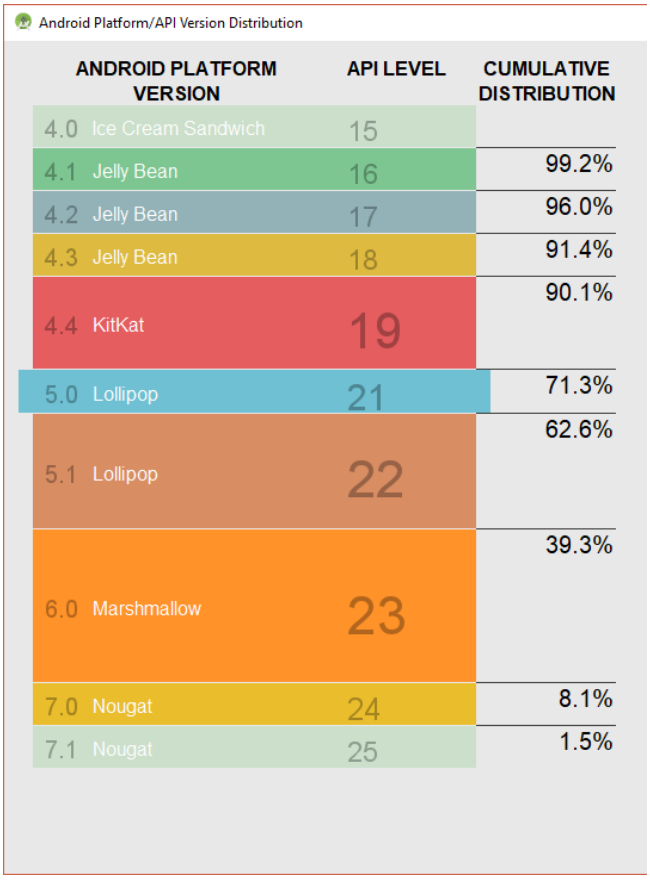
1.3 Target Operating System

The mobile operating system chosen was Android, for various reasons. Firstly, it is by far the most widely used mobile operating system with 87.7% of market share in 2017 (Statista, 2018). It was important to develop for a wide public, as we believe Allen's methodology should improve the productivity of as many people as possible. Developing for *iOS* instead would have led to excluding a very large share of users who might benefit from using the system.

One aspect taken into consideration is that Android devices are, on average, cheaper and less powerful smartphones than *iPhones* (Richter, 2018). However, this was not considered an issue as the application being developed was not to implement any feature that would require a particularly high amount of processing power from the smartphone.

Another reason for choosing Android is that, from a user interface perspective, it was clear that Google's visual language *Material Design* was perfect as guidelines for developing the front-end of the project (Material Design, 2014). Since there were to be many aspects of the interface unfamiliar to the user (GTD terms, tags, *Clarification* popups) *Material Design* would provide enough whitespace and consistency with other system applications. This would be useful to de-clutter the information shown on any given page and familiarise the user with the interface.

The next decision regarded the choice of the API level as the minimum SDK that the application required. This was a delicate matter, since the system updates on Android are generally slow to roll out to older devices, with more than 90% of users using a version of the operating system released at least 2 years ago (as shown in the version distribution displayed in *Android Studio*) (Android Studio, 2017). This meant that choosing a newer API as the minimum SDK while providing useful new features could potentially exclude a very large amount of users from using the application.



Android Platform API Version Distribution

The thought process behind this decision was to target the oldest API which supported lock screen notifications. These were important because the user would be reminded of an Item relevant in that particular moment right away without having to launch the application and search for it manually. This tied well with the action-rather-than-recall concept as well as minimising the user's effort.

The API chosen was level 21 (5.0 *Lollipop*), with a cumulative distribution of 71.3%. This was an appropriate compromise between cumulative distribution and available features needed, mainly lock screen notifications for reminders.

1.4 Objectives

The main objective of the project is to create an Android application that fully automates the *Organise* phase of the *Five Phases* of Project Planning (see Definitions), while centralising resources and providing location-based notifications and action reminders.

The user will have to register for an account and its data will be stored in a cloud-based database.

The system should be accessible to users who have never heard of Allen's book *Getting Things Done* before, while still using its terms and concepts. This should be supported by a design language that recreates a visual environment familiar to the user and as least confusing as possible.

2. Review

Background research for this project consisted mainly in reading *Getting Things Done* in order to generate functional requirements and evaluating existing to-do and reminder mobile applications on Android.

Allen's book was essential in understanding the philosophy of GTD, but the absence of a clear specification of some concepts required arbitrary decisions in the requirement gathering phase. The book was intended as a guide to users rather than developers, so sections such as *The Path of GTD Mastery* were not as useful as sections that explained the workflow, which were used to guide the implementation of question popups to *Clarify* items.

However, parts of the book that were aimed at users were still taken into consideration in order to design towards an experience that is similar to the one described by Allen. For example, missing *Weekly Reviews* of In Tray is explained as one of the easiest ways of getting off track (Allen, 2015, p. 289). While this is not a concept that can be translated directly into a requirement of the system, it was the primary reason for implementing weekly reminder notifications.

On the other hand, the analysis of existing productivity applications was useful in identifying features and design choices that could be implemented in the project. These applications did not necessarily implement GTD concepts, but still shared significant similarities with the system in mind.

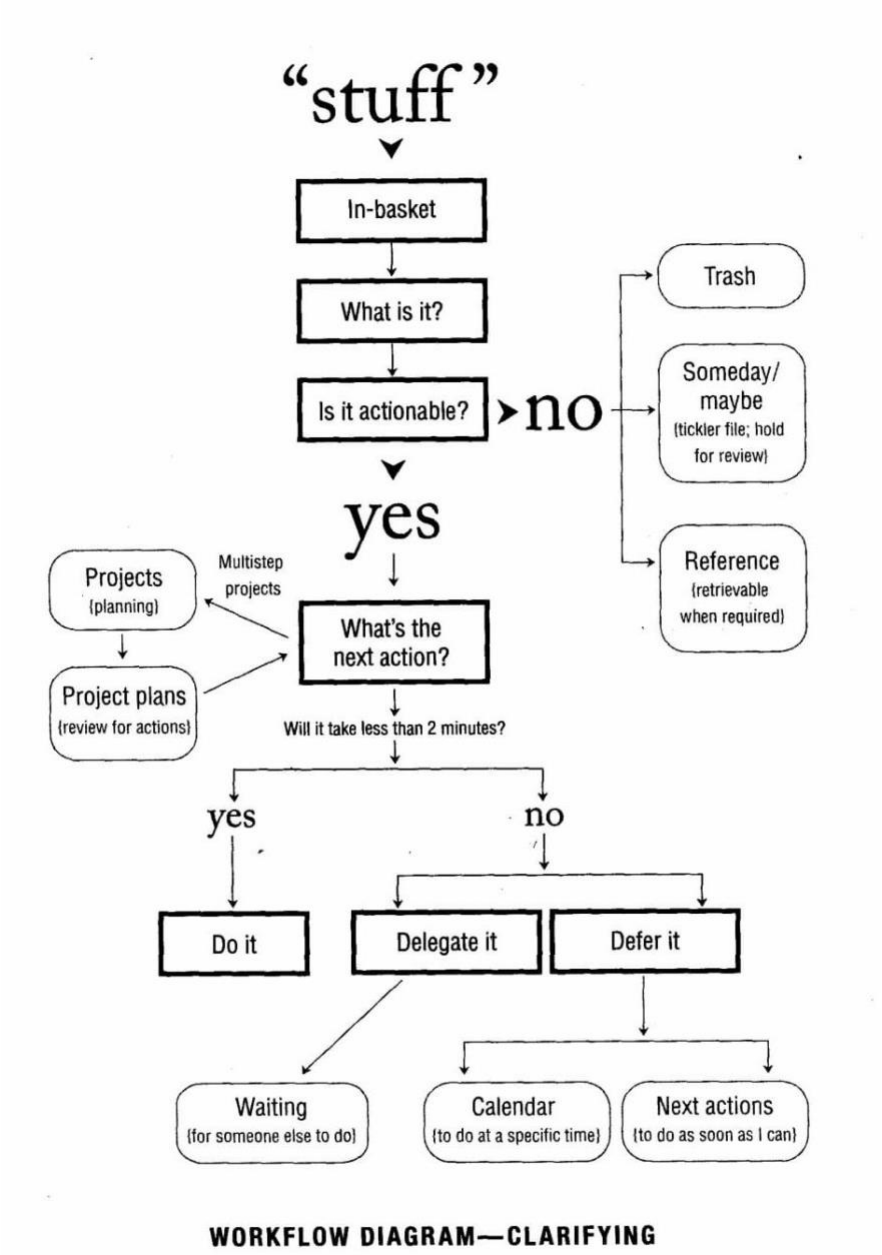
Existing work that did implement *Getting Things Done* was also evaluated, but, as explained later in this report, did not set a particularly insightful example of how the final system should be. Nevertheless, these applications represented a starting point that encouraged improvement in different areas.

2.1 Literary review

Reading *Getting Things Done* brought to surface a number of issues that could potentially cause problems in the development phase of the project. For example, there is a fair amount of ambiguity in the definition of a list. Lists are defined as containers of *Items*, and should be limited to *In Tray, Projects, Calendar, Waiting For, Maybe/Later*, *Reference*, and *Trash*. While each of these has a distinct purpose, it is unclear what type of physical aspect they should have in a pen-and-paper approach. This is intentional in order to give freedom to the users in organising themselves as they prefer. If we also consider the fact that an Item could be virtually anything, from an idea to a pair of keys, it makes sense for a list to be an abstract idea of a collection rather than a fixed concrete container of Items.

In a digital environment, however, Items must be treated as representations of ideas or physical objects. There is also much less freedom to create a personal type of list; the *Calendar* is just like the Android system calendar, In Tray a list of text-based Items. While Allen does acknowledge the benefit of combining *Getting Things Done* with a smartphone in the revised 2015 edition, he opts to continue to devote to paper-based tools and materials (Allen, 2015, p. *xvii*). This meant that many steps in the translation from a pen-and-paper to a digital approach had to be inferred.

Another main issue in reviewing Allen's work was the identification of concepts to prune or change in order to lower the barrier to entry for users who know nothing about *Getting Things Done*. The GTD workflow is a good example of this. As the user goes through the workflow, the Item is Clarified and Organised into the appropriate list.



GTD workflow with Clarifying stages of the process shown in bold rectangles

The main advantage of having a workflow is that the user is guided through a set of questions that, if answered correctly, result with no Item in the wrong list. It removes the responsibility from the user to figure out by themselves how to treat a given Item. This is especially valuable to new users, and it is arguably the most valuable concept of the methodology that can be applied without being familiar with the rest of Allen's work.

However, a few aspects of the workflow were working against this idea and were thought to be slightly incompatible with a digital implementation. With evidence that

suggests that 24% of users abandon a mobile application after just 1 trial of use in 2017, it is crucial that actions are done as quickly and simply as possible (Localytics, 2017). For the same reason, actions should leave the users satisfied of the effort they required without needing further steps to complete. This excludes the possibility of loops in the workflow, namely the one concerning Projects.

Allen considers the Projects list as a temporary placement for Items that should eventually be reviewed. In other words, an Item in Projects a desired result that requires more than one action step (Allen, 2015, p. 41). While this works well with Items that are too challenging to complete as a whole, it requires further actions to be made in order to split these Items into actionable sub-Items. If, instead, the user is instructed to Capture Items that are not too large and do not need to be reviewed, the loop in the workflow does not need to exist. In this way, the Projects list can be considered simply as a list of projects, each of which containing a list of Items.

A possible extension of the workflow was also noted while reviewing Allen's book. One of the outcomes of deferring an Item expects the user to complete the Item as soon as possible. Allen suggests to organise as-soon-as-possible actions in context by stating that "the best way to be reminded of an as-soon-as-I-can action is by the particular context required for that action" (Allen, 2015, p. 146).

The main issue with this approach is that it ignores the case in which the user forgets about the particular Item completely, which is easy to imagine when dealing with a large number of them. However, by attaching a reference of context to an Item, such as a Tag, a mobile application can remind the user of all the Items that are relevant at any given context.

This solution tackles another issue often brought up as criticism of *Getting Things Done*, namely the excessive emphasis on context-awareness that stands in the way of performing actions. In other words, instead of limiting worrying about what to do at any given moment, GTD often increases stress as people try to figure out which tasks really are the most important ones to work on (Lifehack, n.d.). The seamless inclusion of notifications removes this burden completely by shifting the responsibility from the user to the system.

Despite notifications appearing to fit well into the GTD workflow by reducing effort required by the user to determine which Item to complete, a research in 2015 found evidence that cellular notifications, even when one does not view or respond, can significantly damage performance on an attention demanding task (Stothart, Mitchum and Yehnert, 2015). This concern the case of a productivity application, which aims to achieve the opposite.

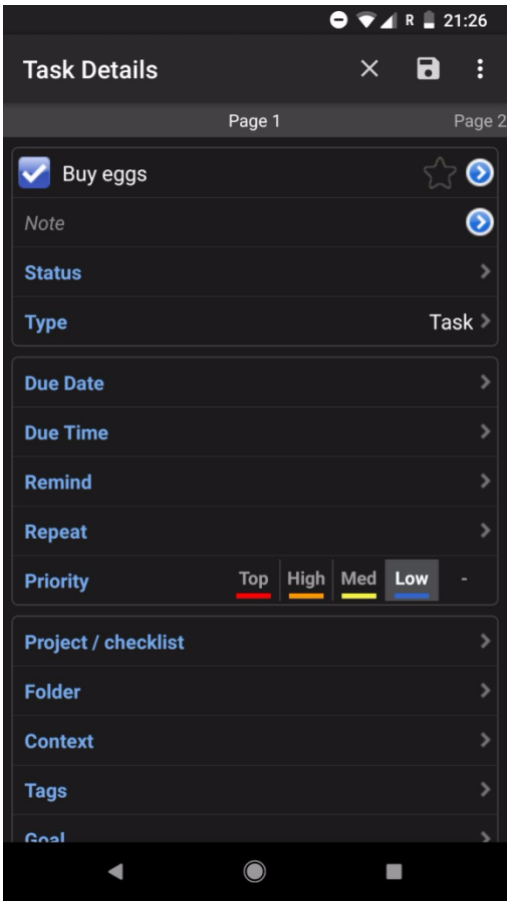
As a solution to this tendency, it was thought that the system could give absolute control of notifications to the user so that, for example, the *Weekly Review* reminder appears at a selected day and time. Also, location-aware notifications were thought to trigger while the user enters a specific location, meaning that the chances of the user performing a task demanding attention are much lower.

2.2 Existing work

The existing work that was analysed in the research phase of the project consisted of two main types of mobile applications present on the Google Play Store (Google Play, 2017). One was productivity applications that claimed to implement *Getting Things Done*, and the other was popular productivity applications that had no concept of it.

The first type was reviewed in order to compare the understanding of the methodology acquired from the book to other developers in the context of a digital representation of Allen's concepts. The first mobile application considered was DGT GTD & To-Do List (DGT GTD & To-Do List [Alpha], 2010). Despite it being in alpha stage of development, it is currently one of the most downloaded *Getting Things Done* mobile applications concepts with 100 thousand downloads.

What was striking about this implementation is the very large amount of features that the system provides. Each Item can be edited to add notes, status, type, due date, due time, reminder, priority. Given that none of these are concepts from Allen's methodology, the developers' priority was to allow the user to customise each task as much as possible. As the focus is placed in Items, lists presented in Allen's book are not implemented, and neither is the workflow mentioned earlier.



Task details page from the DGT GTD & To-Do List mobile application

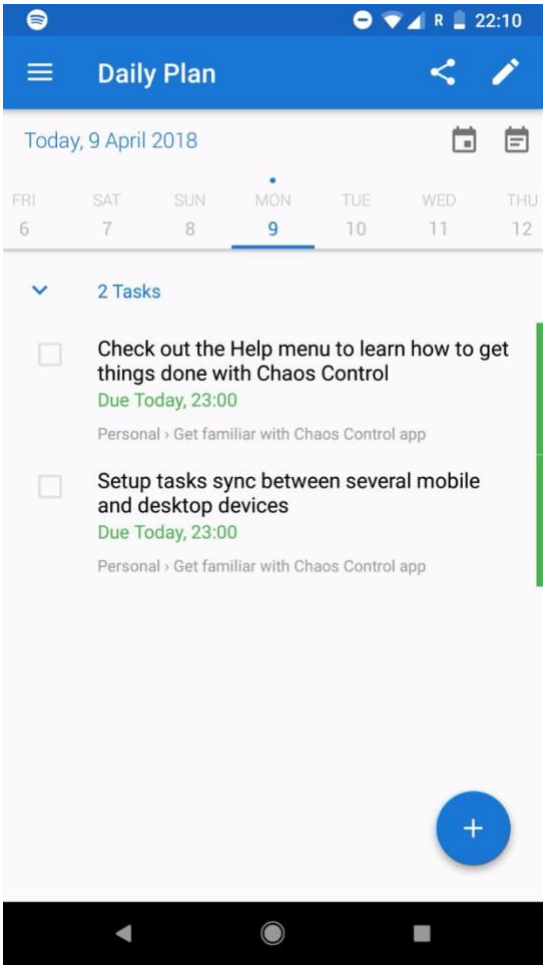
This approach was considered ambitious in the sense that it detached the system almost completely from the methodology by avoiding the use of terms such as *In Tray*, *Clarify*, *Item*, *Capture* and discarding the philosophy's core concepts. While terms are definitely more familiar to the user, it is difficult to recognise almost any aspect from *Getting Things Done*.

The design of the user interface was the exact opposite of what the project intended to deliver. The user is placed immediately in the position of accessing an overwhelming amount of settings in each page, some of which are redundant, like context and tags. It was in this stage that the decision of a simple and clean design language was confirmed. In addition, the absence of a tutorial screen to explain the system to the user was noted as a mistake that the present project would avoid.

The application understanding of projects was also noteworthy, as it matched the one that was acquired from the literary review. Projects are treated by the system as a collection of tasks (Items) instead of Items that should later be reviewed. However, this implementation is probably due to the fact that the GTD workflow is not implemented at all, and, therefore, neither is the review process.

After analysing DGT GTD & To-Do List, it was clear that such application had little to do with *Getting Things Done*, unlike the one delivered by the present project. In order to avoid excessive deviation from Allen's work, the proper terms (or similar, in case of copyright complications) had to be used as well as the GTD workflow of *Capturing*, *Clarifying*, *Organising*, *Reviewing,* and *Engaging*. By using terms that might be unfamiliar to the user, the user interface was to be developed in such way to avoid displaying too much information at any given page.

The second mobile application reviewed was *Chaos Control*: *GTD Organizer & Task List Manager* (Chaos Control: GTD Organizer & Task List Manager, 2012). The introductory tutorial of the system was effective in explaining the various functions of the application and, unlike the implementation previously mentioned, the amount of interactive user interface elements shown at once was kept low. The use of *Material Design* was done effectively by following Google's guidelines and was noted as something the project should try to emulate.



Landing page from the Chaos Control: GTD Organizer & Task List Manager mobile application

However, as in the case of the first analysed system, several design choices were made that did not seem to go in the same direction as the objectives of this project. For example, the focus being the calendar, which is the landing page of the system, Items are placed directly in selected days, which contradicts the *Capturing* process, which allows Items to be placed inside In Tray only.

The application does provide the user with an *In Tray* called *Chaos Box*. However, because the only other list is the calendar, every Item going through the *Clarification* process is either moved to a date or kept in the *Chaos Box*. Again, the GTD workflow is discarded by removing almost every list mentioned by Allen.

The context was also implemented in a similar way that was imagined to be developed in the final application of this project. By creating tags, the user can use the same tags for different Items and provide context such as “weekend” or “morning”. However, these tags are limited to just text and are displayed only when the user selects an Item. Time is set directly when creating or editing an Item, while location is

absent. Because context is essentially the time and place in which to complete an Item, this implementation of tags was not to be emulated in the final system. It was decided at this point that Items should not contain any type of context by themselves, but rather have context tags attached that did.

The inclusion of “cloud sync” for premium accounts was promoted as a way of synchronising data across mobile and desktop platforms. While the scope of this project does not foresee the development of a desktop application alongside the mobile one, user data will still be stored in a cloud-based database. The advantage of this is limited to keeping data when switching smartphones, but facilitates a possible future desktop platform to exist.

Another feature offered to premium accounts is the possibility of adding a “password lock” in order to secure the user’s data. This could prove to be a useful feature in the case where the user is *Capturing* sensitive information. However, the implementation of a cloud-based database requires the user to log in using an email address and a password, after which it is always possible to log out. The option of using a password lock might be redundant and never used by the user. Nevertheless, the inclusion of a security pin was noted to be considered in the requirement gathering phase of the project

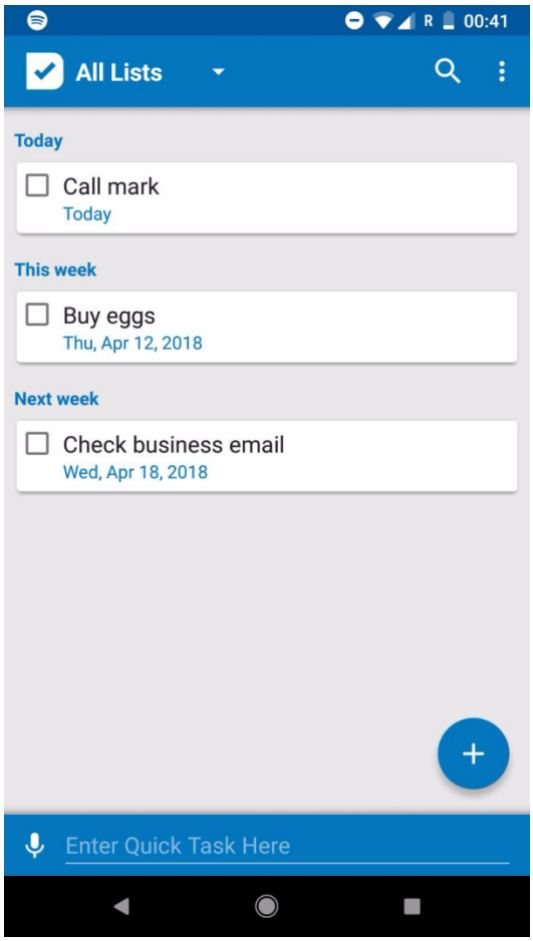
The second type of existing work analysed was the one of general productivity applications that made no claim of implementing any aspects of *Getting Things Done*. These provided insights into how to manage and display user-submitted data, how to design the navigation between different pages and which features to replicate. These applications were *Google Keep*, *Wunderlist* and *To Do List*, and were selected because of their extremely high popularity in the Google Play Store (Google Keep, 2013) (Wunderlist, To-Do List & Tasks, 2011) (To Do List, 2013).

The three systems shared various design choices, such as giving more visibility to the list of inserted items or notes by placing the list directly in the landing page. This allows the user to minimise the number of actions required between thinking of something and writing it down.

Replicating the placement of the list of inserted items in the landing page was thought to be useful for additional reasons. According to the GTD workflow, Items can only be inserted in *In Tray*, from which they can later be *Clarified* and moved into other lists. Also, the user is expected to insert every Item that comes to mind in order to avoid having to recall ideas and things to do. It is likely that the user thinks of new things to do frequently during the day, in which case a new Item should be *Captured*. Forcing the

user to navigate from the landing page to *In Tray* is likely to discourage the *Capturing* process in the long run.

Regarding lists of items, the implementation of *To Do List* stands out from the other two as it divides items by date, grouping them based on whether they are due today, the current week, or the following weeks. This could also be achieved in *In Tray* Items based on the Tags attached to each Item indicating the deadline date. However, since In Tray should be emptied each week during the *Weekly Review*, it would be best to group Items this way directly in the Calendar list.



List of items from To Do mobile application

In terms of navigation, *Google Keep* was the only application out of the three which has more than one page to display. In the case of *Wunderbar* and *To Do*, the fact that there was only one page even excluded the need of a navigation menu. The centralisation of the application functions into a single page can be considered as a head start in terms of learnability of the system. Due to the absence of hidden functions that are only visible after navigating to their appropriate page, the user is familiarised immediately with the entirety of the user interface.

The nature of *Getting Things Done* excludes the possibility of such approach, since each list, such as *Maybe/Later*, *Reference*, and *Trash*, should be given equal importance in terms of visibility. This meant that the application to be developed would present the user with a navigation menu similar to *Google Keep*'s, but containing a larger number of lists. In order to approach the learnability of *To Do* and *Wunderbar* user interfaces, these lists could be designed to look and behave as similarly as possible.

Overall, the evaluation of existing work revealed that there is enough motivation to develop a productivity application that fully implements *Getting Things Done*. The systems that claim to implement Allen's work do this by removing a substantial amount of concepts of the methodology and leave space for improvement. The analysis of these applications, including generic ones like *Google Keep*, *To Do*, and Wunderbar, was crucial in identifying aspects and features to replicate and those to discard.

3. Specification and design

The specification and design of the project were initially drawn from the reviewed literature and the evaluation of existing work. It was decided to refine these by developing a prototype of the application using wireframes in Balsamiq (see Appendix B) and conducting usability testing in order to receive feedback data from a sample of participants (Balsamiq, 2008).

The usability testing phase resulted in various changes to the initial non-functional requirements, while functional requirements remained unchanged. Notes of relevant issues during testing were particularly useful in identifying issues with the user interface before development. The participants were also asked to answer a questionnaire that included the System Usability Scale, which is specialised to be used immediately after the respondent has had the opportunity to use the evaluated system (Brooke, 1996, p. 194).

The specification of the system was illustrated with use case diagrams and state machines, which acted as guidelines during the development phase. Meanwhile, the design of the system was in constant adjustment throughout the review, usability testing and even development phase, which will be explained in detail in the Implementation section of this report.

3.1 Requirements

3.1.1 Requirement gathering

Requirements were initially gathered during the literary review. This process consisted mainly in extracting functional requirements from Allen's book by identifying the features of a GTD working environment.

The first functional requirements were identified from the main concepts of the Allen's (2015) book. The system should display every GTD list, that is *In Tray*, *Projects*, *Calendar, Maybe/Later, Waiting For, Reference*, and *Trash*. Each of these lists should be able to contain Items that the user can only initially enter in *In Tray*. After going through the clarification process for an Item, the system should move the Item from In Tray into the right list. Since *Getting Things Done* emphasises context-awareness over priority for each Item, the user should be able to create Tags that give context to Items.

The initial understanding of a Tag was that it should have a name and fall under one of three categories: *Time*, *Place,* and *Other*. Allen describes context as the place and tools available at the moment. However, it would be impossible for the system to deduce what kind of tools are present at any given moment unless the user describes their surrounding to the system periodically. This alternative approach would require too much effort from the user and would reach very high levels of intrusiveness in everyday life. Instead, Tags were thought to be a specific context, such as a time or a place, that could be re-used for many Items.

Additional functional requirements, such as the inclusion of user accounts and a cloud-based database, were gathered during the evaluation of existing work. For example, as applications such as *To Do* and *Google Keep* offer the option of synchronising data across different devices, it was noted that a similar feature should be offered by the system. This requirement was supported by research that reveals that the current average smartphone user replaces the device every 21 months, meaning that a method of storing data is required (Statista, 2018).

Non-functional requirements were also gathered from the evaluation of existing work. Firstly, the implementation of an introductory tutorial was considered necessary to avoid the type of confusion with interface elements that was encountered while analysing *DGT GTD & To-Do List*. This would also provide the chance of explaining the basics of *Getting Things Done* to users who are not familiar with the methodology.

Another non-functional requirement gathered in this phase was the need to automatise the Clarification process in order to ease the initial learning curve of the GTD workflow. As will be further explained in the non-functional requirements, this was implemented through the use of popup dialog questions translated directly from decision steps from the workflow flowchart.

The decision of using *Material Design* as the design language of the system was confirmed in the comparison between the applications evaluated. Existing work that implemented this, namely *To-Do* and *Google Keep*, showed much less information on any given page, while displaying user interface elements such as the *Floating Action Button*, which are present in Android's system applications such as the *Calendar*. Implementing a similar interface was recognised as a non-functional requirement that would allow the user to ease the familiarisation with the working environment of *Getting Things Done*.

3.1.1.1 Prototype usability testing

The gathering of these initial requirements led to the possibility of creating a wireframe in Balsamiq to use as a prototype for usability testing (Appendix B). The motivation behind usability testing in this phase of the project was that issues with the initial requirements could be identified before developing any code. This decision was supported by the evidence that the cost of change of a system is generally substantially lower in the phase of requirement specification than in the development phase (Oldfield, 2003, p.4).

However, despite the fact that this form of testing provides an almost infallible indicator of potential problems and how to resolve them, issues with this type of testing were also considered (Rubin and Chisnell, 2008, p. 26). For example, one major limitation is that not all functionality within the system can be analyzed to a satisfiable extent. The prototype that was developed could not represent user actions such as long pressing an item or inserting text into text fields.

Moreover, the prototype was tested on a computer, when the final system was intended to run on a smartphone. As a consequence, the evaluation of the results was affected as actions could not be measured in time taken. The obtained data could not be considered of sufficient value as the interaction between the user and the system was through a mouse instead of a touchscreen.

Based on the functional requirements gathered up to this stage, 10 participants were each assigned 12 tasks to perform (see Appendix F). Data gathered consisted of observation of any usability issues that appeared when performing these tasks, as well as any comments made by the participants. Participants were asked to complete a questionnaire at the end of the usability testing (see Appendix G).

The first part of the questionnaire aimed at gathering data that could justify or refute the implementation of features including a tutorial, extensive control over notifications and popup dialogs.

The second part aimed at quantifying the usability of the system by using the System Usability Scale developed by John Brooke in 1986. This would facilitate the assessment of the prototype by producing a score based on the participants' replies. It was also an appropriate method to use given the small amount of participants as one of its main benefits is the ability to produce reliable results from small sample sizes (Usability.gov, n.d.).

The final part of the questionnaire consisted of questions that were specific to the system. Participants were required to describe their experience with a number of aspects of the user's experience and to illustrate possible extensions of the functional requirements of the system. Gathering system-specific data was aimed at taking a user-centred approach in order to clear uncertainties as to how to implement the navigation, Tag creation, Item Clarification and other features.

The results of the prototype usability testing prompted various changes in the requirements that were defined on the basis of the literary review and the evaluation of existing work. The requirements were instead confirmed in cases where the gathered data supported them.

Regarding the performance of tasks on the prototype brought to light some design choices that made little sense to the user. For example, the Clarification of an Item was problematic to start, as the placement of the *Clarify* button was unclear. Instead of clicking on the *Item to Clarify* to show the *Clarify* button, participants tended to check elsewhere. Since the Clarification is, perhaps, the most important step in the GTD workflow, changes had to be made in order to give deserved visibility to the *Clarify* button. A *Clarify* button was therefore placed below each Item to avoid confusion.

An additional issue that was identified was where to create a new Tag. In the tested prototype, Tags could only be created from an Item, that is only after selecting an existing Item with its page opened. However, participants were asked to create a Tag

for the first time from *In Tray*. This meant that the participant had no clue that the Tag creation button was placed inside the Item page, and therefore tended to look in the navigation menu. At this stage of the project, the navigation menu contained just GTD lists as there was no intention of mixing these with a feature that was not directly taken from Allen's book.

A small number of usability problems were justified by the nature of the platform on which the prototype was developed. For instance, when participants were instructed to delete an Item, the majority of them noticed that long-pressing an Item did not result in any feedback from the system, and therefore looked for a *Delete* button elsewhere. While it was not possible to implement a long-press interaction in the prototype, this was indeed the way that the interface was intended to behave.

Meanwhile, results given by the questionnaire were also especially useful in identifying user preferences. It was found that 100 per cent of participants who use a productivity mobile application find reminders useful. Since the system intended to use these regularly, this data was an indication that this feature would be welcomed by users.

One type of data that contradicted the implementation of one of the requirements was the need to have an extensive control over notifications. With an average of 2.8/5 on the Likert scale, participants did not express a particular desire of having the ability to change the notification settings. However, this does not take into account research that, as mentioned earlier in the report, suggests that notifications may disrupt the user's productivity. For this reason, control over notification was still going to be implemented in the final system.

The System Usability Scale in the second part of the questionnaire produced a score of 81.25 out of 100. While this was not particularly insightful in terms of the context of the specific system being developed, it represented an indication that the final application should not deviate too much from the prototype from a user interface perspective.

Furthermore, the last set of results produced data that explained the issues encountered by the participants while performing the tasks given. For example, the participants confirmed that the Clarification process was difficult to start because of the button placement.

Finally, the introductory tutorial turned out to be the best way to explain the functionality of the application (80% of participants included this option in their answers).

The second preferred method of explanation was about a page with summary of concepts. However, the intention was to implement one method only, so that explanations do not become redundant to the user. Following the indications gathered in this question, the tutorial was included as a non-functional requirement.

3.1.2 Functional requirements

The complete list of functional requirements of the system gathered during this phase is as follows:

- User accounts
    - *Email-password login*
    - *Password recovery*
- Database integration
    - *Authentication management*
    - *Storing user-submitted data*
- Items:
    - *In Tray* - Items can only be inserted here
    - *Trash* - Items discarded in the clarification phase
    - *Maybe/Later* - Items that might be useful in the future
    - *Reference* - Items to remember
    - *Projects* - Folders containing Items
    - *Waiting for* - Items that will be useful once something happens
    - *Calendar* - Items that have a due date
- Editable context Tags to be attached to Items:
    - *Time* - Time and day of the week
    - *Place* - Location
    - *Time and Place*
    - *None*
- Location awareness in order to:
    - *Send notifications when in predetermined Contexts*
- *Calendar daily reminders*
- *Weekly Review* - Weekly reminders
- *Exhaustive control over notification* settings

Firstly, user accounts should be implemented in order to authenticate the user with an email-password log in, which allows the inclusion of a cloud-based database. An alternative approach would have been to store data in a local database, which was discarded for two reasons. It would force the user to manually copy all of their data

when changing smartphone, and it would exclude the room for a future extension of the system, namely a desktop client.

The lists of Items should be implemented as separate collections, each containing Items that have been Clarified from *In Tray*. A different way that could have been implemented was one which allows the user to insert Items in any list, not just in *In Tray*. In Tray was kept as the only place where to insert Items in order to conform as much as possible to the GTD workflow. This approach has the advantage of taking into consideration the scenario in which the user already knows where the Item should be placed. As will be discussed in the design section, this issue was addressed in a different manner.

Context Tags were initially designed to consist of either specific times, locations or other. By creating different *Time* and *Place* Tags, the user would be able to reuse these for different Items in order to represent each context in the system. The *Other* category would be included to cover the case where the user intends to use a Tag as a text label only. The *To-Do* and *DGT GTD & To-Do List* tags work in a similar way.

However, this choice posed design problems in terms of different *Time/Place* combinations available to the user. For example, in the case that the user wants to create a “Lunch time at work” Tag, it would be impossible with this current approach. Instead, two different Tags should be created: “Lunch time” (a *Time* tag) and “Work” (a *Place* Tag). The final requirements stated that each Tag should not fall under one category, but rather include any combination of *Time* and *Place*. The reduction of the number of steps in creating a context was desirable in terms of the usability of the system.

Calendar notifications should be sent in the morning of each day that contains Items, so that the user is reminded of Items that could possibly have been forgotten. Ideally, this notification should be sent before the user wakes up in the morning, so that the user should previously set the time of this reminder. Similarly, the *Weekly Review* time should be set by the user in terms of the day of the week and time, since the review should happen in a moment of spare time.

Finally, reminder notifications were limited to *Tags*, *Calendar*, and *Weekly Review.* Tags should send notifications only when the context set matches the user’s. For example, it is 1:00 pm on Friday and the user is at home. The Item “Try the new French place for lunch” has the Tag “Lunch time at work” attached. Despite the Tag having its time set on 1:00 pm Monday to Friday, the system should not send a

notification because the work location on the Tag does not match the actual location of the user.

3.1.3 Non-functional requirements

The non-functional requirements of the system gathered at this phase of the project are listed below. The system should implement:

- Material design in order to:
    - Adhere to Google's design standards
    - Use whitespace to declutter information
- The use of an introductory tutorial in order to explain the basics of GTD and the functionality of the application
- Popup questions in the Clarify Phase when clarifying items in *In Tray* to automatise the *Organise* phase
- A navigation drawer where to access and display lists separately

One of the main concerns that these requirements attempt to address is the steepness of the *Getting Things Done* learning curve. The system should be accessible to users who are not familiar with Allen's methodology by removing hurdles in the acquisition of core concepts. For example, relying uniquely on the user in their ability to determine which Item goes in which list would likely result in various mistakes. Furthermore, the meaning of terms like "In Tray" and "Reference" are not immediately clear to the average user, and need some kind of explanation. As determined in the requirement gathering phase, this was to be done with an introductory tutorial.

Popup questions were to include the questions from the GTD workflow, and appear every time the user Clarified an Item. This requirement was intended to increase the level of engagement from the user in order to leave them satisfied of having successfully clarified an Item. Questions were to be answered by Yes or No answers, so that the process could be finished in seconds and the user did not consider the Clarification process a chore.

Finally, since the number of lists is large, at least compared to the applications previously reviewed, it was important to design the navigation between their respective pages that was simple and intuitive. Following *Material Design*, there were several options set by Google to be considered (Material Design, 2014). The final decision was to implement a drawer menu, as will be explained in the Design section of this report.

3.1.4 MoSCoW prioritisation matrix

The MoSCoW prioritisation matrix of both functional and non-functional requirements reported below illustrates the priority given to each requirement during the development phase.

MUST HAVE	SHOULD HAVE	COULD HAVE	WON'T HAVE
Editable Getting Things Done lists	User accounts	Offline access	iOS support
Calendar reminders	Cloud-based database	Social media integration	
Weekly review reminders	Location awareness		
Popup questions in the Clarify Phase	Context Tags		
Introductory tutorial	Material design		
Exhaustive control over notification settings			

Regarding the "Could have" section, these were requirements that were not explicitly considered during the review and requirement gathering phases of the project. Offline access would be required when data from the cloud-based database cannot be retrieved due to the absence of an internet connection. While this feature would allow users to access the application in every possible scenario, under this column something that would be developed is listed if enough time were available. The priority of the rest of the requirements was simply too much higher to implement this in an early stage of development.

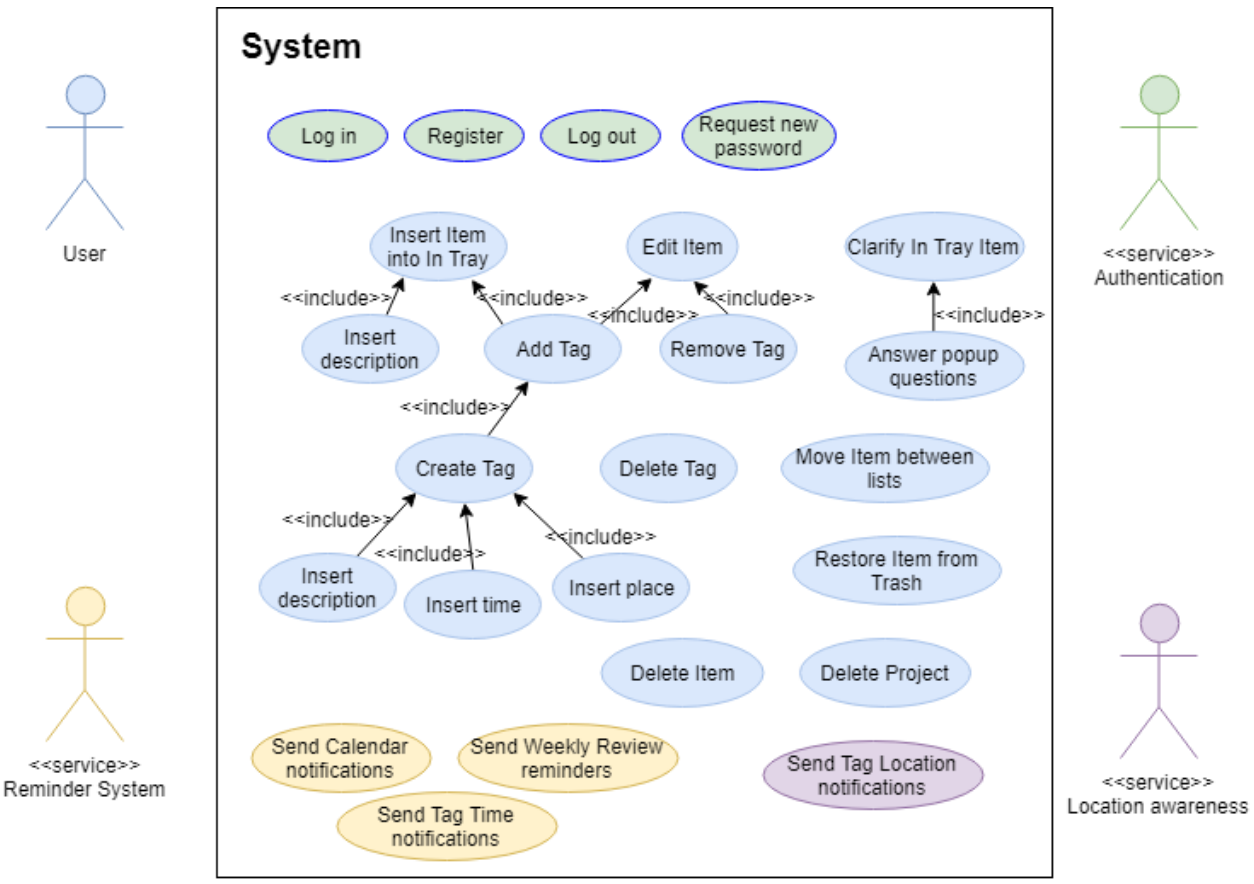
On the other hand, social media could take form in the sharing of Items or Projects that require group effort between the users of the application. By connecting with Facebook or Twitter, it would be possible to add friends who use the application in order to send them relevant Items or Projects (Facebook, 2004) (Twitter, 2006). This requirement was placed under "Could have" as it would substantially increase the scope of this project. Also, results of the usability testing questionnaire quantified the

usefulness of this feature to 1.6/5, meaning that social media integration would likely be overlooked by users.

3.2 Specifications

The specifications of the system are explained below with a use case diagram and a state machine.

3.2.1 Use case diagram



The four agents of the system were identified as the User, the Reminder System, Authentication and Location Awareness. Every action that the user can take is displayed in blue, for example inserting an Item into *In Tray*. This particular action requires the user to insert a description and add a Tag optionally. In turn, creating a new Tag requires the insertion of the description or name and an optional time and place.

Items can also be moved between lists without going through the Clarification process of answering popup questions. Because the meaning of Items can change

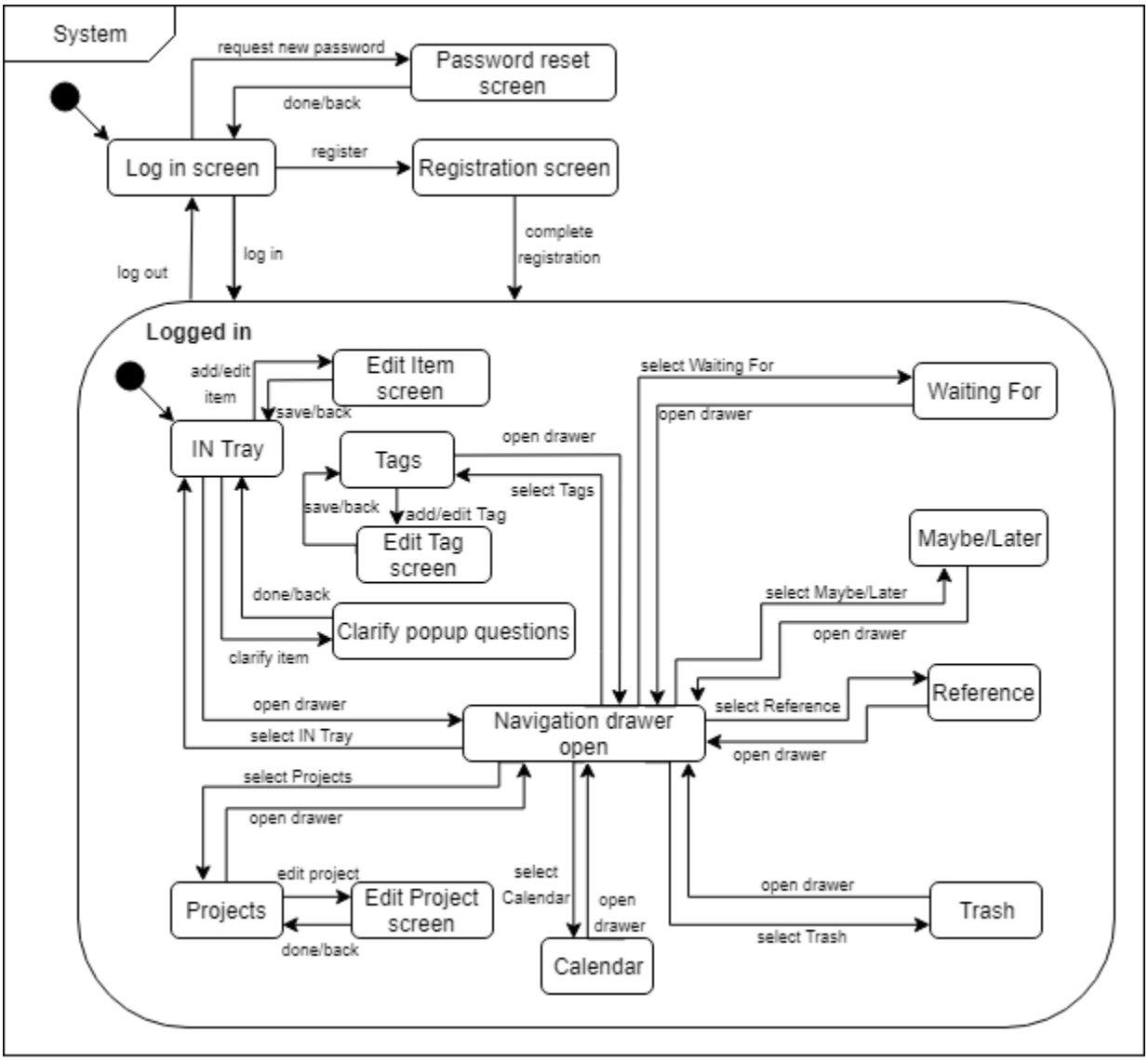
during time, it is appropriate that the lists of Items can adapt accordingly. For example, if the Item “Call Mark” is initially placed inside the Calendar, but Mark the user becomes aware of the fact that they need to wait to finish writing a report first, the Item can be moved directly into the Waiting For list. If, however, the user is unsure about the list in which the Item should be moved, it would still be possible to move it back to *In Tray* and follow the Clarification process from there.

It should be noted that there is no explicit action that the user can perform to create a Project, unlike deleting one. Instead, this happens during Clarification as the Item is Organised into the Projects list. Creating a new Project from the Projects list would mean that this Project contains no Items before Clarifying one to be Organised. This approach was preferred in order to avoid the accumulation of empty Projects in case the user forgets to create them.

The Reminder System service is in charge of sending push notifications to the user based on Calendar Items, Items with Tag with set Times and Weekly Reminders, as discussed earlier. Meanwhile, in order to remind the user of relevant Items, the Location Awareness service checks for the user’s location and sends notifications, if this location matches any of the Tags’ Location (and Time).

Finally, the Authentication service should handle account-specific actions such as logging in, logging out, registering, and requesting a new password. Users should be able to request a new password in case it is forgotten, in which case a warning email message should be sent to the registered address.

3.2.2 State machine



In order to avoid making the diagram too cluttered, actions from every list to the Edit Item screen have been removed. However, whenever an Item is shown in any list, the user can tap on it to edit its description and add or remove Tags. Every list is accessible by the navigation drawer, which is also available from any screen except when the user is editing an Item, a Tag, or a Project.

Overall, the state machine illustrates every possible state that the system can display to the user and the actions that trigger them. The Logged in state can only be reachable through the log in and registration screen, while *In Tray* is shown to be the landing page.

3.3 Design

Various design choices were made during the requirement gathering process, which excluded different approaches of the system. One of these regards the ability to insert Items directly into the lists. This feature was justified by looking at the usability of the application in the long term. Given that the clarify popup questions are always the same, the user will eventually understand the process of clarification and remember which Items go in which lists. It would therefore be cumbersome for the user to answer the same questions for each Item.

However, another solution was considered more compatible with the GTD workflow. The problem with allowing the user to insert Items into any list is that it encourages skipping the Clarification phase. Instead, users should be presented with the option of Clarifying before deciding to skip it. This approach ensures that the user is definitely sure that Clarification is unnecessary, and therefore reduces the chances that Items are placed in the wrong lists. In order to implement this solution, a Skip button was to be shown in the first Clarification popup, which would lead the user to select where to move the Item.

Another design choice to be made was one regarding the functionality of the Trash list. This list is intended by Allen to be a collection of Items that are determined to be not useful in the Clarification process. However, he describes two alternative ways of dealing with Items in this list: “When in doubt, throw it out” and “When in doubt, keep it” (Allen, 2015, p. 128). Since the system was going to adopt one approach only, and Items from the Trash could eventually be deleted, it was decided to keep Items in the Trash.

Furthermore, given that completing Items would result with their deletion from the list containing them, it made little sense to move deleted Items in the Trash. Otherwise, this list would end up containing both useless and completed Items, thus contradicting the list’s intended purpose. Instead, Items would be moved here only from Clarification.

The final issue of this list taken into consideration was what would happen when the Items were restored. Initially, the solution consisted of making the user Clarify the Item being restored just like from *In Tray*, so that it could be Organised effectively into the appropriate list. The main argument against this design is that it would create confusion to the user regarding when to expect Clarification popups. For example, if Items need to be clarified when being restored, what about when moving them from a list as well? For this reason, it was decided to restore Items directly into *In Tray*, from which they could then be clarified.

Another design aspect was the availability of the drawer menu in certain screens. In the current specifications, this menu is available in every state except for *Edit Item*, *Edit Tag*, and *Edit Project*. The purpose of hiding the drawer was a constraint meant to force the user to complete or discard the edit before visiting another screen. Otherwise the user could edit the description and some context tags and immediately go back to *In Tray* through the drawer menu, thinking that the Item was edited successfully. Instead, the user is meant either to save the edit or cancel it in order to receive feedback of the action from a toast message.

The placement of Tags was also an issue. One way of implementing the editing of tags was to include a Tags page in the drawer menu. Initially, this was discarded for two reasons. In the final specification, the menu contains only the core lists and calendar of the GTD methodology. Placing a Tags menu here would create confusion to the user, as it would be the only button that leads to a list that does not contain Items. Also, Tags are only relevant when inserting or editing an Item, as that is the moment when the user might want to edit or create Tags.

On the other hand, this design had a number of issues that were noteworthy to consider. For example, if the user intends to view the list of all the Tags that they have created, this approach would require them to find an Item in order to open the Edit Item screen. In this case, there is no need to edit an Item, but the system forces the user to do it anyway. Additionally, let us say that the user has no Items. To access Tags, the user is forced to create an Item in *In Tray*. Again, the system requires actions from the user which have no correlation with Tags.

After considering the pros and cons of both approaches, it was decided to implement a Tags page that could be accessible from the navigation drawer menu. This page was to be listed at the bottom of the menu in order to suggest that it is an additional list that is not part of *Getting Things Done* as explicitly as the others.

4. Implementation

The implementation of the specifications and design explained in the previous section took place mainly in *Android Studio*, using Java for the back-end and XML for the front-end of the system.

An incremental build model was adopted as a software development method. Each requirement was covered in order by implementing authentication and database integration first. As will be described later in this section, location-awareness was already set up at this point to be used later in development. *Getting Things Done* lists followed, after which came context Tags to attach to Items. Notifications were included in the last phase of development, as they depended on the previous implementation of features.

A number of external libraries were used in order to meet some non-functional requirements. These included the introductory tutorial and the calendar view in the Calendar list. The explanations for the use of external libraries will be included in the section below.

4.1 System functionalities

4.1.1 Database integration

The first functional requirement consisted of implementing user accounts, each of which could be able to store data in a cloud-based database. Firebase was chosen as a hosting service that would fulfill both requirements by managing authentication and the database (Firebase, 2011). There were several advantages of Firebase that distinguished it as the best option available for this project.

Firstly, the authentication service matched the relevant requirements exactly. It was possible to set up different types of authentication methods, including the use of Google and Facebook accounts to possibly be implemented at a later stage of development. The password reset service was also extremely intuitive to set up and manage. The following piece of code shows how simple it was for the system to make a call to the Firebase API reference to send an email to the user in order to reset the password.

```
firebaseAuth.sendPasswordResetEmail(email)
        .addOnCompleteListener(new OnCompleteListener<Void>() {
            @Override
            public void onComplete(@NonNull Task<Void> task) {
                if (task.isSuccessful()) {
                    Toast.makeText(ForgotPasswordActivity.this,     R.string.email_sent,
Toast.LENGTH_SHORT).show();
                }
            }
        });
```

Sending a password reset email to the user

Another advantage of using Firebase was the use of the console when editing entries in the real time database. For example, during the development of lists of Items, it was possible to insert data in the lists from the console before implementing the insertion or movement of Items. This was particularly useful when testing adapters in order to ensure a correct behaviour before moving on to the implementation of the Clarification process.

Moreover, it is important to consider the fact that API calls that are used to read data from the database are fully asynchronous. Calls had to be made using listeners to the database references that were triggered once for the initial state and whenever the dataset changed. Compared to a synchronous approach, where data is read whenever the execution of the code reaches the call, this offered both advantages and disadvantages.

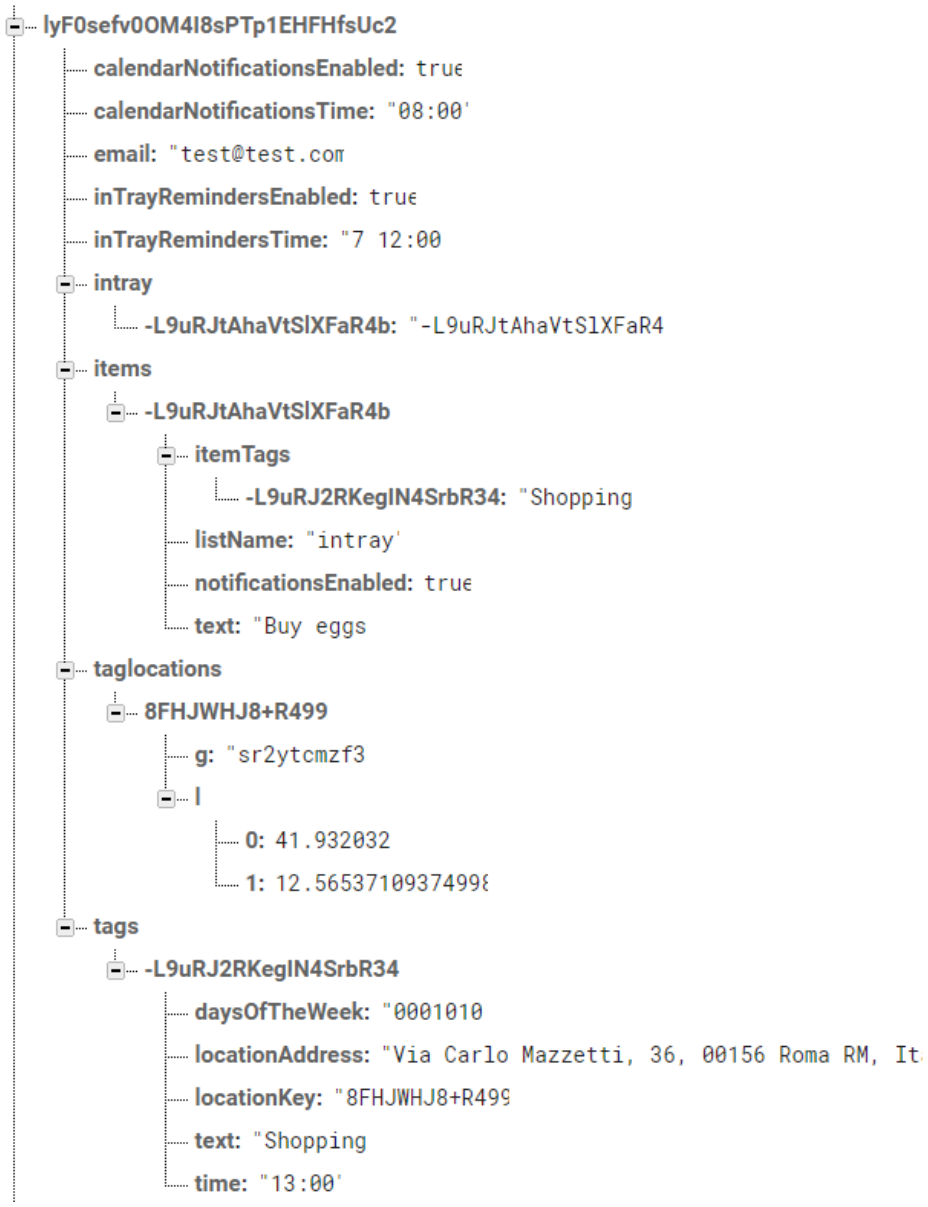
```
databaseReference.child("users").child(firebaseAuth.getCurrentUser().getUid()).child("
intray").addChildEventListener(new ChildEventListener() {

   @Override
   public void onChildAdded(DataSnapshot dataSnapshot, String s) {

       for (int i = 0; i < MainFragmentActivity.getItems().size(); ++i) {
           Item item = MainFragmentActivity.getItems().get(i);

           if (dataSnapshot.getKey().equals(item.getKey())) {

               inTrayFragment.getEmptyInTrayText().setVisibility(View.GONE);
               inTrayFragment.getProgressBar().setVisibility(View.GONE);

               items.add(item);
               break;
           }
       }

       notifyDataSetChanged();
   }
```

Example of an asynchronous API call that reads In Tray Items from the database and notifies the list adapter

By letting the API choose when to trigger the reading of the initial state, the system removes the possibility from the main thread to freeze the whole application when reading a large amount of data. The blocking of the main thread made from a synchronous call would be detrimental to the user experience and would therefore require a management of threads that could include potential mistakes.

However, the restriction of control over when to trigger the retrieval of data from the database made the implementation of user interface elements such as the progress bar when loading Items more challenging. Another example is the case where one call should be made before a second, meaning that the order of calls is crucial to control. While it would be possible to nest a call within another call, a preferable solution was to take advantage of the fact that some types of listeners are always called after other types. This approach was used when all the Items from every list needed to be read and stored in the `MainFragmentActivity` class before reading and displaying *In Tray* Items.

Regarding the structure of the database, guidelines set by Firebase were followed. The major concern in this phase was to avoid nesting data wherever possible as fetching data at a location also retrieves all of its child nodes. In order to flatten the data structure, dataset keys were used as entries in many occasions.



Example of data flattening in the Firebase database

As shown in the screenshot of the database structure above, Items are kept under the “items” child regardless of the list that contains them. These are retrieved and stored in the items ArrayList in the `MainFragmentActivity`. When the system makes an API call to read *In Tray*, it retrieves the keys of the Items that are stored under the “in Tray” child. These keys are then checked against those of the Items in the `ArrayList` in order to reference the data that the relevant Items contain. The same happens with Tags, which are stored under the “tags” child and whose keys are referenced under “itemTags” in each Item.

If Items and Tags were not kept in separate children, but instead in the lists that contained them, whenever an Item would be read by an API call, every Tag that it contained would be read as well. This nesting was completely avoided with this data structure that kept their keys as reference.

4.1.2 Location awareness

When implementing location awareness, the initial method considered was to retrieve Tag locations manually from the Firebase database in order to check periodically if the user's current location matched one or more of them. This would involve calculating the distance between the user and each location and setting a maximum radius which would determine if the user was close to the location or not.

However, while the periodic check of the user's current location was necessary, the retrieval of Tag locations could be removed by using the set of open-source libraries that Geofire offered (GeoFire, 2014). By "selectively loading only the data near certain locations", this solution would be extremely beneficial in case the user stores a large number of Tag locations that would otherwise increase the load on the main thread every time the location of the user was checked.

Another advantage of using Geofire was the fact that it was designed to be used in conjunction with Firebase by storing data in its own format in the database under the "taglocations" child in the database structure shown beforehand. The implementation of Geofire in the system consisted of two steps. The first one was creating a new location from the Geofire reference (namely the database reference of the "taglocations" child) by setting the ID, a `GeoLocation` including latitude and longitude, and a `CompletionListener`. This procedure is to be called every time a new location is included in a Tag by the user.

```
geoFire.setLocation(place.getId(), new GeoLocation(place.getLatLng().latitude,
place.getLatLng().longitude), new GeoFire.CompletionListener() {
   @Override
   public void onComplete(String key, DatabaseError error) {
       Toast.makeText(TagActivity.this,      "Location      stored      successfully",
Toast.LENGTH_SHORT).show();
   }
});
```

Creating a new location with GeoFire

The second step was to create a query that would return all the locations near to the user. This location was to be checked every set interval in order to keep the system updated of where the user was located. In order to request the location of the user, the

LocationRequest API from Google was used (Android Developers, 2017). The API would first require the user to grant permission to track their location while both the interval and the priority of the request had to be set according to the system's needs.

The main concern of the use of this API was the impact that frequent requests could have on the battery of the smartphone. Therefore, a balance had to be found between battery drain and accurate location updates. A solution was found that was appropriate to the needs of the system. This consisted of using the `PRIORITY_BALANCED_POWER_ACCURACY`, setting the "fastest" interval to 1 minute and the normal interval to 5 minutes.

```
LocationRequest locationRequest = new LocationRequest();
locationRequest.setPriority(LocationRequest.PRIORITY_BALANCED_POWER_ACCURACY);
locationRequest.setFastestInterval(60000);
locationRequest.setInterval(300000);

if (ContextCompat.checkSelfPermission(mainFragmentActivity,
android.Manifest.permission.ACCESS_FINE_LOCATION ) ==
PackageManager.PERMISSION_GRANTED) {

	mainFragmentActivity.getFusedLocationClient().requestLocationUpdates(locationRe
	quest, new LocationCallback() {
		@Override
		public void onLocationResult(LocationResult locationResult) {
			if (locationResult == null) {
				return;
			}
			for (Location location : locationResult.getLocations()) {
				setUpTagLocationNotifications(location);
			}
		}
	}, null);
}
```

Setting the priority and interval of location requests

Regarding the priority, this would ensure that the location accuracy was about 100 metres. More accurate priorities were not necessary, as these are usually used by systems that display the user's location on a map. This system would not require a high level of accuracy, but rather determine whether the user is close to a general location such as home, the supermarket, or the office.

On the other hand, two intervals were used in order to fully take advantage of the API's functionalities. The "fastest" interval was to be the fastest rate at which the application would receive location updates. However, unlike with the normal interval, these updates could be triggered by other applications that were requesting the user's location. This would not make a new request, but rather use those that were made by other applications on the user's smartphone.

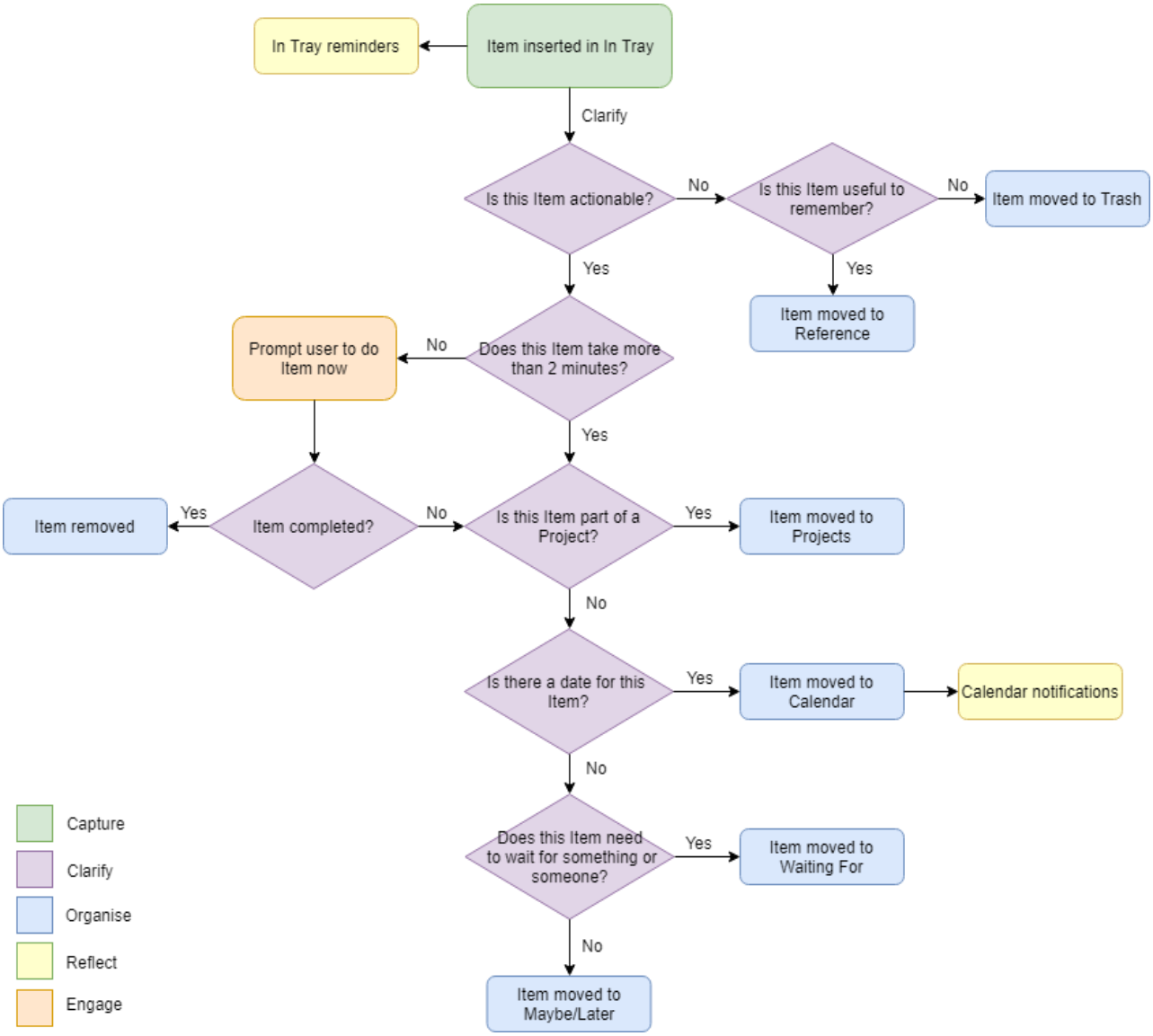
Meanwhile, the default interval was set to 5 minutes. A smaller interval would have been preferable but not necessary, since users were expected to stay at least 5 minutes in locations used by Tags. Nevertheless, the fact that the fastest interval was much lower meant that the average of time between each request was less than 5 minutes.

```
GeoQuery geoQuery = geoFire.queryAtLocation(new
GeoLocation(currentLocation.getLatitude(), currentLocation.getLongitude()), 0.2);

geoQuery.addGeoQueryEventListener(new GeoQueryEventListener() {
	@Override
	public void onKeyEntered(final String key, GeoLocation location) {

		databaseReference.child("users").child(firebaseAuth.getCurrentUser().getU
		id()).child("tags").addListenerForSingleValueEvent(new
		ValueEventListener() {
			.
			.
			.
```

Creation of a GeoFire query based on the user's current location

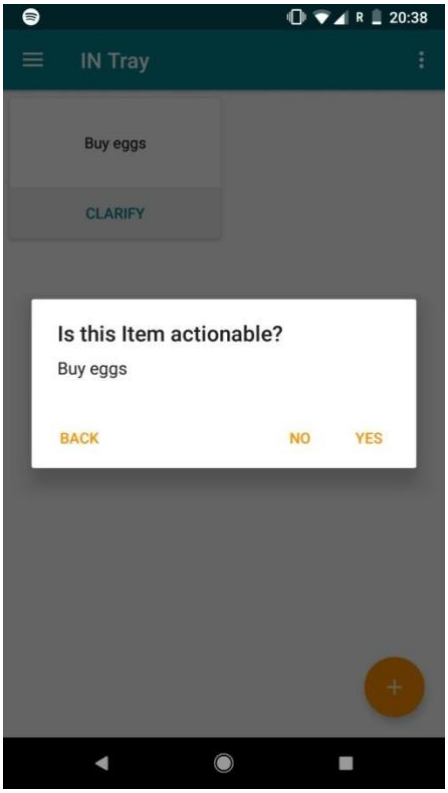
4.1.3 Clarification process

The Clarification process was implemented with popup dialog questions as specified by the non-functional requirements of the system. The flowchart below illustrates the questions asked to the user and the Organisation of the Items to which they lead. Each phase of the Five Phases of the GTD workflow are represented by a different colour.



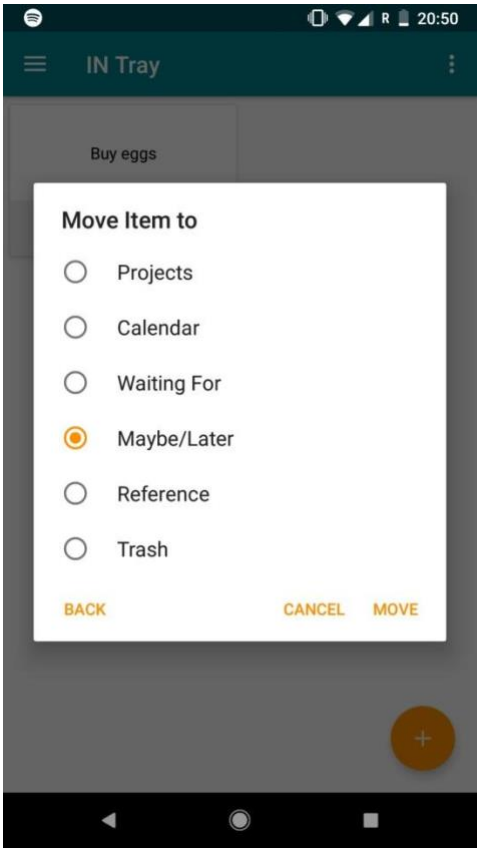
Clarification process implemented in the system

The process was developed using `AlertDialog` that could display at most 3 buttons, which, in most cases, included a *Yes, No,* and *Back* option. By including a *Back* button, the system covers the case in which the user presses *Yes* or *No* by mistake, or is unhappy about the list that the process indicated to contain the Item.



Example of a Clarification question popup

Note that the text of the Item being Clarified was included in every `AlertDialog` so that each question being asked could relate to the Item in a more clear fashion. Before Organising the Item into the appropriate list, the user is told beforehand with an additional popup that can be confirmed to complete the Organisation of the Item. A *Skip* button was also included in the first `AlertDialog` so that the user can move the Item directly into a list.



Popup opened when the user skips the Clarification process

Each `AlertDialog` is created by separate methods that set the neutral, negative, and positive buttons (from left to right) and are encapsulated in the Clarification class that is instantiated within the InTrayAdapter class when an Item's Clarify button is pressed. The decision to place these methods inside a separate class was that, since each `AlertDialog` required a separate method to handle each action performed by the user, the clarity of the `InTrayAdapter` class would suffer if the code was placed here.

```
public void showClarifyDialog() {

   AlertDialog.Builder builder = new
AlertDialog.Builder(inTrayFragment.getActivity());
   String itemText = item.getText();
   builder.setTitle(inTrayFragment.getResources().getString(R.string.clarify_item));
   builder.setMessage(itemText);

   builder.setNeutralButton(R.string.skip, new DialogInterface.OnClickListener() {
       @Override
       public void onClick(DialogInterface dialogInterface, int i) {
           showSkipDialog();
       }
   });

   builder.setNegativeButton(R.string.cancel, new DialogInterface.OnClickListener() {
       @Override
       public void onClick(DialogInterface dialogInterface, int i) {}
   });

   builder.setPositiveButton(R.string.clarify, new DialogInterface.OnClickListener() {
       @Override
       public void onClick(DialogInterface dialogInterface, int i) {
           isThisItemActionablePopup();
       }
   });

   AlertDialog dialog = builder.create();
   dialog.show();
}
```

Creation of the initial Clarification popup

4.1.4 Notifications

Tag notifications were, perhaps, the most challenging requirement to implement because of the different conditions that had to be met in order to trigger them in the correct scenarios. On the other hand, *In Tray* and *Calendar* notification reminders were to trigger at a set time, meaning that the system could implement the use of repeating `AlarmManager` every week or every day. For example, the `MainFragmentActivity` calls the `setUpInTrayReminders` method which makes a Firebase call to read from the database the time and day set for this reminder. This data, converted in milliseconds, is then used to set the repeating of a new `AlarmManager` to trigger every 7 days at the specific time set by the user.

```
alarmManager.setInexactRepeating(AlarmManager.RTC_WAKEUP, triggerAtMillis,
AlarmManager.INTERVAL_DAY * 7, pendingIntent);
```

Setting the repeating interval of the AlarmManager in charge of sending weekly notifications

Subsequently, *In Tray* reminder `BroadcastReceiver` receives the alarm and checks the database for Items that are contained in *In Tray* in order to include their text in the notification sent.

While a similar approach was used for Calendar reminders, notifications for Tags required more effort to implement. Tags should trigger notifications if:

- Time matches current time and neither day of the week or location are set
- Location matches current location and neither time or days of the week are set
- Time and current day match current, but location is not set
- Time and location match current, but days of the week are not set
- Time, day of the week and location all match current

The handling of these checks were placed inside a separate `TagsNotificationManager` class that would create the appropriate `AlarmManager` to send alarms to the `TagsNotificationReceiver BroadcastReceiver`.

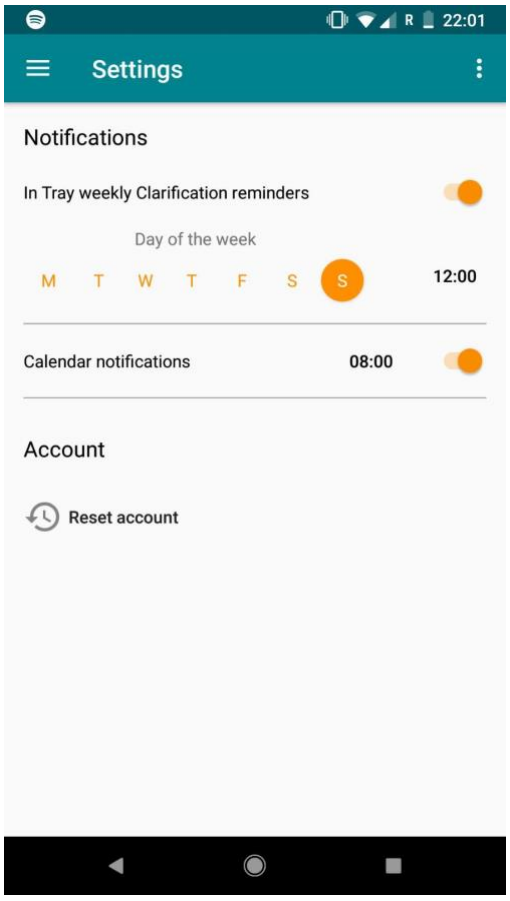
The first method inside the `TagsNotificationManager` class, `startLocationUpdates`, concerns the request of location updates discussed previously. The current location is then passed to the `setUpTagLocationNotifications` method, in order to set up a GeoFire location query. In case of a Tag location matching the current user's, the `setUpAlarmManager` method is called with the relevant Item and Tag as parameters.

On the other hand, the `setUpNonLocationNotifications` method also calls the same `setUpAlarmManager`, but without making a GeoFire query. Instead, this method checks every Tag whose location is not set and the Items to which it is attached.

Finally, the `setUpAlarmManager` reads the time and days of the week from the Tag and passes it as milliseconds to a new, repeating `AlarmManager`. However, because `AlarmManager`s repeat indefinitely, those set by the `setUpTagLocationNotifications` method are removed in the `onKeyExited` method of the GeoFire query.

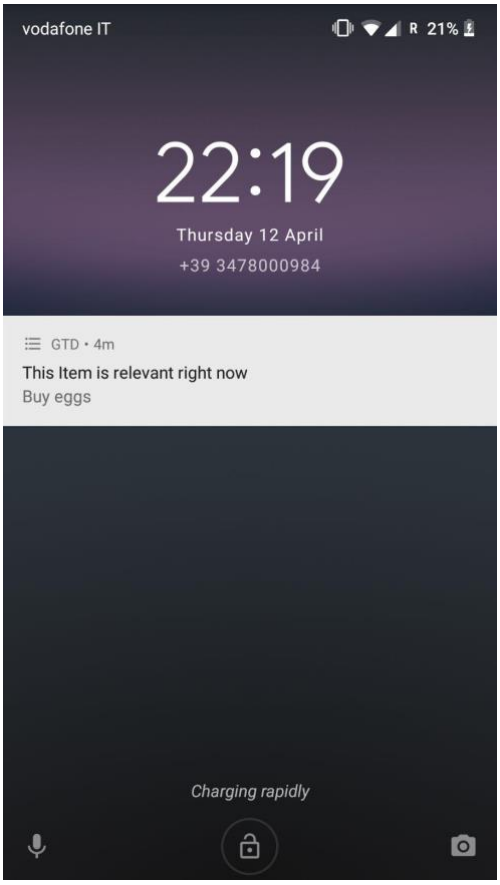
For example, if the user sets a Tag to trigger at home every working day of the week at 12:00 pm and stays at home for two days in a row from Monday to Tuesday, a single repeating `AlarmManager` is used. However, as soon as the user leaves the house, the `AlarmManager` is canceled so that the Tag does not send notifications.

Finally, user control over notifications was included in a separate Settings `Fragment` accessible from both the navigation drawer menu and the options menu on the top right corner of the `MainFragmentActivity`.



Settings page giving the user control over notifications

As the user can select their preference over if and when to receive *In Tray* and *Calendar* notifications, this data is stored in the Firebase database. Regarding Tag notifications, it was decided to give more choice to the user on which Tags is able to send notifications. This was because some Tags could simply be used as a way to display context of an Item. Despite having set a time and a day of the week, the user might not want notifications to trigger. Therefore, a notification switch was added to each Tag with its value to be stored in the database.



Example of a notification displaying the Item whose Tag is currently relevant

4.1.5 Material design

Material Design guidelines were followed when creating layouts for activities and fragments. Starting from the color palette, which was picked using the Material Color Tool, it was crucial to pick a secondary colour that was legible on the light grey background that was used in the majority of times. Both primary and secondary colours, turquoise and orange, were selected from the Material palette. This implementation was justified by research that found evidence of the blue colour reducing stress and anxiety, which are to be kept low in a productive environment (Kwallek, Lewis and Robbins, 1988).

Dimensions were also kept in line with Material metrics, such as paddings of layouts and the `CardView`s used to display Items. These generally equated to 16dp, a value that was saved in the `dimens` file inside the values folder in order to be reused for every layout.

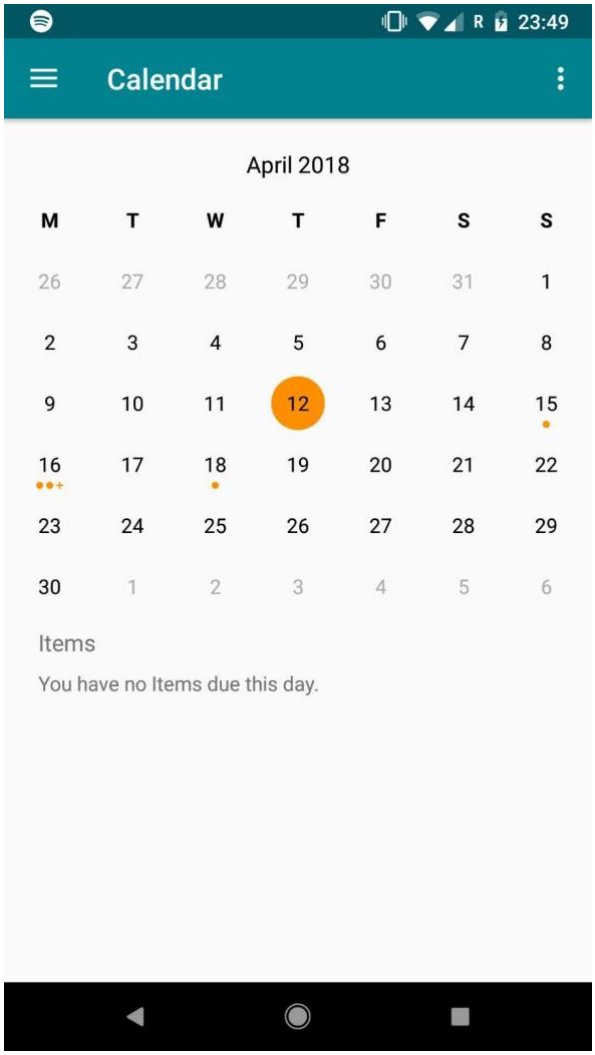
In a similar fashion, text size adhered to Material conventions with the additional implementation of using scale-independent pixels rather than density-independent pixels as units. By using this unit, the text can be scaled to the user's font size preference, meeting the needs of users with myopia.

4.2 External libraries

Additional external libraries were implemented and are visible in the project-level `build.gradle` file. These consist of CompactCalendarView by SundeepK and AppIntro by APL Developers (CompactCalendarView, n.d.) (AppIntro, n.d.). Although the use of external libraries was avoided when possible, these libraries were implemented as they met some requirements explicitly by providing features that were considered essential.

After inserting an Item into the Calendar, the system should visualise in some way that some days contain Items. This way, the user is not forced to select day after day in order to know which Items are due on which dates. Unfortunately, the `CalendarView` from the `android.widget` widget package (which was used in the Clarification process) does not provide such a feature.

On the other hand, CompactCalendarView not only displays "events" for each day, but also their number. This is obtained by showing dots below the day of the month.



Example of the CompactCalendarView displaying dots for Items in the Calendar list

In the screenshot above, the current day is selected and shown with a circle using the secondary colour. As there is an Item on the 15th, 3 on the 16th, and 1 on the 18th, a proportional amount of dots is displayed. These dots enhance the usability of the system by communicating the presence of Items on certain dates.

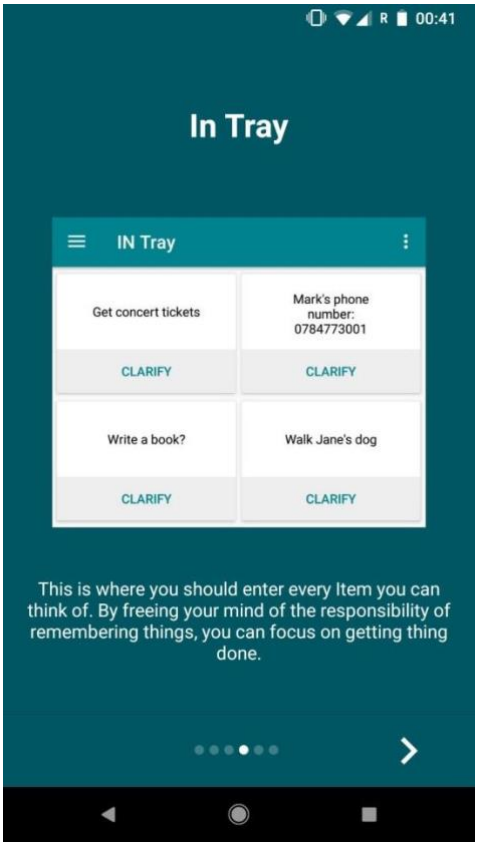
The implementation of *CompactCalendarView* posed some initial issues due to the fact that Items had to be converted to “events” in order to show as dots. This was solved by creating an event each time an Item was inserted or moved into the *Calendar* and removing it whenever the Item was either deleted or moved.

The second external library, AppIntro, was used to implement the introductory tutorial that was mentioned in the requirement gathering phase of the project. By using the `addSlide` method, the library required only one class,

`IntroductionActivity`, to be created. An alternative method of creating the tutorial would have consisted of one activity containing a number of fragments equal to the number of slides to show. Besides increasing the number of classes and the size of the project, fragments would also have been slower to run, as the `FragmentManager` would have had to begin and perform a new `FragmentTransition` each time the user swiped from one fragment to the other.

```
addSlide(AppIntro2Fragment.newInstance(getResources().getString(R.string.first_intro_fragment_title),
        getResources().getString(R.string.first_intro_fragment_description),
        R.drawable.ic_launcher_foreground_for_intro,
        getResources().getColor(R.color.colorPrimaryDark)));
```

Adding a slide to the IntroductionActivity class



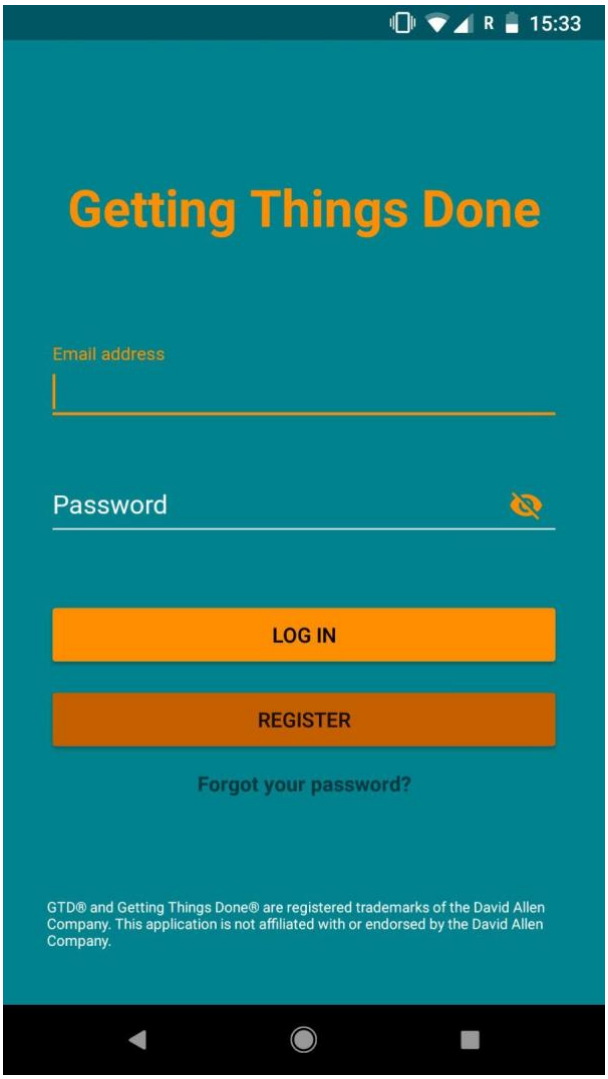
Slide explaining the functionality of *In Tray* in the introductory tutorial

5. Awareness of professional issues

The standards set by the British Computer Society Code of Conduct were respected in order to address legal, social, ethical, and professional issues (British Computer Society, 2015).

5.1 Legal issues

Since the application developed makes use of concepts from *Getting Things Done*, credit was to be given to the David Allen Company, which holds the registered trademarks of terms like “GTD”. Moreover, the system was not affiliated with or endorsed by the Company, which had to be stated for legal purposes. The disclaimer is displayed in the log in page, which is also the landing screen of the application. This way, the user is aware that the concepts used by the application belong to David Allen and that the system is not affiliated with his company before using the application for the first time.



Landing page of the application displaying the copyright disclaimer at the bottom of the layout

5.2 Social and ethical issues

The British Computer Society states in its Code of Conduct that “you shall have due regard for public health, privacy, security and wellbeing of others and the environment.” (British Computer Society, 2015) Concerning the context of this project, the main relevant issue was maintaining the privacy of the user.

Regarding the user-submitted data stored inside the Firebase database, this can include sensitive information such as credit card or personal details and should therefore be only accessible to the user that submitted it. Consequently, the following security rules were set up for the database in order to restrict read and write access.

```
{
	"rules": {
		"users": {
			"$uid": {
				".read": "$uid === auth.uid",
				".write": "$uid === auth.uid"
			}
		 }
	}
}
```

Firebase database security rules included in the google-services.json file

By following these security rules, data of each user can only be read and written by an authenticated user, who has an ID equal to the key of the user child node. In other words, each authenticated users can read and write the own data without the possibility of a third party having access to it.

Another social and ethical issue was the one concerning location requests. The privacy of the user might be compromised if the system did not handle this type of information transparently and with care.

Firstly, the application asks for the user's permission to track their location. If this permission is not granted, the system will never check for the user's location. Secondly, permission is only asked after the introductory tutorial which explains the functionality of Tags and, therefore, the reason for this permission. Lastly, the history of the user's location is never stored, either locally or on the database. In fact, the current location is not even stored as a variable inside the `TagsNotificationManager`, but instead passed as a parameter to the `setUpTagLocationNotifications` method as soon as it is retrieved.

Finally, usability testing required the approval of the College Research Ethics Committees. The approval was obtained after submitting a Minimal Ethical Risk Registration Form (see Appendix C). This ensured that the research made to design and evaluate the usability of the system complied with ethical clearance.

When recruiting participants, the Information Sheet (see Appendix D) was given in order to provide sufficient information to make an informed decision about whether taking part (King's College London, n.d.). This document stated that data was to be kept confidential and only to be used for the purpose of this project. Additionally, data

collected was to be completely anonymous to preserve privacy and would not include names, age, or any information that could be used to recognise the identity of participants. A Consent Form (see Appendix E) was also given to participants in order to allow them to express their informed consent after reading the Information Sheet.

5.3 Professional Issues

The first point of regarding Professional Competence and Integrity of the British Computer Society Code of Conduct states that “you shall only undertake to do work or provide a service that is within your professional competence.” (British Computer Society, 2015) In order to be able to undertake the development of a mobile application for Android, this requirement was met by conducting research in order to learn concepts regarding the Android platform.

These included the behaviour of activities, fragments, adapters, receivers, as well as the different features of layouts and widgets to be used in the user interface. Overall, this process was critical in the acquisition of professional competence required by the objectives of the project.

Another point which was thoroughly met by this project is the one concerning respecting and valuing alternative viewpoints and seeking, accepting and offering honest criticisms of work. By using a design-centred approach with usability testing and questionnaires, criticisms of work were not only accepted, but played an integral part in the development of the final system. Requirements were adjusted to the results of this phase so that user needs were met as rigorously as possible.

6. Evaluation

The final mobile application was evaluated against the requirements of the system after conducting a second phase of usability testing. Despite the fact that every requirement was met, some aspects are noteworthy to discuss in order to establish strong and weak areas of the implementation.

6.1 Final implementation of usability testing

Participants of the usability testing used in the requirement gathering phase of the project were invited to install the final version of the application on their own smartphones at the end of development. After a regular use of the system lasting about one week, a questionnaire was to be answered, including the same System Usability Scale (SUS) used in the first questionnaire plus other system-specific questions (see Appendix H).

This particular approach to usability testing, called in-situ, was preferred over a laboratory-based one for various reasons. Firstly, it removes the influence of the researcher over the behaviour of participants so that they have complete freedom over the interaction with the system. This, combined with the absence of specific tasks to perform, could produce data that might be extrapolated to a widespread use of the application. Therefore, the system was to be evaluated within the same environment where it would exist when deployed

Moreover, the usefulness of a core system functionality, notifications, could not possibly be tested in a laboratory setting. *Context*, such as *time* and *location*, was a variable which had to change during the usability testing of the system in order to send reminders to the user. Therefore, the participants were instructed to use the application as they would actually do in real life, so that the questionnaire could produce data applicable to a real world scenario as much as possible.

Since it would be useful to compare the final implementation with the prototype developed initially, the questionnaire included a SUS. On the other hand, the second set of questions aimed at producing data that could be used to evaluate the effectiveness of functionalities, problems with design choices, and overall impressions.

Results from the questionnaire showed that the final implementation scored 90.25 on the SUS. Compared to the prototype, which had a score of 81.25, it can be

stated that the changes made to the initial requirements were effective in increasing the usability of the system. However, it is also true that the set of participants was the same as that of the prototype usability testing. Therefore, the final SUS score should be considered slightly inflated as this was not the first time that the participants interacted with the application.

Moving to answers regarding system functionalities, the first interesting result concerns the Clarification process, whose implementation was the most important functional requirement of the application. Results indicate that 70 per cent of participants found the process to be extremely intuitive, which should be considered a significant success. The remaining 30 per cent chose the penultimate option, demonstrating an overall satisfaction with the implementation of popup questions.

The three questions that followed concerned the usefulness of the three categories of notifications: *In Tray Weekly Review* reminders, *Calendar* notifications and *Tag* notifications. *In Tray Weekly Review* reminders proved to be the less popular feature of the three, with an average score of 3.6 out of 5. This could be explained by the fact that, perhaps, users did not keep Items in *In Tray* for a long period of time, meaning that the *Weekly Review* reminder did not even trigger. Another explanation could be that, while users left various Items in *In Tray*, the reminder did not push them to begin the Clarification process.

The *Weekly Review* is considered to be a crucial step to “keep the mind from taking back the job of remembering and reminding” (Allen, 2015, p. 50). Therefore, it was disappointing to witness the failure to convey this importance to the user. In order to increase the efficacy of these reminders, it could be possible to include a mention of the *Weekly Review* in the introductory tutorial as part of the main concepts explained.

Another possible solution would be the implementation of timers for each *In Tray* Items. After an Item has been inserted into the In Tray, the timer would ensure that, after a week, it is moved into the *Trash* list. Reminders could be sent to the user that a specific Item is expiring and needs to be Clarified. However, this feature should be tested with users beforehand to gather enough evidence that users are not annoyed or anxious of expiration timers. Moreover, it is likely that some users would *Clarify* Items into the wrong list just for the sake of keeping them away from the *Trash*.

Conversely, *Calendar* and *Tag* notifications were generally considered useful by the participants with respective scores of 4.7 and 4.3 out of 5. Furthermore, since the question asks whether *Tag* notifications were useful, the positive answers imply that Tags themselves were used often by participants. The implementation of context

awareness was identified as the major extension to the system's basic requirements, and this result justified it. Exhaustive control over notifications was also achieved with an average of 90 per cent of participants stating that they had enough control.

The results indicate that the introductory tutorial was also evaluated positively. However, participants considered the tutorial to be more effective in explaining the functionality of the application rather than the basics of *Getting Things Done*. Indeed, effort was put more on the explanation of the system as the number of total slides was kept low. This was to avoid the case where the user does not want to read too much text and skips the whole introduction. In order to explain the concepts of *Getting Things Done* more thoroughly, an "about" page could have been included in the navigation drawer menu.

A number of design problems emerged. One particular answer raises an issue that was not taken into consideration when designing the system. The answer states that the participant would have liked to see all their Items in one place. The solution was implemented for Tags, but not for Items. Instead, displaying every Item in one page would allow the user to see Items that might otherwise be hidden away in the Calendar. Unfortunately, this scenario was not considered until the results of this questionnaire were obtained.

As the scope of the questions widens, a more general evaluation of the project can be obtained. For example, the statement "I would still prefer a pen-and-paper approach to the *Getting Things Done* methodology", received the average response of 1.5 (where 1 is "not at all" and 5 "a lot"). This result suggests that the objective of the project was satisfied as users approved of the system's particular implementation of Allen's methodology and preferred it to the original approach.

The overall impressions of the system from participants were positive, with the majority of comments relating to the usability of the user interface. "Simple" and "easy to use" appeared frequently, meaning that users did not find it difficult to deal with *Getting Things Done* concepts. As the average initial familiarity of the participants with Allen's methodology was 2.2 out of 5, the high level of usability is particularly significant. Despite the fact that the average user did not know about the methodology very well, the system resulted approachable and simple to use.

6.2 Limitations

The application developed in this project has various limitations, some of which derive from the design choices and some from the scope of the system. For example, the application requires an internet connection in order to load *Items*, *Projects,* and *Tags* from the cloud-based database and perform tasks with this data. In case of an absent connection, progress icons keep spinning until the user connects to the internet. This limitation excludes the possibility of using the system in certain environments such as the tube or a plane.

Another limitation is the fact that each Item can only be contained by one list at a time. For example, it would not be possible to give an Item in a Project a due date, as this would require the user to move the Item into the *Calendar*. Similarly, a *Reference Item* such as a phone number could be useful to the user on a particular date, in which case the Item should be moved into the *Calendar*.

Regarding Items in the *Waiting For* list, there is no way for the user to insert who or what the Item should wait for. Instead, the user is expected to enter this information as part of the Item's description. However, it would be useful to categorise *Waiting For* Items in such a way that the user knows immediately whether an Item waiting condition has been satisfied or not.

There is also no way for users to delete the entirety of their data from the database. Instead, the account can be reset in the *Settings* page so that the user's data is reset to where it was immediately after registration. While this removes everything except the user's registered email and default notification settings, the only way to completely remove an account and its data is to do it from the Firebase console.

Finally, another major limitation is that the application is only accessible to users who know English. Every string used in the user interface was stored as a variable in the `strings` file to allow possible translations into other languages, but the final version did not implement these functions.

6.3 Overall

Overall, every requirement was met in the development phase successfully. Some implementations were evaluated to be more effective than others, which could have

taken place using different design approaches. Nevertheless, compared to the results expected from theory, those obtained in practice were impressively similar.

Getting Things Done lists were implemented to consist of a collection of Items which are Clarified from *In Tray*. This Clarification process is performed by the user while answering popup questions to guide them through the GTD workflow. Notifications also behave as expected, triggering for the *Weekly Review*, *Calendar* Items, and *Tag* contexts.

An introductory tutorial was included to remove the user's barrier of entry to the system and *Getting Things Done* concepts. While this was more successful in explaining the functionality of the system, the overall efficacy was to be considered satisfactory.

User accounts and a cloud-based database were also met, and their implementation was not cause for any concerns raised in usability testing. Users can register, log in and out, as was expected in the design phase of the project, while their data is stored securely on Firebase.

Meanwhile, *Material Design* guidelines were followed to the letter, producing a user interface that was welcomed by testing participants and that was almost identical to that of Android system applications.

The main objective of the project was to create an Android application that fully automates the *Organise* phase of the *Five Phases* of Project Planning, while centralising resources and providing location-based notifications and action reminders. An evaluation of the final artefact reveals that this objective was met by implementing every requirement that was gathered during the earlier phases of this project.

7. Conclusion

This project developed a mobile application that enhances the productivity of the user by making use of the *Getting Things Done* methodology and context awareness.

Research made during the literary review and the analysis of existing work influenced the requirement gathering process greatly by determining the functionalities to be included in the final system. Likewise, the usability testing of an initial prototype raised issues during development by providing data submitted by participants.

While the development of the application was carried out successfully regarding the requirements gathered, the evaluation of the final system identified both strong and weak areas. Nevertheless, it is possible to envision results that would be obtained when applying the present work to a widespread use.

As Items can be used to represent a large variety of ideas, things to do or to remember, the system can adapt to many different scenarios and uses. It is possible to imagine the list of users to include students, doctors, businessmen, caretakers, and any other individuals who benefit from an enhanced productivity. The personalisation of Tags and context creation allows the system to be applied in any kind of environment while providing notifications reminding users of tasks to perform.

As for possible future works, these should be focused on resolving limitations brought to light by the evaluation, as well as extending the scope of the project by implementing additional functionalities. Offline access to the entirety of the system should be considered, similarly to the translation of strings into different languages.

Regarding offline access, this would require a copy of the cloud-based database to reside locally in the storage of the smartphone. Then, whenever the user goes back online, the Firebase database would be updated with the differences between the two data sets.

On the other hand, the translation into other languages should take advantage of the fact that every string used was stored inside the `strings` file. Each of these can be translated and placed in a separate file for each language.

Future works could also include the development of clients for different platforms, such as a desktop or iOS application. These should connect to the same cloud-based

database in order to allow users to log in and access the same data across different devices.

A final note should be made concerning the technological innovation that will take place in the field of smartphones and the impact that it will have in the maintenance of the system developed.

As smartphone battery life increases, whether in terms of capacity or type, the relative power requested by location updates will decrease. This will allow the application to request more accurate updates and at a shorter time interval. Meanwhile, as digital assistants improve significantly over the next years, the application should take advantage of voice commands in order to perform tasks.

8. Definitions

GTD - Getting Things Done

Item - Something to do or to remember

In Tray - List containing non-clarified Items

GTD workflow - The five main phases of Capturing, Clarifying, Organising, Reflecting and Engaging

Capturing - Inserting Items into the In Tray

Clarifying - Identifying in which list In Tray Items belong

Organising - Moving an Item from the In Tray into the list it belongs

Reflecting - Reviewing the In Tray and other lists

Engaging - Performing an actionable Item

Activity - A window in which to place a user interface

Fragment - A sub-activity that can be re-used in different activities. An activity can contain many of these

Adapter - Service in charge of representing data in an Activity or a Fragment as a list

AlertDialog - Popup displaying a title, a message and buttons

AlarmManager - Service to run at specific time intervals to trigger a BroadcastReceiver

BroadcastReceiver - Component in charge of managing events sent by AlarmManagers

9. References

Allen, D. (2015). *Getting Things Done*. 2nd ed. London, United Kingdom: Piatkus.

Android Developers. (2018). *Android Developers*. [online] Available at: https://developer.android.com [Accessed 2 Dec. 2017].

Android Studio. (2017). Google.

Balsamiq. (2008). Balsamiq Studios.

British Computer Society. (2015). *BSC Code of conduct*. [online] Available at: http://www.bcs.org/category/6030 [Accessed 3 Jan. 2017].

Brooke, J. (1996). Usability Evaluation in Industry. 1st ed. Boca Raton, Florida: CRC Press, pp.189-194.

Caplan, J. (2007). The Five Secrets to Getting Things Done. *Time*. [online] Available at: http://content.time.com/time/business/article/0,8599,1595805,00.html [Accessed 1 Dec. 2017].

Chaos Control: GTD Organizer & Task List Manager. (2012). Chaos Control.

DGT GTD & To-Do List [Alpha]. (2010). Viganello, Switzerland: dgtale.

Facebook. (2004). *Facebook*. [online] Available at: https://www.facebook.com/ [Accessed 9 Dec. 2017].

Firebase. (2011). *Firebase*. [online] Available at: https://firebase.google.com/ [Accessed 10 Dec. 2017].

GitHub. (n.d.). *AppIntro*. [online] Available at: https://github.com/apl-devs/AppIntro [Accessed 15 Mar. 2018].

GitHub. (n.d.). *CompactCalendarView*. [online] Available at: https://github.com/SundeepK/CompactCalendarView [Accessed 12 Feb. 2018].

GitHub. (2014). *Geofire*. [online] Available at: https://github.com/firebase/geofire [Accessed 14 Dec. 2017].

Google Keep. (2013). Google.

Google Play. (2017). *Google Play Store*. [online] Available at: https://play.google.com/store [Accessed 4 Dec. 2017].

Info.localytics.com. (2017). *24% of Users Abandon an App After One Use*. [online] Available at: http://info.localytics.com/blog/24-of-users-abandon-an-app-after-one-use [Accessed 6 Feb. 2018].

King's College London. (n.d.). *Preparing an information sheet for participants*. [online] Available at: https://www.kcl.ac.uk/innovation/research/support/ethics/Applications/Recruitment-Documents/infosheet.aspx [Accessed 3 Dec. 2017].

Kwallek, N., Lewis, C. and Robbins, A. (1988). Effects of Office Interior Color on Workers' Mood and Productivity. *Perceptual and Motor Skills*, [online] 66(1). Available at: http://journals.sagepub.com/doi/abs/10.2466/pms.1988.66.1.123#articleCitationDownloadContainer [Accessed 20 Dec. 2017].

Lifehack. (n.d.). *Toward a New Vision of Productivity, Part 3: The Trouble with GTD*. [online] Available at: https://www.lifehack.org/articles/featured/toward-a-new-vision-of-productivity-part-3-the-trouble-with-gtd.html [Accessed 3 Dec. 2017].

Material Design. (2014). *Introduction - Material Design*. [online] Available at: https://material.io/guidelines/ [Accessed 28 Nov. 2017].

Oldfield, P. (2003). Cost of Change - Modernised. *Appropriate Process Movement*. [online] Available at: http://www.aptprocess.com/whitepapers/CostOfChangeModernised.pdf [Accessed 3 Dec. 2017].

Richter, F. (2018). *Infographic: The Smartphone Price Gap*. [online] Statista Infographics. Available at: https://www.statista.com/chart/4954/smartphone-average-selling-prices/ [Accessed 2 Feb. 2018].

Rubin, J. and Chisnell, D. (2008). *Handbook of Usability Testing: How to Plan, Design and Conduct Effective Tests.* 2nd ed. Indianapolis (IN): Wiley.

Statista. (2018). *Apple: most popular app store categories 2018*. [online] Available at: https://www.statista.com/statistics/270291/popular-categories-in-the-app-store/ [Accessed 1 Feb. 2018].

Statista. (2018). *Average smartphone replacement cycle worldwide 2017*. [online] Available at: https://www.statista.com/statistics/781708/global-average-smartphone-replacement-cycle/ [Accessed 8 Feb. 2018].

Statista. (2018). *Mobile OS market share 2017*. [online] Available at: https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/ [Accessed 1 Feb. 2018].

Statista. (2018). *Number of smartphone users worldwide 2014-2020*. [online] Available at: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/ [Accessed 1 Feb. 2018].

Statista. (2018). *Smartphone daily usage time worldwide 2017*. [online] Available at: https://www.statista.com/statistics/781692/worldwide-daily-time-spent-on-smartphone/ [Accessed 1 Feb. 2018].

Stothart, C., Mitchum, A. and Yehnert, C. (2015). Supplemental Material for The Attentional Cost of Receiving a Cell Phone Notification. *Journal of Experimental Psychology: Human Perception and Performance*. [online] Available at: http://www.psych.uncc.edu/pagoolka/jephpp2015p893.pdf [Accessed 4 Dec. 2017].

To Do List. (2013). Splend Apps.

Twitter. (2006). *Twitter*. [online] Available at: https://twitter.com/ [Accessed 9 Dec. 2017].

Usability.gov. (n.d.). *System Usability Scale (SUS)*. [online] Available at: https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html [Accessed 7 Dec. 2017].

Wunderlist: To-Do List & Tasks. (2011). 6 Wunderkinder GmbH.