

School of Electronic Engineering
and Computer Science

Final Year

Undergraduate Project 2018/19



Final Report

Programme of study:

BSc Computer Science

Project Title: **Intelligent Work Planner**

Supervisor: Thomas Roelleke

Student Name:

Tahmeed Iftikhaar Ahmed

Date: 22/04/2019

This report, with any accompanying documentation and/or implementation, is submitted as part requirement for the degree of BSc Computer Science at Queen Mary University of London. It is the product of my own labour except where indicated in the text. The report may be freely copied and distributed provided the source is acknowledged.

Abstract

This project conveys the development of an android mobile application, 'Intelligent Work Planner', for users in a work-related environment. The application focuses on handling a user's workload by managing their tasks, giving reminders when needed.

This report covers an in-depth explanation and justification of all research that I have carried out, primary or secondary, including researching students that are potential users, the different methodologies and models available to pursue, evaluation of existing similar apps and design for my application.

The most important development cycle stages such as design, implementation and testing are described in detail. The evaluation covers how I handled the project, if requirements were met, objectives and future work.

Contents Page

1.0 INTRODUCTION.....	page 4
1.1 Project Aims and Objectives.....	page 4
2.0 DEVELOPMENT METHODOLOGIES.....	page 5
2.1 Waterfall Lifecycle Model.....	page 5
2.2 Iterative Model.....	page 5
2.3 Prototyping Model.....	page 6
2.4 Chosen Development Model.....	page 6
3.0 RESEARCH METHODS.....	page 6
3.1 Primary Research.....	page 7
3.1.1 Interviews	page 7
3.1.2 Questionnaires.....	page 7
3.1.3 Observations.....	page 9
3.2 Secondary Research.....	page 9
4.0 System Requirements.....	page 9
4.1 Functional Requirements.....	page 9
5.0 EVALUATION OF EXISTING WORK PLANNERS.....	page 10
5.1 Evaluation of 'Tasks: To do list, Task List, Reminder'	page 10
5.2 Evaluation of 'Planner Pro – Personal Organizer'	page 11
5.3 Evaluation Summary.....	page 11
5.4 How This Affects My Implementation.....	page 11
7.0 Design.....	page 12
7.0 DEVELOPMENT LANGUAGES AND ALGORITHMS.....	page 14
7.1 Algorithms.....	page 14
7.1.1 Decision Tree Algorithm.....	page 15
7.1.2 Logistic Regression Algorithm.....	page 17
7.1.3 K-Nearest Neighbour Algorithm.....	page 18
7.1.4 Naïve Bayes Algorithm.....	page 20
7.2 Chosen Algorithms to Implement.....	page 22
7.3 Programming Languages and Development Tools.....	page 24
7.4 Changes in Environment and Databases.....	page 24
8.0 Implementation.....	page 25
9.0 TESTING AND USERFEEDBACK WITH CHANGES.....	page 40
9.1 My Testing.....	page 40
9.2 User Feedback and Changes.....	page 45
10. CONCLUSION / EVALUATION.....	page 47
11. REFERENCES.....	page 50
12. Appendix.....	page 50

1.0 INTRODUCTION

1.1 Project Aims and Objectives

Aims and Objectives

This project aims to provide mobile phone users in a work environment an application, called 'Intelligent Work Planner' (IWP), with 2 main useful features. The first feature is to provide an intelligent way to manage tasks that need to be completed set by a person of high authority (task setters). The second feature is to give the users 'intelligent' reminders/notifications to complete any outstanding tasks depending on how effective they are at remembering to do tasks. These 2 functionalities are incorporated with tackling human forgetfulness and memory.

Target Audience

The application is targeted at organisations or businesses where person(s) of high authority assign tasks or work to people they manage or have a relationship with. For example, employees, students etc. Whereas the people of high authority may be employers, managers, teachers/lecturers etc. For instance, a teacher may set an assignment to a student(s) to complete their homework by next week. Or an employer may set a task for an employee(s) to submit a financial analysis report.

Task setters manually enter into the system tasks that need to be completed by specific people. Each user will get intelligent reminders to complete the task, depending on how often they need to be reminded.

Problem IWP is Solving

The problem or challenge my project aims to solve is finding out how forgetful a person is using artificial intelligence and machine learning. To what extent is a person forgetful? And what determines how forgetful they are?

My project, with research into human's memory, will learn for each user, how forgetful they are and how often they need to be reminded to complete their tasks. Machine learning, artificial intelligence and algorithms – all will play a part in understanding and learning about each user to give them personalised intelligent reminders.

Conclusion

To conclude the introduction, Intelligent Work Planner will use machine learning to predict if a person will forget to do a task or not and receive an intelligent reminder. The application will not give predetermined set reminders. For example, an hour before, a day before, or a week before. Only calculated intelligent ones!

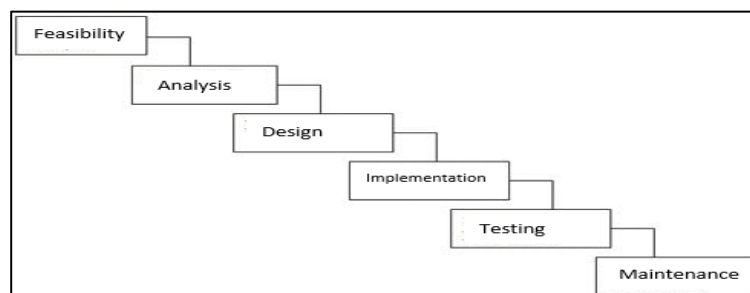
2.0 DEVELOPMENT METHODOLOGIES

A development methodology is a framework that is used to structure, plan and control the process of developing a system. It is important to consider the different methodologies that are available for me to implement. They must be assessed, based on their respective advantages and disadvantages. This section covers the Waterfall Lifecycle model, Iterative model and Prototyping model – these are assessed and a suitable one is chosen for me to conduct throughout my project based on advantages, disadvantages and suitability.

2.1 Waterfall Lifecycle Model

The Waterfall Lifecycle model is the methodology approach conventionally used to complete the development in a structured way which consists of different stages. The model is a sequential process in which progress is made by flowing steadily downwards through the stages of Feasibility, Analysis, Design, Implementation, Testing and Maintenance [1].

A Waterfall Lifecycle model is a good methodology for projects that require a step by step development. This is because the end goal is determined very early. And the different stages involved in the model contribute to achieving the end goal. However, the model is not good to use if the development stages are continually revisited – this could be due to reasons such as a change in requirements or developer capabilities.



2.2 Iterative Model

The Iterative model is another methodology that emphasises on an initial, simplified implementation, which then progressively gains more complexity and more features through each iteration. The model carries out many iterations until the final system is complete, which meets required standards. After a stage, a small handful of previous stages can be repeated numerous with each iteration of the cycle incrementally improving.

The Iterative model can be very useful when developing a system. After each iteration, there can be improvements after evaluation of each stage. The model allows for adaptability to potential change in requirements as developers can work on a previously finished aspect in the next iteration. However, this model can be very time consuming if there are too many iterations and fallbacks.

2.3 Prototyping Model

Prototyping is the methodology that is used for systems development whereby systems are continually being modified. A prototype (an early representation of the final system) is built, tested and reworked until the prototype is at an acceptable state in which the final system can be built from [2]. Prototyping model is a framework where the developer can go back to a certain stage as many times as required. It is almost a necessity that developers go back to the various stages and make changes.

The prototyping methodology, as the name suggests is actually a model of the final system. It is like testing a system which is designed to see if it works properly before actually developing it. It is very advantageous as clients can get a feel of the final system early and changes in requirements are feasible at early to mid-stages. But it can still be argued that this model can be time consuming.

2.4 Chosen Development Model

I have decided to opt towards an Iterative development model. This is because it would allow me to assess, at each stage, where I am and what can be improved in the next iteration. It allows for continual improvement. As I am inexperienced in developing an android mobile application, the Iterative model allows the project to be changed and adapted to my capabilities. If one function is not possible for me to complete at the time due to time constraints for the project, I could work around it and do it differently or do something else. Since there isn't an actual client, the requirements are fixed for my application and can be amended.

The Waterfall model was rejected because I felt the Iterative model was better suited for my project and my capabilities. With the Waterfall model, requirements would need to be clearly defined and set which have to be worked towards to at each stage. There wouldn't be any 'going-back' to previous stages. If at the implementation stage after design stage, I wouldn't be able to go back to the design stage to change something if it wasn't working due to time constraints.

Similarly, the Prototype model was also rejected as I felt this would require too much time and effort. As I am inexperienced in developing applications, having an early working prototype was not possible.

After deciding on the methodological approach for my project, it was essential to come to a decision about how information required before designing and developing was going to be collected. The next chapter looks at the research methods that were adopted and how they were implemented.

3.0 RESEARCH METHODS

There are two main types of research methods. The first one is primary research. Primary research is research that is done first hand by you. I will need to carry it out to find out more about my applications needs. Secondary research is research that has already been done by other people that can be used to assist myself in my project. These researches could be essential in assisting to create my final system. Initially, I had to decide on what to do for my project. After lots of thinking and analysis, I decided to create an Intelligent Work Planner. The different research methods confirmed whether to go with my idea.

3.1 Primary Research

The primary research I have carried out involved gathering information directly from people that had not been previously gathered by other people. There are many different methods of carrying out primary research. The ones I carried out are interviewing people, creating and handing out questionnaires and observations. The following segments briefly touch on these research methods.

3.1.1 Interviews

Interviews are discussion-based fact-finding methods. It conventionally consists of 2 individuals, one who is an interviewer that asks several questions to extract useful information, whilst the other individual is the interviewee, which is the person being asked many questions who is able to give their insight into particular topics or concepts.

Interviews typically involve long answer questions which allows interviewees to go in depth and explain their insight, how they feel or elaborate on what they know. This allows a discussion to formulate as when the interviewee answers a question, interviewers can expand or ask another follow-up question to get more detailed information.

One of the main reasons why I chose to carry out interviews is due to the fact that interviews are flexible as questions can be changed according to the answers given to the previous questions.

Questions and Findings from Interviews

After interviewing several different students informally around Queen Mary's campus, I was able to come up with many conclusions. Some of the questions I asked were:

- Do you think a work planner that is specific to each user is a good idea?
- Do you think organisations will benefit from an application such as the proposed one?
- Are you likely to use a similar application to maintain relations with lecturers/teachers?
- Can you explain how current work planners can be improved?
- What is likely to attract you to use a work planner?

From the questions that I asked many students, many conclusions have been made which suggest my project and application is worth pursuing. Some of the responses, summarised, are as follows:

- People don't usually use work planners as they feel its not necessary, but ones that are specific to each user that utilises artificial intelligence is a good idea
- People want a sophisticated work planner which suggests which tasks to do in which order
- Maintaining relations with lecturers and teachers is a good idea to keep track of everything
- Organisations are likely to benefit from the concept of the application. But this may be for relatively small organisations.
- Current work planners are not appealing as people feel they can remember themselves
- A key factor is the notion of intelligently suggesting tasks to do in a specific order

3.1.2 Questionnaires

As part of my primary research and requirements collection, I have prepared an online questionnaire for people to answer. With each question, respondents were able to add notes if they wanted to elaborate

or justify. This was done to get an insight into what people think of my idea and what is their perception on existing similar applications.

28 people responded to my questionnaire. Here is a representation of their answers. Diagrammatical representations can be found in appendix 12.1.

1. How often are you likely to use a work planner?

From this question, it can be concluded that that majority of people are more likely to use a work planner. 81% of people are likely to do so. This connotes that my application is appealing to people and they may want to use it.

2. How useful would an 'intelligent' work planner be to you?

92% of people believe that an intelligent work planner will be useful for them. This shows that my Intelligent Work Planner could be put to good use.

3. To what extent would you like the Work Planner to manage all of your tasks, suggesting which ones to do first? (1 being 'You don't want this and 5 being 'You want this')

Majority want the application to manage their tasks in an appropriate order. People don't just want to set their tasks in order by date, a good algorithm should be in place where it manages and orders tasks elegantly.

4. The application would remind you about work-related tasks between you and your task setter (e.g. submit analysis of findings by tomorrow). Would you want the application to remind you on trivial daily tasks (e.g. purchase milk today)?

67% of respondents stated that they want the app to be specific to work related tasks set up by task setters. Trivial tasks should not be a part of this, otherwise it would be very generic. Some comments made in the notes section are as follows: "It should be strictly work related" and "It would be good if its work related only, otherwise it might get too complicated or messy".

5. The application will assess your memory, privately to you. Is this an issue?

From this question, it can be seen that most people are comfortable with my application interpreting how good their memory is. It should be kept private to them. Only 17% of people are not completely comfortable with it. Some comments are: "As long as it doesn't ridicule people, it is fine" and "It should interpret memory sensitively". These comments show that it is a sensitive topic and should be handled with care.

6. To what extent is including a small social space to communicate with your task setter appealing?

Results from this question imply that most people want to be able to communicate with the person who sets their tasks. It can allow for them to quickly ask questions if they require help or are unsure and require clarification. 78% of people want a small social space to communicate. One comment made was: "Communicating will make it easier to reach out".

7. Applicable to task setters: Are you willing to set up tasks for each user with important information e.g. deadline, prominence, difficulty etc?

This question was specifically made for people that assign tasks for other users. Other people have been advised not to complete this question. 87% of people have said that they are willing to set up tasks with the required information. This means that both parties are willing to use the proposed application to their benefit.

8. Are set reminders unappealing compared to intelligent predicted ones?

82% of respondents prefer intelligent predicted reminders/notifications compared to predetermined set ones. This will be the basis of my Intelligent Work Planner.

Summary

After reviewing the answers from the questionnaire, I have a good insight into what people are expecting and want in the application. These will be used to develop system requirements.

3.1.3 Observations

A short and simple method I carried out, but not too intensively are observations. This involved me, the analyst, inspecting random people use currently available work planners. In this case, these people were random students in the Queen Mary campus. As the observer, I watched and made notes on how the applications are being used. It gave me a good perception of what can be improved. Some perceptions were retrieved by me actively asking questions to the observed whilst they use the work planners. From the observation, I concluded what my application should do, what it should look like and how it should be used.

Findings from Observations

- People preferred a simple GUI
- All tasks should be shown in a page
- A colourful and spacious interface was preferred
- No one struggled to set up a new task / reminder
- There was not enough innovation in the apps
- Users found app loading times to be a bit too long

3.2 Secondary Research

Secondary research can be carried out in different ways. Some ways are extracting data from books, the internet, articles, journals etc. It involves looking deeper into the research that has already been done in specific topics.

As part of my secondary research, I had to research algorithms on machine learning and artificial intelligence to understand how to implement them into my application. More on this can be found in chapter 7.

4.0 System Requirements

In this section, I will be listing out the requirements that I have gathered from my research and project specification. They are open to change, depending on how my project goes. The requirements are mostly

functional requirements, which should be met. There aren't any non-functional requirements as of yet. This is because my application is for general users and not specific like students. So most non-functional concepts have to be addressed appropriately, catering for generic use.

4.1 Functional Requirements

ID	Requirement
R1	System must store all tasks for each user e.g. assignment, project etc.
R2	For each task, system should collect information - deadline, difficulty, importance and whether it is assessed or not
R3	System must sort all tasks suggesting which ones to do first
R4	User enters how much they have completed each task (percentage wise) and algorithm for suggesting which ones to do updates
R5	Users can mark off tasks when they have been completed
R6	There should be an algorithm that learns about each user (forgetfulness)
R7	User should receive reminder/notification about outstanding tasks – if algorithm suggests they may need to be reminded
R8	System should send reminders based on the ability and personality of user (in terms of forgetfulness)
R9	For similar tasks, system should use an algorithm to predict whether a person will forget that task based on other similar users
R10	Users enter times they are free, and system suggests when to do each task
R11	Users can choose level of reminders as user preferences
R12	There should be a social page where user can communicate with their task setter
R13	Each user must be identified with a unique username.
R14	User should be able to login with a username and an asterisk-protected password
R15	Incorrect login details should display a popup error message, prompting user to try again
R16	Users need to enter details e.g. name, dob etc, which can later be edited
R17	Task setters should be able to assign tasks to 1 or more other users
R18	Users should be able to receive assignments from multiple task setters
R19	There should be a 1-1 relationship between a user and a task setter
R20	Users should be notified when a new task is assigned for them
R21	System should be a mobile application (android)

5.0 EVALUATION OF EXISTING WORK PLANNERS

In this section, I will be documenting my research and findings of existing work planner applications or similar ones. This is necessary to get an insight of what my target audience may expect, what features I will need to include and what aspects I may need to remove or improve.

5.1 Evaluation of 'Tasks: To do List, Task List, Reminder'

This application is found on the Play Store, published by developer Stephen Nottage [3]. My findings are as follows:

- Simple application to add tasks with a date
- Reminders are sent on a chosen day and time by user
- Uses gestures to do certain actions

- Colours (customisation) is heavily included
- Ability to 'snooze' reminders

5.2 Evaluation of 'Planner Pro – Personal Organizer'

This application is found on the Play Store, published by developer Appxy [4]. My findings are as follows:

- Users are able to track tasks by project, subtask, and status
- There is an additional optional feature to add 'notes'
- Calendars are the main focus

Following these 2 applications, I researched more applications and they were all very similar. So, I did not document them here as they all achieved the same thing.

5.3 Evaluation Summary

Having researched 4 different applications on the Play Store, I am going to explain the similarities and difference to my proposed application.

Similarities

There were several commonalities between all of the applications I have researched. Majority of apps usually included a large calendar in the main screen and users click a specific data to add a reminder.

Majority of apps included the feature to add notes. The user can give detailed information about the task or event alongside the title.

Differences

Some apps allowed users to receive reminders at a specific time of the day, which the user can choose.

Some applications required users to register an account in order to use the app.

5.4 How Research Affects My Implementation

From my research into existing systems, it can be concluded that there is no application like the one I am going to build. None of the applications involved a relationship between 2 or more people (user and task setter). Users set their own tasks and aren't assigned tasks by anyone else. The applications are not organisation or business oriented. They only involve one user who sets their own reminders. It is very generic.

The aspects I am going to include in my application from existing ones are:

- Use of calendars to allow users to choose specific dates
- Feature to allow users to enter notes with each reminder
- Consider having a simple GUI for adding new tasks

Alongside these features, I will be implementing many other concepts. There will be tasks set out by others which need to be completed. A social space will be included to allow communication between the task setter and the users.

6.0 Design

Login Page

Intelligent Work Planner

Username

Password

login

Don't have an account yet?

[Sign up](#)

This is the first main page a user will come across. User has the option to enter their login details and login to the main page. If they don't have an account, they can register by clicking 'Sign Up'.

Sign Up Page

Intelligent Work Planner

Sign Up

Username

Password

Name

Email

...

Sign Up

Here, users need to enter required details such as a username, password, name, email etc. More required fields will be in the final application. When the 'Sign Up' button is clicked, it will check the data is valid. And then direct them to a welcome page, or to the homepage.

Tasks to do Page

Tasks to do

Task	Due by	Percent Complete	Note from Task Setter
Submit Analysis	25/12/2018	10%	This is very important and should be done to a professional standard
Start Report	06/01/2019	0%	
Amend Report	04/12/2018	50%	URGENT!
X	xx/xx/xxxx	xx%	Xxxxx xxxxxxxxxxx x x xxxxxxxxxxx
Y	yy/yy/yyyy	yy%	

This is an illustration of what the homepage may look like. It would immediately show the tasks that a user has to do, if there are any.

There is a button in the top left which would open a menu of actions that a user can do. This is shown in the next design.

Menu Bar

Tasks to do

Profile pic

Name

Email

Home

Manage Tasks

Timetable

View Suggested Tasks

Social Space

Find People

View Relations

Settings

Help and Feedback

Note from Task Setter

This is very important and should be done to a professional standard

URGENT!

Xxxxx xxxxxxxxxxx x x xxxxxxxxxxx

This design shows all of the possible actions that a user may carry out.

For each option, there is a design illustrating how they will look, following this one.

Manage Tasks Page

Manage Tasks

Task	Due by	Percent Complete	Note from Task Setter
Submit Analysis	25/12/2018	10%	This is very important and should be done to a professional standard

Task Name:

Due by:

Assessed?

☒

Difficulty

Importance

After choosing a task, from either the homepage or from the ‘Manage Tasks’ page, it will display all information regarding that task such as name, prominence, deadline, notes etc. The user can delete the task to signify it is complete or amend the percent complete so that the algorithm for suggesting which tasks to do updates.

Suggested Tasks

Suggested Tasks

Task	Due by	Percent Complete	Assessed	Difficulty	Importance
Submit Analysis	25/12/2018	10%	Yes	5	4
Y	06/01/2019	0%	Yes	4	5
X	04/12/2018	50%	No	5	4
Amend Report	xx/xx/xxxx	xx%	No	3	2
Start Report	yy/yy/yyyy	yy%	no	1	2

This page displays a suggestion of which tasks to do first. It will always change depending on the algorithm I build (e.g. percent complete, importance level, difficult etc.)

This is different to the tasks show in the main page because that just lists all of the tasks to do. But here, it uses an algorithm to suggest which ones to do – which the order will always change.

Social Space Page

Social Space

You: Hi, could you explain how I would go about my task?

Tom: Hi, sure. You have to first research the theme and then document it. Hope it helps.

You: Thanks Tom, that really helped.

Also, could you give an example of where I could research?

send

This page can be accessed by clicking ‘Social Space’ from the menu. A user will be prompted to choose a person they are following and upon clicking one, it will show the messages that have been sent. And they can also use the text bar on the bottom of the page to type and send to the other user.

Find People

Find People

Search for people to connect with

Name:

John

Search

John Alderwield

View profile

Message

Connect

John Cena

View profile

Message

Connect

John Young

View profile

Message

Connect

John Stone

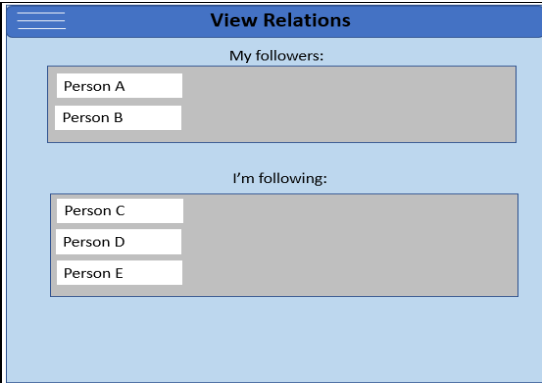
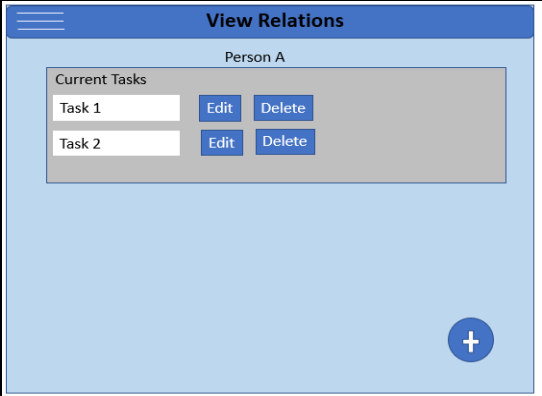
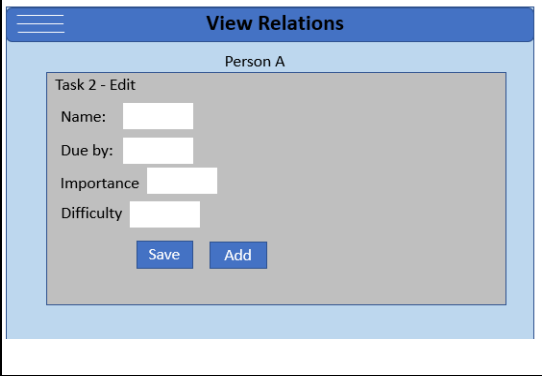
View profile

Message

Connect

Through this page, a user can find another user to follow. This allows people to be connected and would allow a person to assign a task to another.

View Relations

	In this page, a person can view all of the people that they currently are following and the people that are currently following them.
	When a person is clicked from the previous page, if under 'My Followers', it would show all of the tasks that that the user has assigned for that specific person. From here, they can edit or delete the tasks that they have set.
	If a user decides to edit one of the tasks, it will show this page, displaying the information regarding that task. These fields can be edited or can be used to add a new task for that person.

7.0 DEVELOPMENT LANGUAGES AND ALGORITHMS

7.1 Algorithms

For my project, to achieve many of the requirements, I will have to learn and use machine learning algorithms. For example, to sort and suggest which tasks a user should do, an intelligent algorithm will have to be used. Another example, for predicting whether a user will complete their task before the deadline or not will also require the implementation of a machine learning algorithm.

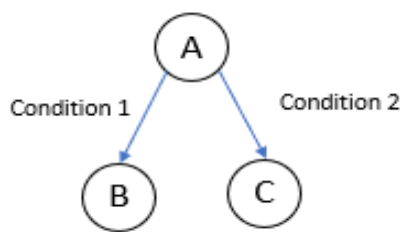
As a result, this section contains the research I’ve carried out on different machine learning algorithms: Decision Tree, Logistic Regression, K-Nearest Neighbour and Naïve Bayes.

7.1.1 Decision Tree Algorithm

Decision Tree Algorithm uses a 'tree' to make a decision from observations and findings. But first, what is a 'decision tree'?

A Decision Tree is a decision support tool which contains a starting node. The initial node contains paths to other nodes [5].

From node A, we go to either node B or node C. This depends on the condition. To illustrate, we could have a variable x . Condition 1 could be ' $x < 5$?' and condition 2 could be ' $x \geq 5$?'. If our variable x has value 7, then we follow the path to node C because the condition is true and the other one is false.



Now we know what a 'Decision Tree' is, we can focus on the Decision Tree Algorithm. As mentioned above, Decision Tree Algorithm uses the Tree to make decisions. It starts from the initial node, and follows several paths based on conditions. This is done repeatedly until the last set of 'decision' nodes has been reached – one where there is no condition to go to another path. It is the final decision and it is the conclusion that can be agreed on.

Advantages

Decision Trees are really easy to follow without any complications.

Decision Trees are relatively easy to code and implement. Lots of 'if-statements' would be required to make decisions.

Disadvantages

Decision Trees will not guarantee if it chose the right answer. It is more of an interpretation. We won't know if it is the best or correct choice. A little change in the tree will significantly impact the final decision made [6].

If there is a change in one of the nodes due to unforeseen circumstances, the Decision Tree would need to be amended from the point of the change right until the very bottom. This can be time-consuming [7].

My Own Implementation

Decision Tree can be found in appendix 12.15

The decision tree I constructed checks a person's memory score and acts accordingly. It follows paths and conditions to reach a decision to decide by how much the score should be changed.

The decision tree is used to change memory score when a person completes a task. Only then will the decision tree algorithm be executed.

There are 3 starting routes which can be followed through, which break down into more potential routes further down.

The first route can be progressed through if the memory score of the user is less than 40. A score that is less than 40 denotes that the user generally is good at remembering to complete tasks in time. There are

further routes that can be taken from here, depending on when the task was finished, in relation to the deadline.

If the task was completed a few days before the deadline, a separate route is followed. The average score is then calculated and compared to the current score. If the current score is less than the average score, that connotes that the user is improving in completing tasks on time. Whereas if the current score is larger than the average score, this implies that the user is beginning to or almost begin to complete tasks after the deadline, which is not a good sign. So, if the current score is less than the average score, since it is a good sign, the user's memory score is decreased by 5 points, which shows a substantial improvement. On the other hand, if the current score is greater than the average, it shows slight deterioration in the improvement – but they still have completed the task a few days before the deadline. As a result, the memory score is decreased by 4, compared to the 5 as this time round, the performance has slightly worsened.

If the task was completed very close to the deadline, another route is taken. This route shows good performance but not as good as the one described in the previous paragraph. This is because, in the first route, the task was completed a few days before the deadline which is good performance, so the memory score was reduced by a significant amount. As a result, in this route, the scores decrease by smaller proportions. If the current score is less than the average score, the memory score is decreased by 2. This is a small improvement, so the score is improved by a small amount, 2. If the current score is greater than the average score, then the user has worsened compared to their average, but still has completed the task in time but very close to the deadline – so the memory score is reduced by only 1.

If the given task was completed after the deadline, the memory score increases by specific amounts. So if the current score is less than the average, it is a slight improvement compared to the average, but the user still finished after the deadline - so the memory score increases by a small amount, 2. If the current score is greater than the average score, this implies worsened performance. As a result, the memory score is increased by 4.

The 3 paragraphs above contain 3 different paths depending on when the task was finished. If the user's memory score is less than 40, it shows 1 large path. There are 2 more large paths. One is met if the current score is between 40-60 whilst the other one is met if the score is more than 60. The large paths follow the same structure (in terms of checking when the task was completed and comparing current score to average score) but with different increases and decreases in scores. For example, for the third large route (when memory score is more than 60), if a task was completed after the deadline and the current score is greater than the average score, it connotes that the user has worsened very much. As a result, the memory score increases by 7 which is significant.

All values for each path have been taken into consideration carefully and can be seen and followed through the decision tree in the appendix.

7.1.2 Logistic Regression Algorithm

Logistic regression is a mathematical model used to predict the probability of an event occurring. It falls under binary classification where an event can either happen or not. It takes binary values 0 and 1 where 0 means the event did not take place and 1 means the event did take place [8].

Before I go more in-depth into logistic regression, I will have a brief look at a similar model, linear regression, and how they differ.

Linear Regression

Linear regression models the relationship between 2 variables. We have a dependent variable and an independent variable. The independent variable is used to predict the value of the dependent variable. It uses a linear equation which describes a line that best fits the relationship between the variables [9]. An example could be how the area in square feet of houses affect the price of the house.

Back to Logistic Regression

In logistic Regression, the outcome (dependent variable) has a limited number of possible values, whereas on the other hand, the outcome is continuous with Linear Regression – it can have an infinite number of possible values [10]. To elaborate, with the house example, we can keep on adding to the regression graph. The bigger the house, the more the house will cost. With Logistic regression, because it's a binary classification, we can model questions such as whether a house will cost more than £300,000 if it has a size x. We either get a binary answer of 'yes' or 'no'. So, the outcome is categorical which fits perfectly with my project to determine whether to send reminders (notifications).

Logistic Regression uses the logistic function which gives an 'S' shaped curve for modelling the data. The curve is constrained between 0 and 1, showing the probability for dependent variable for a given value of the independent variable.

Advantages

Having the dependent variable categorical also means that the independent variables can take any form. This allows several problems to be modelled and satisfied. For example, whether a given house with size x metres squared will be cheaper than £300,000.

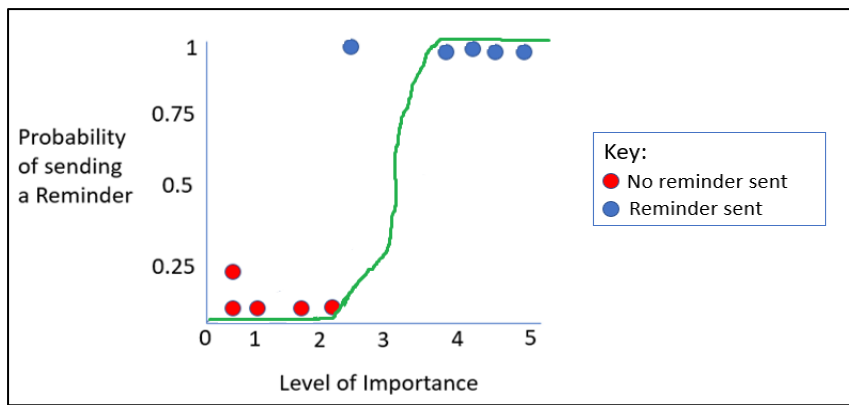
Disadvantages

Logistic Regression does not perform too well when the feature space graph is too large. Modelling becomes much more complicated and arduous.

This model also doesn't handle well large number of categorical variables or independent variables. When you have multiple independent variables, you would require more dimensions in the regression graph [11].

My Own Implementation

In my own implementation, I would model independent variables such as 'Level of Importance' with dependent variable 'Probability of sending a Reminder'. An illustration can be seen below.



From previous data, I would construct a similar computed Logistic Regression graph with the independent variables. As a result, when a new input comes, for example Level of Importance is Level 4, it can be predicted that this time a reminder should be sent. This implementation takes into consideration only one independent

variable. As a result, the logistic regression model would need to be used in conjunction with all other variables to make a decision whether or not to send a reminder.

In order to cater for other independent variables such as 'Level of Difficulty', 'Assessed or not' and 'days until deadline', individual logistic regression models can be created. For each independent variable, the probability of sending a notification would vary.

For example, probability of sending a notification when importance level is 4 may be 0.8. Probability when days until deadline is 20 days may be 0.6 etc. All of the different probabilities would then be multiplied. If the answer is greater than 0.5 for example, a notification would be sent, otherwise not. E.g. $0.8 * 0.6 * 0.9 * 0.9 = 0.39$. So, it can be concluded that a notification would not be sent.

7.1.3 K-Nearest Neighbour Algorithm

K-Nearest Neighbour (k-NN) is an algorithm used for machine learning that determines the class of an input using its k-nearest neighbours. There is a feature space which is an n-dimensional vector on values that illustrates an object or concept. The feature space contains the same representational data scattered around on the vector based on factors [12].

These factors are like variables which make up the data representation. And these factors or variables determine the positioning of the data representation in the vector/feature space.

The k nearest data representations are identified and whichever class occurs the most within the identified set of data, the new input is classed as that.

K-NN is a lazy algorithm because it doesn't learn a discriminative function from the training data set. Instead, it memorises the training data set – meaning that it doesn't create models of the training inputs in advance. When new inputs are queried, the algorithm is executed to predict the class of the new input. Due to the fact that inputs are locally approximated for each new one, k-NN can simultaneously solve several problems and deal with changes in the problem domain.

Advantages

This algorithm is very effective if the training data is large. We would get a more accurate prediction.

There are no assumptions made unlike Naïve Bayes (e.g. assumes factors/attributes are independent). K-NN is based on the current local data and lets the 'data speak for itself'.

The use of dimensions can be very useful when plotting the data and using it to predict the class of a new input [13].

The cost of the learning process is inexpensive.

Disadvantages

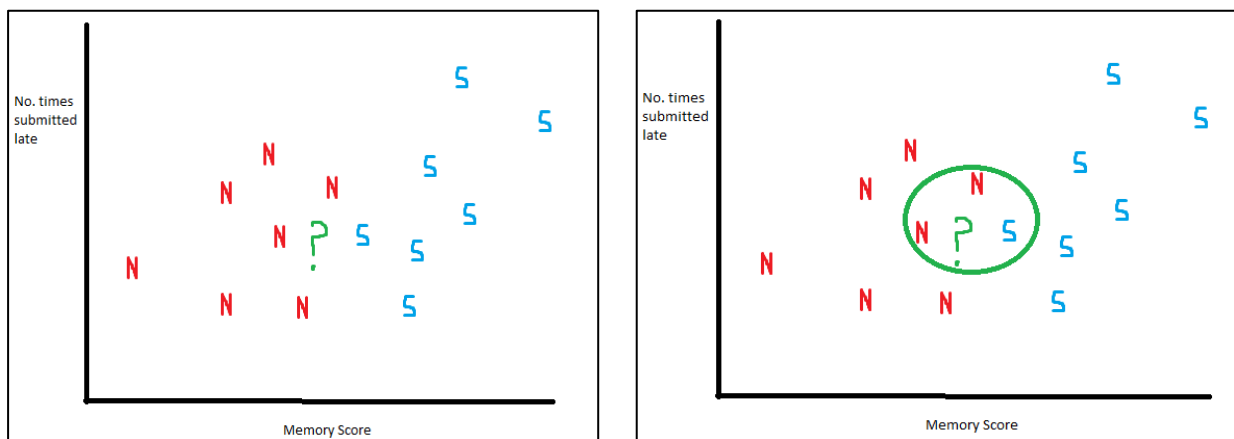
We have to determine an appropriate value of k for finding k nearest neighbours. The value can have a significant impact on the accuracy of the prediction, dependent on the training data set size [14].

Computing the distance between new input to all other nearest data representations can be computationally costly – especially when dataset is very large.

My Own Implementation

Use k -NN and have 2 factors/variables:

- 1) Number of times they submitted late or almost submitted late
- 2) Memory Score



For each task, there will be previous data on the vector space, indicating whether for each person they missed the deadline or not. In the graph above, there are 2 axes. One axis represents the memory score, 0-100. The other axis contains number of times people submitted tasks late. This would vary depending on the number of tasks submitted late overall.

The system would, for each task, plot the respective data – the number of times they submitted late, their memory score and whether or not a notification was sent. This is denoted by the red 'N' and blue 'S' in the graphs. 'N' to denote a notification was not sent and an 'S' to denote a notification was sent. This will be done for all tasks currently in the database. This process will be carried out each time the application is opened. This is to get updated information.

When there is a new query for a new input, the input will be approximated locally depending on the factors/variables – the axes. A decision to determine the class of that new input will be made using k to find nearest neighbours, by distance. We must give k a suitable value.

With the new input, denoted by the question mark on the graph, we can determine its class – whether or not to send a notification. Using value $k=3$ for example, the 3 nearest data representations are located. The algorithm will handle locating the nearest ones. From the 3 data representations, the occurrence of

the 2 classes will be counted. The class that occurs the most will be what the algorithm returns. This returned class will determine if the new input task is likely to be completed after the deadline or not. If the class 'Sent' occurs more than 'Not Sent', then a notification will then be sent for the new input task.

Based on the new task, the memory score and number of times submitted late are placed accordingly in the feature space. In this example, we used $k=3$ to determine nearest neighbours. In the diagram, it can be seen that the first 3 nearest neighbours are circled. In the circle, there are 2 'N's and 1 'S'. From this we conclude that the new task is not likely to be forgotten so a notification is not required. This is due to the fact that there are more 'N's than 'S's.

The value of k needs to be odd. This is to ensure that a most occurring class can be determined. If k was even, they could be the same which would not be useful. A new value of k would need to be used.

Euclidean Distance Formula

A common distance metric that is used to calculate the distance is the Euclidean Distance formula.

$$d(\mathbf{p}, \mathbf{q}) = d(\mathbf{q}, \mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}$$

$$= \sqrt{\sum_{i=1}^n (q_i - p_i)^2}.$$

Figure x:

The Euclidean distance is the straight-line distance between two points. Points have coordinates. And the formula above is a Pythagorean formula which uses the different coordinates of the points to find the distances [15].

This Euclidean distance formula will be used and interpreted in Java to find the distances. This will be used to determine which points have the shortest distance, meaning which ones are the nearest. This will be used to find the varying k nearest neighbours to determine the closest ones. The occurrences of the different classes will be counted ('notification sent' and 'notification not sent') to determine what will be the class of the new input.

7.1.4 Naïve Bayes Algorithm

Naïve Bayes classification is a type of Probabilistic Classifier that predicts the class type of a given input based on its features. It is a form of machine learning to learn about inputs. Naïve Bayes uses Bayes' Theorem to define the probability of an event, based on features or conditions known beforehand.

This classification method is called 'naïve' because it assumes that the occurrence of events/attributes is independent of the occurrence of other events/attributes. As a result, the probability of the events occurring simultaneously is the product of their respective probabilities [16].

With Naïve Bayes classification, we have pre-classified examples (previous inputs). These examples and their probabilities are used to determine the probability of a class of a new input. With machine learning, we initially go through a 'training' phase where we learn about the features of examples (previous inputs)

that are already assigned to classes. So, when a new input comes along, we match its features against the learnt features of classes to calculate the probability of which class the input should be assigned to.

$$p(\text{class}|\text{features}) = [p(\text{features}|\text{class}) * p(\text{class})] / p(\text{features})$$

The above formula is used to determine which class a new input should be classified into. For each class we have, we use the formula, and which ever had the highest probability, the new input will be classified into that class.

Advantages

It is fast and relatively easy to predict the class of an input data set.

It performs well in multiclass predictions due to the fact that events are independent to each other which therefore performs better than other models such as Logistic Regression [17].

Disadvantages

Naïve Bayes assumes all events/attributes are independent of one another which is not always the case. Some features may actually be dependent on one another [18].

For more reliability, it is good to have large data set of previous inputs when making predictions. Naïve Bayes can still be carried out with a small data set but this reduced the reliability of the classification.

Another disadvantage is that if we have incomplete training data, we may find it troublesome to predict the class of an input. For example, in the training data, if we have no instances of a certain class with a specific attribute/event, then the conditional probability would be 0.

My Own Implementation

I would have my own features such as 'Importance Level', 'Difficulty Level' and 'Assessed?'. They would have their own values:

Importance Level: Insignificant; Significant

Difficulty Level: Very Easy; Easy; Neutral; Hard; Very Hard

Assessed: Yes; No

Example Data:

Dataset

Importance Level	Difficulty Level	Assessed?	Sent Reminder?
Insignificant	Easy	No	No
Significant	Hard	No	Yes
Significant	Very Hard	Yes	Yes
...

Individual Probabilities for Each Feature

Importance Level				
	Reminder	Reminder'	p(Reminder)	p(Reminder')
Insignificant	X	X	y/z	y/z
Significant	X	X	y/z	y/z

Total	T	T	T	T
-------	---	---	---	---

Difficulty Level				
	Reminder	Reminder'	p(Reminder)	p(Reminder')
Very Easy	X	X	y/z	y/z
Easy	X	X	y/z	y/z
Neutral	X	X	y/z	y/z
Hard	X	X	y/z	y/z
Very Hard	X	X	y/z	y/z
Total	T	T	T	T

Assessed				
	Reminder	Reminder'	p(Reminder)	p(Reminder')
Yes	X	X	y/z	y/z
No	X	X	y/z	y/z
Total	T	T	T	T

If we are given a new input with

features:{
 Importance Level = Significant,
 Difficulty Level = Hard,
 Assessed: Yes
 }

From the data, I would use the appropriate probabilities from the individual probabilities for values for each feature. This would be done twice – once when Reminder is sent and another when Reminder is not sent. The probabilities would need to be multiplied for both classes and normalised. Then the class with the highest probability will be the new classification for the given input.

7.2 Chosen Algorithms to Implement

After thorough research of all the different machine learning algorithms, I have had to decide which algorithms to implement into my project. After careful analysis and consideration, the chosen algorithms are as follows:

- Decision Tree Algorithm
- K Nearest Neighbour Algorithms

7.2.1 Decision Tree Algorithm

I chose this algorithm for the Memory Level concept. As mentioned previously, the tree would have roots or paths depending on what the memory score is. And based on that, more decisions would be made depending on the average score from previous tasks.

I have chosen Decision Tree for this concept because it is a good way to determine how the memory score should change depending on the user's performance. I will be able to follow paths. For example, if their memory score is more than 60, I would check how late they submitted. An appropriate decision

would be made by following other paths by computing average score from previous 3 tasks and see how much to increase or decrease the score by.

With Java, I could effectively implement a Decision Tree algorithm, considering nodes, branches, methods to add nodes, initialise trees, etc. The algorithm would make use of elegant 'if-statements' to follow different paths and make sensible decisions.

I will look more in-depth into implementing a Decision Tree algorithm in Java nearer to the time I start planning all my algorithm implementations.

7.2.2 K-Nearest Neighbour

A huge aspect of my project is deciding whether or not to give a user a reminder on a new task. The app would need to see if it is similar to previous tasks and make a calculated and sensible decision. K-Nearest Neighbour is the chosen algorithm for this concept because it is an effective way to plot similar tasks and compare with other ones and see if a reminder should be sent or not – this will be based on factors such as memory score and to what extent people forgot or submitted late.

With K-Nearest Neighbour, we can position a given input to the graph according to its features (memory score and number of times they submitted late / almost submitted late). Using this position, we can check the number of nearest classes and the class that occurs the most will be the new class of the given input (send reminder or don't send reminder).

This was chosen over other algorithms such as Naïve Bayes and Logistic Regression. This is due to many reasons. For example, with logistic regression, a simple model can only determine the probability of sending a reminder against one feature only. To model with more than one feature, another dimension would need to be introduced or having more models, which I feel would be time-consuming to implement. Moreover, regarding Naïve Bayes, it assumes events/features are always independent of each other. Considering features like 'Importance Level' and 'Assessed or not' can be dependent of each other. So, I thought it would be best not to implement Naïve Bayes.

7.2.3 Manage Workload Algorithm

This is my very own algorithm for the 'Suggestions' or 'Tasks to do' concept of my application. Each assignment or piece of work will come with many features specific to it. For example, a given deadline, how important it is (according to the person who set the task), how difficult it is (also depending on the person who set the task) and whether or not the assignment/task is assessed.

For the above features mentioned, they will have a scale, 1-5, to determine the extent of the value of the feature. For instance, 'Importance Level', 1 would be least important and 5 would be the most important.

Regarding my algorithm, it would need to work in a way that it utilises all of the features associated with the given tasks and sort all of them from least important to most important. How would this be done? Since each feature would have a scale indicating its values, my algorithm will add up the score for each feature for each task. Then all tasks will be listed in the 'Tasks to do' section in order of their total score.

An example might be, for task 'Complete Report for Manager'. It may contain features with scores as: Importance Level = 4, Difficulty Level = 2, Assessed = 5, Deadline = 4. The sum of these scores is 15/20. There might be another task and scores may be: Importance Level: 3, Difficulty Level = 1, Assessed = 0, Deadline = 5. The total score would be 9/20.

The algorithm would do these calculations and sort them in order of their total scores. So, in the example above, the task with score 15 would be suggested to be at the top, above the other task due to the fact that it has a lower score. This algorithm would be applied for all tasks that need to be completed for each user.

7.3 Programming Languages and Development Tools

This section will inform which database system and developing languages will be used for my project as well as developing environment.

7.3.1 Database to be Used

There are many different databases that can be used such as SQLite, MySQL, MongoDB, CouchDB, Oracle Database etc. All are very good for storing and retrieving data, especially when working with mobile applications – some more than others.

I have opted to choose SQLite. This is because it is a relational database management system. It will be used to store any relevant information that may need to be retrieved by other users. SQLite is suitable for my project because it is well suited for storage of data from mobile applications. Moreover, SQLite is 'lite' so it doesn't require a lot of storage, which could be an issue as some mobile phones may have limited storage capacity. I also have some experience in SQLite which would make it easier for me to work with. I have a good understanding of primary keys, foreign keys, tuples, tables etc.

7.3.2 Programming Languages to be Used

The development languages, tools and libraries I will be using to implement my application are Java, JavaFX, and CSS. I chose Java because it is my favourite language which I am experienced in. JavaFX is a useful library for developing user interfaces as part of Java. So, this will be utilised with CSS for styling.

Java will be used with SQLite. The Java Database Connectivity (JDBC) API will be used to achieve this. Java Database Connectivity API is supported with Java and I have practiced using it once which worked out pretty well. I feel as if it would be a good idea to do so again as I would know what I am doing.

7.4 Changes in Environment and Databases

After lots of consideration and research, nearer to the development stage, I had decided to change how I would approach my project. First and foremost, I discovered Android Studio which is designed to help build android applications. Since I am developing a mobile application, it would be better for me to develop on Android Studio. There is also lots of support online for android app development. This is a key factor in deciding to use Android Studio. Even though I am experienced with JavaFX, I believe it would be better to opt towards Android Studio as I would learn something new. Moreover, it makes sense to use

Android Studio rather than JavaFX to build applications as Android Studio is a norm for this. Mobile applications can be developed more appropriately and professionally in Android Studio, providing lots of useful features specific to app development and deployment.

Another big change was the change in the database system I decided to use. Initially, I had chosen to use SQLite to store user data. This was because I had some experience and felt it is suitable. Even though SQLite is suitable for my project, I discovered Firebase, a Backend-as-a-Service database, which was a real time online database. It is very convenient when it comes to storing and retrieving user data. Editing data is even better as you can directly amend the data as it is all in JSON form. You are able to directly change the JSON file which can then be imported as the new database and the changes will be reflected immediately. Another factor that contributed to the decision I made of choosing Firebase over SQLite is that Firebase is well supported with Android Studio. Creating and connecting to a database is very convenient as the functionality to do so is provided within Android Studio, making the task very quick and easy to do. In addition to this, I have not worked with Firebase before whereas with SQLite I have some experience. It would be more beneficial for me to learn how Firebase works as I feel it would be more practical and useful – especially in the real industry.

8.0 Implementation

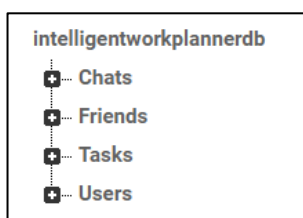
In this section, I will be explaining the most important aspects of my design I have developed and met. For each one, I will show relevant snippets of Java code of classes I wrote or used from external sources to help me develop the pages.

Due to the page limit of this report, the code shown in this report will not be indented to include as much code as is needed.

Disclaimer: YouTubers 'KOD Dev' and 'Coding in Flow' were used to learn how to use Android Studio and code key functionality – these are referenced in the source code.

8.1 Firebase

Before going in to the Java implementation, I will be explaining the Firebase database I have created.



Firebase consists of nodes. It is a tree structure, where data is imported to and exported from JSON files. The JSON files contain the textual representation of the 'database'. The 'Users' node contains all users of the application. There are nodes coming off this node which each represent a user. The user is assigned randomly a key. And in each of these nodes/keys, there are more nodes which represent the data associated with each user such as name, date of birth, username etc.



This diagram shows a segment of the different users there currently are in the application. In this example, '4Bk6kBn...' is the id/key for that user. Each part of this node e.g. 'dob', 'fullName', 'gender' etc is a node which contains reference to the actual data.

The other main nodes such as 'Chats', 'Tasks' and 'Friends' follow the same structure'.

8.2 Login Page

In the login page, the title, logo, username and password fields, login button, forgot password button and register button can be seen. *Code segment in Appendix 12.2*



An on click listener is attached to the login button. The values from the user text fields are retrieved in String format. There is some validation to ensure the text fields are not empty with the 'TextUtils.isEmpty(String object)' method. If they are empty, user is notified and prompted to enter again.

```
if (TextUtils.isEmpty(txt_email) || TextUtils.isEmpty(txt_password)){
    Toast.makeText(LoginActivity.this, "All fields required", Toast.LENGTH_SHORT).show();
}
```

If the text fields are not empty, the firebase 'sign in' method using email and password is attempted. This is considered a task. And if this task is successful, it would go to the next required page/class.

```
auth.signInWithEmailAndPassword(txt_email, txt_password)
```

The class also has functionality for opening the 'Reset Password' page and 'Register' page by clicking their respective buttons which have on click listeners attached to them.

8.3 Register Page

When the signup button is clicked, the values are retrieved from the text fields. If empty, it is handled through validation as before. For example, if password is less than 6 characters, it will not be accepted. *Code segment in Appendix 12.6.3.*

```
else if (txt_password.length() < 6){
    Toast.makeText(RegisterActivity.this, "password should be more than 6 chars", Toast.LENGTH_SHORT).show();
}
```



If validation is successful, my method 'register()' is called, taking in required login details as arguments. This method creates a user with the Firebase email and password concept, as mentioned before

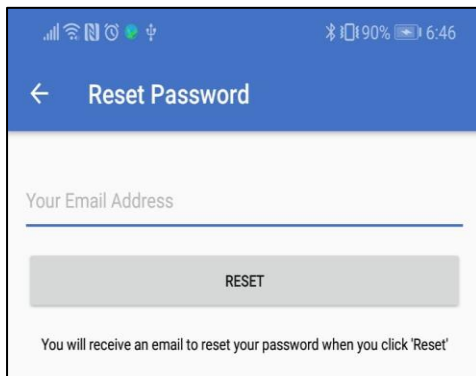
```
auth.createUserWithEmailAndPassword(email, password)
```

The user needs to be saved onto the Firebase database. A reference is first made to the Firebase hierarchy and a specific node "Users". And the user details are added to the node under its unique identifier which itself also is a node. Memory score is initially set to 0.

```
reference = FirebaseDatabase.getInstance().getReference("Users").child(userid);
User = new User(userid, newUsername, "default", dob, newFullName, "Male",
    username.toLowerCase(), new MemoryScore(0,0,0));
reference.setValue(user)
```

On completion, user is directed to the Memory Quiz page – this will be visited later.

8.4 Reset Password Page



When the reset button is clicked, the text from the appropriate text field is converted to String and checked it is not empty. If its empty, user will be promoted to enter a valid one. If valid, Java provides firebase functionality to send a reset email password email directly to the email entered with the function to reset password, providing a GUI for it. After doing so, the user is redirected to the login page, where the new password would only be accepted. *Code segment in Appendix 12.4.*

8.5 Memory Quiz Page

Code segment in Appendix 12.5. In the class, the instance variables represent the items in the page. The user is assigned 50 points to start off with which would vary depending on how they answer the questions.

```
private int score = 50;
```

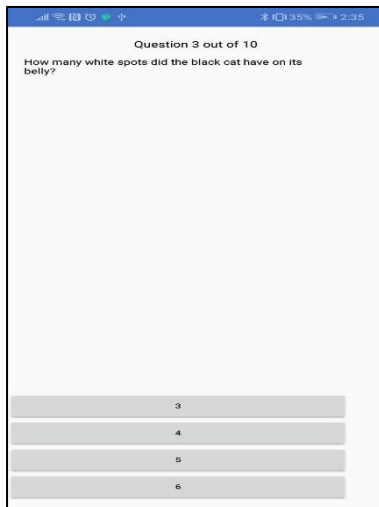
There is also a reference to class QuestionSet which has all questions that the user will be asked and answers for each question.

```
private QuestionSet questionSet = new QuestionSet();
```



Method process(0) is first called with argument 0. The 0 argument refers to the question that the user will see – all questions are in arrays so we start off with index 0 for 1st question. When process is called, it will display or hide the correct questions, fields, buttons and images (when required).

For each question, there is a timer that the question will be displayed for. A 'handler' is used to run the



code after a specific amount of time. For each option that a user clicks, the system checks if the clicked option is the correct answer from the QuestionSet by using the instance method 'getAnswer(int)' to check if they are the same. Whether it's the same or not, the user's points are amended appropriately.

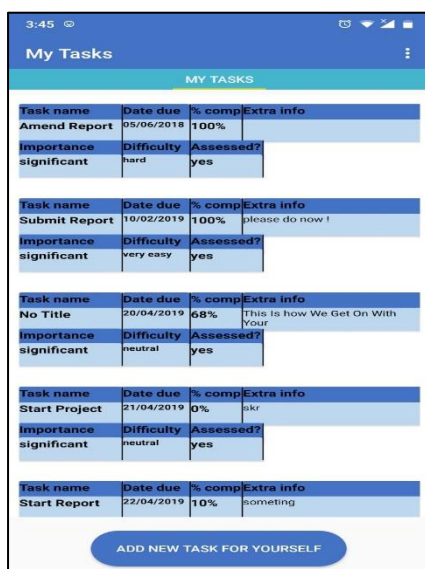
There are a few other instances methods 'updateQuestion', 'updateExtract' and 'updateAnswerOptions' which are self-explanatory.

8.6 Tasks to do Page

Code segment in Appendix 12.6.1. The code is from class 'HomepageActivity.java' which is the page that includes tab layout and view pager to include a fragment. The code sets up the holder for fragments.

There is code from class 'TasksFragment.java', which represents a fragment. Code segment in Appendix 12.6.2.

The most important aspects here are the uses of 'RecyclerView' and 'adapters'. With the recycler view, I am able to add data from Firebase to the recycler view. Initially, I could add one segment of data and display it, for example, one task. I couldn't display other tasks. This was problematic as I needed to display multiple tasks.



After thorough research, I discovered the RecyclerView which 'recycles' data. This allowed lots of tasks to be displayed in the recycler view. Credit to YouTuber: KOD dev.

The method 'fillInUsersTasks' fills in the data needed to display to the user (the tasks). A snapshot is retrieved from the Firebase data and for each node in snapshot, they are converted to the Task object so Java methods can be applied to them.

```
for(DataSnapshot snapshot : dataSnapshot.getChildren()){
    Task task = snapshot.getValue(Task.class);
```

Then the task would be added to an array list if the task is assigned to the logged in user. This needs to be checked as the Firebase data stores all tasks, and different tasks belong to different users – so I need to check for the tasks that belong to the logged in user.

```
if (task.getBelongsTo().equals(fuser.getUid())) {
    usersTasks.add(task);
```

The 'sortTasksyDate' method sorts the tasks by date in ascending order.

Below, I use the adapter variable to call the adapter class 'TasksAdapter.java' which takes the list of tasks and processes it further. Then the recycler view sets the adapter and displays the data to the users.

```
taskAdapter = new TasksAdapter(getContext(), usersTasks);
recyclerView.setAdapter(taskAdapter);
```

Task Adapter

There is code for the TaskAdapter.java class used in the fragment. *Code segment in Appendix 12.6.3.*

One of the instance variables in this adapter class is a list containing Tasks. There is another class inside this class, at the bottom, which refers to the properties in the Layout Resource file task_item which was inflated to the view. This is done so the items can be accessed for each task.

```
public class ViewHolder extends RecyclerView.ViewHolder{
    private TextView taskTitleText; private TextView dateDueBy;
    private TextView percentComplete; ...
```

```
public ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
    View view = LayoutInflater.from(mContext).inflate(R.layout.task_item, parent, false);
```

The method 'onBindViewHolder' adds functionality for each task. When a task is clicked, it will open a new page which displays the single task. The use of method 'putExtra(data)' from class Intent allowed me to display the correct details for the task that was clicked. Through variable 'position', the application will know which task was clicked as each task in the recycler view has a position number.

Task name	Date due	% comp	Extra info
Start Project	21/04/2019	0%	skr

Importance	Difficulty	Assessed?
significant	neutral	yes

Task is set by Admin Ahmed

Update the task below

Percent Complete xy%

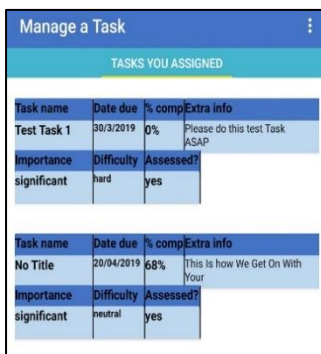
UPDATE

```
@Override
public void onBindViewHolder(@NonNull final ViewHolder viewHolder, int position) {
    final Task task = mTasks.get(position);
    viewHolder.taskTitleText.setText(task.getTaskTitle());
    ...
```

View holder holds the view of the rows of tasks. There's an on click listener to the 'item view' of the view holder so that when the view/row is clicked, the page with the correct details will be displayed. The page that is opened is 'Manage a task'.

```
viewHolder.itemView.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent(mContext, ManageATaskActivity.class);
        intent.putExtra("taskTitle", viewHolder.taskTitleText.getText().toString());
        intent.putExtra("dateDueBy", viewHolder.dateDueBy.getText().toString());
```

8.7 Manage a Task Page

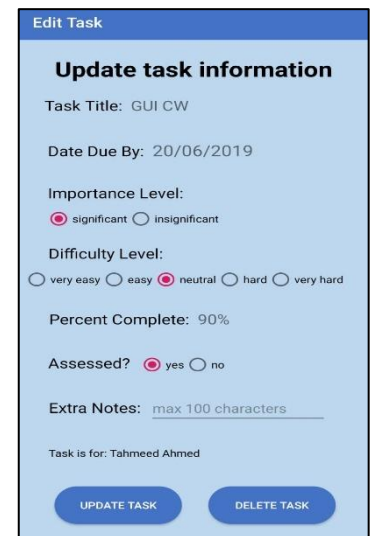


Manage a Task			
TASKS YOU ASSIGNED			
Task name	Date due	% comp	Extra info
Test Task 1	30/3/2019	0%	Please do this test Task ASAP
Importance	Difficulty	Assessed?	
significant	hard	yes	
Task name	Date due	% comp	Extra info
No Title	20/04/2019	68%	This is how We Get On With Your
Importance	Difficulty	Assessed?	
significant	neutral	yes	

Code segment in Appendix 12.7. The method 'getStringExtra("string")' retrieves the data passes onto it from the previous intent activity that matches the same string in the argument. This is an effective way to pass data.

This page allows the user to update the percent complete field for the task. This is done

so the application and its algorithms can track the progression. There is also validation to ensure that the user only enters correct values e.g. 10%, 100%, 5% etc. When everything is validated, a boolean variable 'proceed' will be set to true. In that case, the value is updated to Firebase using the Java functionality. The method 'setValue(data)' updates or inserts to the nodes of the firebase structure, where the reference is set to.



Edit Task

Update task information

Task Title: GUI CW

Date Due By: 20/06/2019

Importance Level:
☒ significant ☐ insignificant

Difficulty Level:
☐ very easy ☐ easy ☒ neutral ☐ hard ☐ very hard

Percent Complete: 90%

Assessed? ☒ yes ☐ no

Extra Notes:

Task is for: Tahmeed Ahmed

UPDATE TASK **DELETE TASK**

```
reference.child(taskid).child("percentComplete").setValue(percCompAnswer.getText().toString());
```

8.8 Navigation Drawer



Code segment in Appendix 12.8.

Fragments are used because I want to show the navigation drawer without opening a new page so it would be displayed on top of the current page. Fragments make this possible.

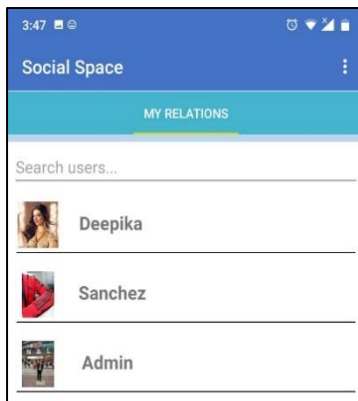
'onCreateOptionsMenu()' method inflates the required Layout Resource file that contains the buttons, the text and icons for each option present to the user.

'onNavigationItemSelectedListener()' method processes which button the user clicked. Each button/item has an individual ID. And depending on that ID, it opens the

appropriate page.

8.9 Social Space

The social space class uses tab layouts and view pagers the same way as mentioned before to utilise Fragments. It uses a different fragment instead, 'SocialSpaceRelationsFragment()'. Code segment in Appendix 12.9.1.



As before, I have included a recycler view to show more than one person a user can message. The 'fillInUsersRelationsList()' gets snapshot of the data from the friend's nodes. For each user in the nodes, it checks if that user and the logged in user are friends. If they are friends, then an array that should contain all their friends will add the user to its list.

```
FriendRequest request = ds.getValue(FriendRequest.class);
if (request.getAFollowedByB().equals("true") || request.getAFollowsB().equals("true")){
    usersRelationsList.add(ds.getKey());
}
```

The 'readUsers()' function uses a for loop to iterate through all users in the Firebase data and checks if any are friends with the logged in user. If so, they are added to the list of users to display.

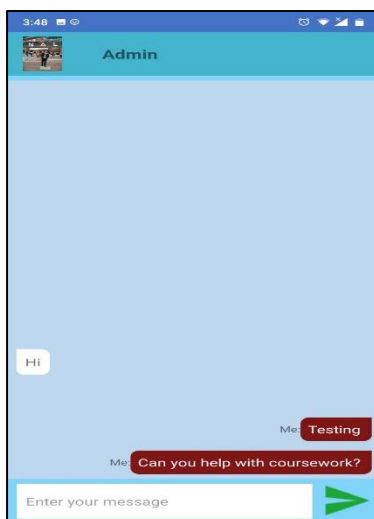
Finally, an adapter is created and set to the recycler view to display users.

```
if (!user.getId().equals(firebaseUser.getId()) && usersRelationsList.contains(user.getId())) {
    mUsers.add(user);
}
```

Social Space Adapter

Code segment in Appendix 12.9.2. The purpose of this adapter is to add functionality to the rows in the recycler view. For each position, the users have a username and a profile image which is indicated from the 'ViewHolder' class. The 'onBindViewHolder()' method sets the username and images for each of them with an on click listener to open the messages page when clicked.

8.10 Messages Page



Code segment in Appendix 12.10. There are a few instance variables to refer to items such as image views, text views etc. There is also a reference to the intent which is needed to retrieve important data from previous activities such as user's ID.

There is an on click listener to the send message button. When ever it is clicked, if the message text field is not empty, 'sendMessage()' method is called. This takes in required data such as who the sender and receiver are and what the message is. A hash map is used to save this data to the Firebase database.

```
reference.child("Chats").push().setValue(hashMap);
```

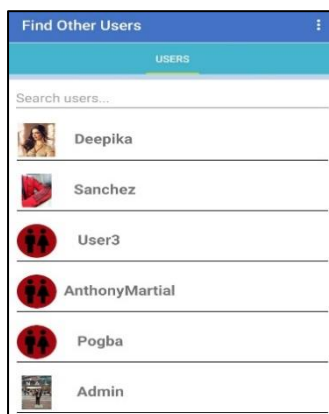
Under the overridden 'onDataChange()' method, user's username and profile picture is updated. This method is called when ever there is a change in the Firebase database reference or when the reference is first accessed. When ever a message is sent by person A, the message is stored on the database. This is considered a change in the database and so this method is invoked which invokes the 'readMessage()' method which would display the sent message.

8.11 Find People Page

Code segment in Appendix 12.11.1. As done before, tab layout and view pager are used for including fragments. The fragment used is 'UsersFragment.java'. *Code segment in Appendix 12.11.2*

I have created a recycler view and an adapter. There is a text field 'search_users' which has an 'text change' listener.

```
search_users.addTextChangedListener(new TextWatcher() {  
    @Override  
    public void onTextChanged(CharSequence charSequence, int start, int before, int count) {  
        searchUsers(charSequence.toString().toLowerCase());  
    }  
})
```

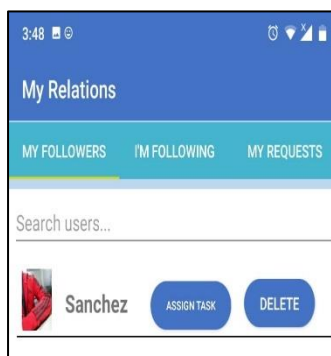


When the user types something, the 'searchUsers()' method is called. It queries user's username that begin with what the user typed and displays them immediately. 'readUsers()' method behaves similarly if not the same as the ones from before.

8.12 View Relations Page

Code segment in Appendix 12.12.1. In this class, there are 3 fragments being added to the view pager. One fragment shows all people that follows the logged in user, another shows all people that the logged in user is following and the last fragment displays any requests that a person might have sent to the logged in user. The tab layout consists of the 3 different tabs which the user can switch between.

Users Followers Fragment and Adapter



Code segment in Appendix 12.12.2. This is the same as other classes in terms of adding people to the list to display. Same method is used using adapters and recycle viewers, with a few variations. The most notable one is getting the snapshot from Firebase and getting a 'FriendRequest' (class) object equivalent. This object uses the instance method 'getAFollowedByB()' to see if the person in the loop being checked follows the logged in user or not. If so, they are added to the list and is then displayed with the recycler view with the adapter.

The adapter: *Code segment in Appendix 12.12.3*

As done before, under 'onBindViewHolder()' method, each person gets added a username and profile image. The code for this is not included here as it has been already shown for previous pages.

Within the same method, the user is able to choose whether or not to delete a person by clicking on their respective item view in the recycler view. By doing so, the application displays an Alert Dialog box. This gives the user the option to either 'delete' the follower or 'cancel' and go back. If delete button is chosen, the application sets the nodes to determine if one is a follower of another in the Firebase database to false.


```
reff.child(firebaseUser.getUid()).child(holder.id).child("AFollowedByB").setValue("false");
reff.child(holder.id).child(firebaseUser.getUid()).child("AFollowsB").setValue("false");
```

Deleting a follower is one option that the user can do. The other option is to assign them a task which is done so by clicking the relevant button. By clicking, the class 'TaskAssignActivity.java' is invoked.

Task Assign Page

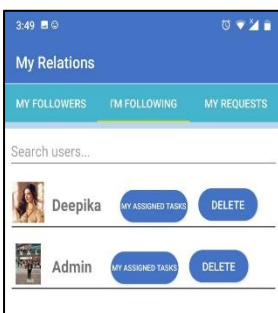
Code segment in Appendix 12.12.4. This class has instance variables referring to all text fields, buttons, radio buttons etc. There is also a 'date picker dialog' which allows users to select dates rather than type one – ensures all dates are valid.

After user has entered in the details for the task that they are setting and clicked the submit button, there are a few validations checks to ensure everything is correct. For example, task title shouldn't be more than 30 characters or empty, if a date hasn't been selected yet, if the percentage of task completion is in the correct format or not and if the extra notes is more than 100 characters which is not valid.

After validation, if all entries are correct, the task is stored on the Firebase database with all the necessary information.

```
DatabaseReference reference = FirebaseDatabase.getInstance().getReference("Tasks");
HashMap<String, String> hashMap = new HashMap<>();
hashMap.put("assessed", RGAAssessedAns.getText().toString()); hashMap.put("belongsTo", intent.getStringExtra("followersId")); ...
hashMap.put("taskTitle", taskTitleAns.getText().toString());
reference.push().setValue(hashMap);
```

Users Following Fragment and Adapter



Code segment in Appendix 12.12.5. Similar code is used here as before. When getting the 'FriendRequest' (class) object equivalent, the instance method 'getAFollowsB()' is used this time as opposed to 'getAFollowedByB()' method. The used instance method checks if the logged in user follows the other person. If so, the id/key of the node in which the user belongs to is added to the required list.

Then 'readUsers()' method will add users to a new list and this list of users gets sent to the adapter class to be dealt with. The adapter: *Code segment in Appendix 12.12.6.*

The adapter has many similar functions as before, which will be left out here. But to briefly mention, it adds, to each user, username and profile picture, alert dialog to delete with different text etc.

There is a new feature where if the view holder (person/row) is clicked, it opens a new activity displaying all tasks that the person who was clicked on, has assigned the logged in user. This activity is 'TasksAssignedActivity.java'

Tasks Assigned Page

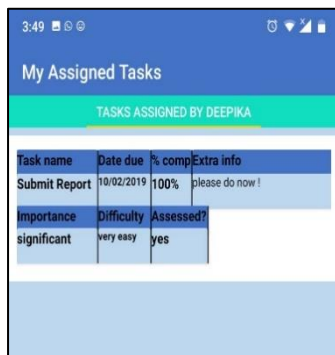
Code segment in Appendix 12.12.7. This class uses fragments also. I had to use 'Bundles' to send a and retrieve data from one activity to another. Previously, I sent extra data to intent (activities) through the method 'putExtra(data)'. This is not possible for fragments. Data can be sent to fragments through

bundles, however. The task setter's ID is sent to be used in the fragment. This class uses the fragment 'TasksAssignedFragment.java'. Code for it: *Code segment in Appendix 12.12.8.*

Method 'fillInUsersAssignedTasks()' retrieves a snapshot from Firebase data reference, converting to 'Task' (class) object equivalent and checks whether the task is assigned to the logged in user and that the task was set by the person (task setter) who was clicked from the previous page. The task setters ID is retrieved by:

```
taskSettersId = getArguments().getString("taskSettersId");
```

The tasks in firebase has a node 'setBy' which has the ID of the user it was set by. If this id on the node matched the string that contains the desired ID, the task is added to a list which is sent to an adapter.



A handler is used to wait for a few seconds before the recycler is set. This is because the data needs to be retrieved from Firebase and unfortunately it isn't instantaneous. There is also a sort method which sorts the tasks by date – same as before.

The adapter used is 'TasksAdapter.java' which has been described before. The same one is used as this adapter just takes in a new list of tasks and displays them with appropriate functionality.

Relation Request Fragment and Adapter

Code segment in Appendix 12.12.9. This fragment uses recycler view and an adapter similarly. The method 'getUsers()' gets a reference to the Firebase database nodes 'Friends'. A FriendRequest (class) object equivalent is retrieved for each key in the snapshot. The instance method 'getAFollowsB()' will return the status of the relation between the 2 people. If its equal to 'pending', then a request has been sent and this user will be added to a list which will be used in the adapter to display (RelationRequestAdapter.java).

```
if (friendRequest.getAFollowsB().equals("pending")){
```

The adapter: *Code segment in Appendix 12.12.10.*

The difference in this adapter to others is in the 'onBindViewHolder()' method. Whenever a person/row is clicked on, an Alert Dialog box is shown to the user in which they are given two options. They can either accept the friend request or decline it.

If accepted, the node in the Firebase reference is set to true:

```
reference.child("AFollowsB").setValue("true");
requestersReference.child("AFollowedByB").setValue("true");
```

If declined, the values are set to false:

```
reference.child("AFollowsB").setValue("false");
requestersReference.child("AFollowedByB").setValue("false");
```

8.13 Suggested Tasks Page

This page suggests tasks using my 'Manage Workload' algorithm to suggest tasks for users to do intelligently. *Code segment in Appendix 12.13.1.*

There is a new fragment, called 'SuggestedTasksFragment.java'. It is added to the page's view page adapter.

Suggested Tasks Fragment – Manage Workload Algorithm

```
public List<Task> intelligentTasksSort(){ ...
    for (Task task : usersTasks){
        int score = 0; ... int ran = new Random().nextInt(2); ...
        if (task.getAssessed().equals("yes")){
            if (ran==0) score = score + 4;
            else score = score + 5; ...
```

I have created a for loop that iterates through all the tasks a user has. Each task is assigned a score, starting with 0. The score will increase depending on its variables. There is also an object of class Random that returns random numbers to allow fluctuations in the scores. If the task is an assessed one, it will get a +5 or +4 score.

```
if (dateTaskDueBy.compareTo(todaysDate) > 0){
    diff = dateTaskDueBy.getTime() - todaysDate.getTime();
    diff = diff / 1000; diff = diff / 86400; diff = diff + 1;
    if (diff <= 3) score = score + 5;
    else if (diff >= 4 && diff <=7) score = score + 4;
    else if (diff >= 8 && diff <=10) score = score + 3;
    else if (diff >= 11 && diff < 14) score = score + 2;
    else score = score + 1; ...
```

The code segment above caters for the days until the deadline for the task. I first check that the task's deadline is still upcoming, using the 'compareTo()' method. 'getTime()' is used on the dates (deadline and current day) to get the difference in days between the 2 dates. The longer the days until the deadline, the less 'urgent' it is so therefore, lower scores are added.

```
if (task.getDifficulty().equals("very easy")) score = score + 1;
else if (task.getDifficulty().equals("easy")) score = score + 2;
else if (task.getDifficulty().equals("neutral")) score = score + 3;
else if (task.getDifficulty().equals("hard")) score = score + 4;
else if (task.getDifficulty().equals("very hard")) score = score + 5; ...
```

The code segment above adds scores depending on the difficulty level of the task. The harder it is, the more 'urgent' it is as more time is likely needed to be spent on it. Therefore, higher scores are added for harder tasks. The same structure is followed for importance level.

```
String percentComplete = task.getPercentComplete();
char[] charray = percentComplete.toCharArray();
String percCompNumb = "";
for (int i=0; i<charray.length-1; i++) percCompNumb = percCompNumb + charray[i];
int x = Integer.parseInt(percCompNumb);
if (x <= 20) score = score + 5;
else if (x > 20 && x <= 40) score = score + 4;
else if (x > 40 && x <= 60) score = score + 3;
else if (x > 60 && x <= 80) score = score + 2;
else if (x > 80 && x <= 99) score = score + 1;
taskScores.add(score); ...
```

In the last part of the algorithm, the percent complete value is retrieved. It is in the form of xyz%. To operate on this, the '%' character from the value is removed by using a character array and removing the last character. Scores are added to the task score depending on how much is complete. If its less than 20%, then the score is increased by 5. The more it is complete, the lower score it gets, which can be seen in the code segment. In the end, the total score is totalled and added to an array list of scores for all tasks.

```
for (int j = 0; j<usersTasks.size(); j++) unsortedMap.put(usersTasks.get(j), taskScores.get(j));
Map<Integer, String> sortedMap = sortByValues(unsortedMap);
Set set = sortedMap.entrySet();
Iterator iterator = set.iterator();
while (iterator.hasNext()){
    Map.Entry me = (Map.Entry)iterator.next();
    usersSuggestedTasks.add((Task)me.getKey());
}
Collections.reverse(usersSuggestedTasks);
return usersSuggestedTasks;
```

Suggested Tasks			
SUGGESTED TASKS			
Task name	Date due	% comp	Extra info
Start Report	22/04/2019	10%	something
Importance	Difficulty	Assessed?	
significant	hard	yes	
Task name	Date due	% comp	Extra info
Start Project	21/04/2019	0%	skr
Importance	Difficulty	Assessed?	
significant	neutral	yes	
Task name	Date due	% comp	Extra info
Change Section A	14/07/2019	60%	something else
Importance	Difficulty	Assessed?	
significant	very hard	yes	
Task name	Date due	% comp	Extra info
Amend Report	05/06/2018	100%	
Importance	Difficulty	Assessed?	

The array list is then used to store the scores into a map, using keys (tasks) and values (scores). The map is sorted and converted into an array list. This sorted list of tasks based on scores is then returned to a 'TasksAdapter.java', same as before, to display the tasks. The method used to sort the map: *Code segment in Appendix 12.13.2.*

8.14 Deciding Whether to Send a Notification

In this section, I will describe how I implemented the decision making of whether to send a notification to the user when a task deadline is approaching.

```
Calendar calendar = Calendar.getInstance();
calendar.set(
    calendar.get(Calendar.YEAR), calendar.get(Calendar.MONTH), calendar.get(Calendar.DAY_OF_MONTH), 15, 34, 0
);
notificationCheck(calendar.getTimeInMillis());
```

An instance of the calendar class was used to set a specific time in which some code can be executed. From the snippet, 'calendar.set()' is used to set a time in which some code is invoked. In the arguments of the method, 3 numbers are taken: 15 (hour, 24-hour format), 34 (minutes) and 0 (seconds). As a result, when the application is running and the time 3:34pm is reached, the method 'notificationCheck()' is called.

```
private void notificationCheck(long timeInMillis){
    AlarmManager alarmManager =(AlarmManager) getSystemService(Context.ALARM_SERVICE);
    Intent intent = new Intent(this, Notification.class);
    PendingIntent pendingIntent = PendingIntent.getBroadcast(this, 0, intent, 0);
    alarmManager.setRepeating(AlarmManager.RTC_WAKEUP, timeInMillis, AlarmManager.INTERVAL_DAY, pendingIntent); }
```

An alarm manager is needed to handle the times when the method is called. An intent is used to call a class where the real functionality can take place – Notification.java. There is a pending intent – this is used as the intent is 'pending' and will run when needed. The 'alarmManager.setRepeating()' method ensures that the application will become activated even when the phone is on sleep mode with the 'RTC_WAKEUP'. The method also ensures that this 'notificationCheck()' is called every day, using the 'AlarmManager.INTERVAL_DAY'.

Notification.java

```
final List<Task> deadlinesUrgent = new ArrayList<>();
final List<Task> deadlinesOneWeek = new ArrayList<>();
final List<Task> deadlinesTwoWeeks = new ArrayList<>();
```

I created 3 array lists. They contain tasks which have an upcoming deadline. They are sorted into arrays depending on how many days until the deadline. If it's very close, it will go to the urgent one. If its in one week, it will go to the one-week list etc. For each task, the number of days is calculated and stored accordingly. Below shows by how many days, represented by variable 'diff'. It is important to note that just because a task is in one of these lists, it doesn't mean a notification will be sent for it. It depends on the memory score.

```
if (diff > 0 && diff <= 3) deadlinesUrgent.add(task);
else if (diff == 7) deadlinesOneWeek.add(task);
else if (diff == 14) deadlinesTwoWeeks.add(task);
```

The Users memory score is then retrieved.

```
for (Task urgentTask : deadlinesUrgent)
    if (memoryScore[0] > 60) notifyUrgent(context, urgentTask.getTaskTitle(), urgentTask.getDateDueBy());
for (Task oneWeekTask : deadlinesOneWeek)
    if (memoryScore[0] > 70) notifyOneWeek(context, oneWeekTask.getTaskTitle(), oneWeekTask.getDateDueBy());
for (Task twoWeeksTasks : deadlinesTwoWeeks)
    if (memoryScore[0] > 80) notifyTwoWeeks(context, twoWeeksTasks.getTaskTitle(), twoWeeksTasks.getDateDueBy());
```

In this segment, I iterate through all the tasks for each list. For each list, a notification is sent if the user's memory score is greater than a specific amount, dependant on the list. For example, for the urgent list of tasks, if the user's memory score is greater than 60, then the user should receive a notification. This is because the deadline is approaching and a memory score greater than 60 is not very good. So, it might be best to send a reminder just in case. 60 as a value was chosen to cater for people who have a really bad score, around 75+ and for those who have a decent score but still might be likely to miss it.

For the list containing tasks with a deadline of one week, a notification is sent if the memory score is greater than 70, instead of 60. This is because there is roughly a week until the deadline. Users with a low score are more likely to have remembered about it. Potentially, users with a score of 70+, are more likely to forgot. Sending a reminder to them is beneficial as there is only a week left for the task.

For the last list, which contains tasks with a deadline of two weeks, a notification will be sent if the users memory score is 80+. This is because a score of 80+ means that the user regularly forgets to complete tasks on time. So, it is definitely worth reminding them 2 weeks in advance as they are much more likely to forget to complete the task on time.

Notifications are triggered with methods 'notifyUrgent()', 'notifyOneWeek()' and 'notifyTwoWeeks()'. An actual notification sent with the following code:

```
NotificationCompat.Builder builder = new NotificationCompat.Builder(context, "channelID")
    .setSmallIcon(R.drawable.ic_launcher_foreground)
    .setContentTitle("Task '" + taskTitle + "' needs completing")
    .setContentText(...)
    .setPriority(NotificationCompat.PRIORITY_DEFAULT);
NotificationManagerCompat manager = NotificationManagerCompat.from(context);
manager.notify(random.nextInt(1000000), builder.build());
```

The notification needs to be built. An icon is set which the user will see. The title is also set, telling the user a deadline is approaching. The content is also set, giving more information on by how many days the

task is due in, with the 'setText()' method. This varies depending on which method it is in. If urgent, then:

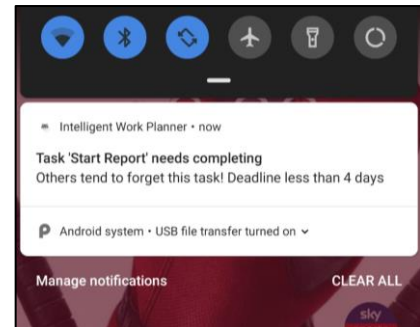
```
setText("Task's deadline less than a week !")
```

If in one week:

```
setText("Task's deadline less than a week !")
```

This is done similarly for when tasks are due in two weeks.

All notifications are assigned an ID. This is for the phone to identify and manage all notifications on the phone. Notifications should have a different ID; otherwise they would override each other. So, in case more than one notification was to be sent, a random number is assigned using Java's Random class.



8.15 K-Nearest Neighbour Algorithm

The initial idea was to use K-Nearest Neighbour (KNN) algorithm to decide whether to send a notification. The algorithm was built and developed in Java, working perfectly fine. Unfortunately, I was unable to incorporate the required data from the database to my KNN algorithm. This was explained in detail further down.

K-Nearest Neighbour Algorithm Implementation

```
class KNearestNeighbour implements Comparable<KNearestNeighbour>{
    int value; // 0 = sent, 1 = notSent
    double x, y;
    double distance; // distance from the new input

    KNearestNeighbour(int value, double x, double y) {
        this.value = value;    this.x = x;    this.y = y;    }

    KNearestNeighbour(double x, double y) {
        this.x = x;    this.y = y;    }

    @Override
    public int compareTo(KNearestNeighbour o) {
        if (this.distance > o.distance) return 1;
        else if (this.distance < o.distance) return -1;
        return 0;
    }
}
```

The code above is the data structure to store the required data. The variable value represents the predetermined class/outcome. The coordinates of the vector space are represented by variables of type Double x and y. Each point has a distance variable, which will be the distance from the new input to be tested.

There are 2 constructors. One takes a value/outcome/class as an argument and one doesn't. The first that takes the argument is the 'training phase' and the latter is for testing the class of a new outcome.

There is also an overridden 'compareTo()' method which indicates which point is the furthest away from 2.

```

class Classifier {
    static String classifyNewPoint(KNearestNeighbour[] arr, int n, int k, KNearestNeighbour p) {
        for (int i = 0; i < n; i++)
            arr[i].distance = Math.sqrt(Math.pow((arr[i].x - p.x), 2) + Math.pow((arr[i].y - p.y), 2));
        Arrays.sort(arr);
        int freqOfNearNotificationsSent = 0;
        int freqOfNearNotificationsNotSent = 0;
        for (int i = 0; i < k; i++) {
            if (arr[i].value == 0) freqOfNearNotificationsSent++;
            else if (arr[i].value == 1) freqOfNearNotificationsNotSent++;
        }
        if (freqOfNearNotificationsSent > freqOfNearNotificationsNotSent) return "sent";
        else return "notSent";
    }
}

```

This is a classifier method which classifies the outcome/class of a given input. When explaining the Euclidean algorithm needed for the distance metric, a formula was given. My Java implementation of that formula is:

```
Math.sqrt(Math.pow((arr[i].x - p.x), 2) + Math.pow((arr[i].y - p.y), 2));
```

It iterates through an array containing the previous data needed to test the new input against. Object 'p' is the new input, containing a KNN object with coordinates but no class as of yet.

Variables 'freqOfNearNotificationsSent' and 'freqOfNearNotificationsNotSent' keep a count of the frequency of the classes on the nearest k points, which is also represented by variable k. Then, depending on which variable occurs the most, the corresponding outcome/class is returned in String form.

Problems with my KNN Algorithm

The KNN algorithm works correctly and efficiently. This can be seen in the testing chapter. Unfortunately, I was unable to retrieve the data required in the correct format to use on the algorithm. The application would crash due to an error which I was not able to fix. Had I had more time, I would've researched more into the error and tried fixing it.

8.16 Sending Notifications Based on Similarity

This section describes how I implemented the decision to send a notification based on similar tasks being completed after the deadline. The algorithm needs to check for similar tasks and see if they were completed on time or not.

```

public static List<String> getNamesInPoolOfSimilarTasks(){
    List<String> similarNames = new ArrayList<>();
    similarNames.add("task"); similarNames.add("report"); similarNames.add("coursework"); similarNames.add("cw");
    similarNames.add("project"); similarNames.add("section"); similarNames.add("finance");
    return similarNames;
}

```

This method adds a few common words that describe a few tasks to an array list. This array list is then checked later for tasks containing any of these common words.

```

double taskOccurrence = 0;
double timesSubmittedLate = 0;
String[] arr = taskName.split(" ");

```

I created variables to keep a track of occurrence of tasks in a for loop and how many times they were submitted late. An array is also used to split the task name into words, each in an index. This is to see if the word contains any of the similar/popular task-name words.


```
if (poolOfSimilarTaskNames.contains(arr[i]) || poolOfSimilarTaskNames.contains(arr[i].toLowerCase()) ||
    poolOfSimilarTaskNames.contains(arr[i].toUpperCase())){
```

This checks that the 'pool' of similar task-name word is contained in one of the tasks name, using a for loop. I handled cases where the words might be all lowercase or uppercase.

```
for (Task t : allTasks){
    if (t.getTaskTitle().contains(word) || t.getTaskTitle().contains(wordLowerCase) ||
        t.getTaskTitle().contains(wordUpperCase)) {
        taskOccurrence = taskOccurrence + 1.0;
    }
}
```

This increases the occurrence variable for the task, if it is in the pool of similar tasks.

```
for (Task tt : allTasks){
    if (tt.getTaskTitle().contains(word) || tt.getTaskTitle().contains(wordLowerCase) ||
        tt.getTaskTitle().contains(wordUpperCase)) {
        if (tt.getPercentComplete().equals("100%")){
            if (tt.getCompletionDeadlineStatus().equals("completed after deadline day")){
                timesSubmittedLate = timesSubmittedLate + 1.0;
            }
        }
    }
}
```

If the task is completed, given by the '100%', I check the completion deadline status. If it was after the deadline day, it means it was not completed on time. As a result, the variable representing the number of times it was submitted late is increased.

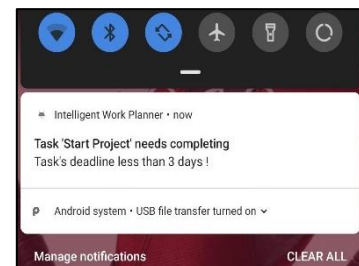
```
double percentage = (timesSubmittedLate / taskOccurrence)*100;
```

Here, I calculate the percentage of times the task was submitted late.

```
if (percentage >= 30) {
    ...
    if (dateTaskDueBy.compareTo(todayDate) > 0){
        long diff = dateTaskDueBy.getTime() - todayDate.getTime();
        diff = ...
        if (diff == 4) sendNotificationOnSimilarity(context, taskName, task.getDateDueBy(), 4);
        else if (diff == 6) sendNotificationOnSimilarity(context, taskName, task.getDateDueBy(), 6);
        else if (diff == 13) sendNotificationOnSimilarity(context, taskName, task.getDateDueBy(), 13); ...
    }
}
```

If the percentage is greater than 30, then a significant amount of people has not completed the task on time. As a result of that, we can conclude that the user might also not complete the task on time. So, a notification is sent via a different method, 'sendNotificationOnSimilarity()'. This is similar to other notification building methods, but with different text information:

```
setContentView("Others tend to forget this task! Deadline less than " + days + " days")
```



9.0 TESTING AND USER FEEDBACK WITH CHANGES

In this section, I will be carrying out tests of different cases of my application and assesses them. I will also find volunteers to test out my system and provide feedback.

9.1 My Testing

Login Page

Test: Check if application only accepts correct login details

Process: I have entered data that isn't stored in the database for users. The screenshot displays login details being entered.

Outcome: After clicking login, the user is notified of the error. A 'toast' is sent to do this. Unable to login with incorrect details so *test was successful*.

Register Page

Test: Check incorrect input data is not accepted

Process: I entered an invalid password of length 1. A toast was shown to user to indicate the password should be longer.

Outcome: User was unable to create an account. After a correct password was used, the user was created which can be seen on the right – a screenshot of the user node stored on the Firebase database. *Test was successful*.

Memory Quiz

Test: Check if user's memory score is correctly calculated and updated to Firebase database

Process: I have completed the quiz for the new test user. I chose random answers, taking into account which ones are correct.

Outcome: The value of the scores have changed after completing the tests. They were 0 when the user was first created, but now have changed to 50 after the test. *Test was successful*.

Reset Password

Test: To check if user is able to change password by 'resetting'

Process: I have opened the reset password page, entered the email and checked to see if the reset password has been sent to the email.

Outcome: The reset password email was sent. I was able to change the password. *Test was successful*.

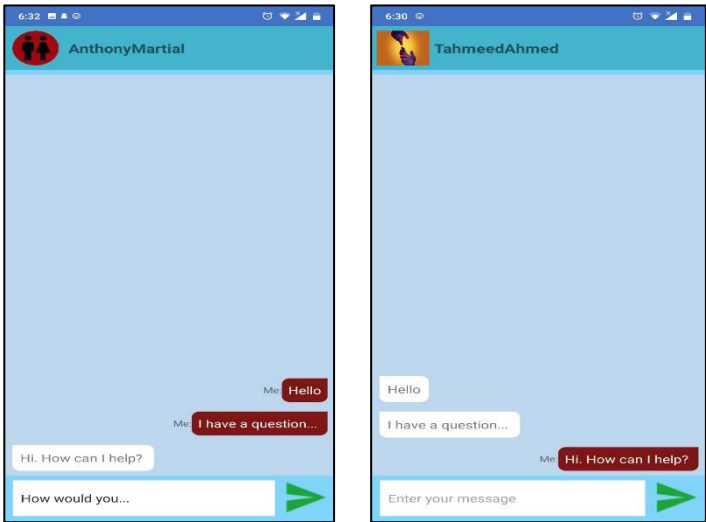
Social Space

Test: Check if person B receives message from person A

Process: I will choose a random person and send a message. Then I will log in to the other person to see if they have received the message.

Outcome: Message sent was received by the recipient. The recipient is also able to send a message back which the other person would then immediately see on their chat screen.

Test was successful.

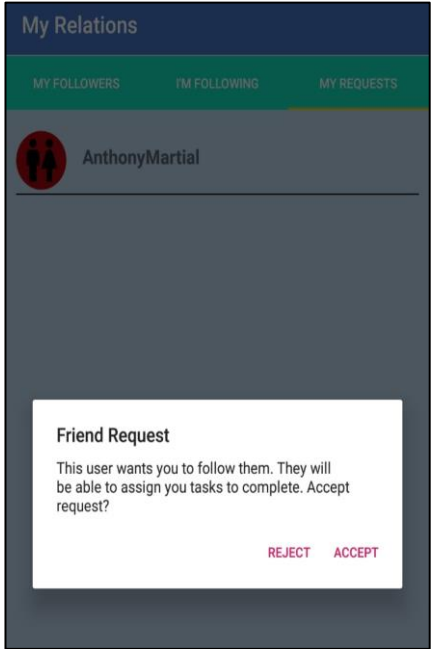


Send Follow Request

Test: Check if sending a request is received and accepted

Process: I will use person A and request person B to follow person A. I will then check the Firebase database to see that the status of the request is pending. Then When person B accepts, this should be reflected on the database.

Outcome: A request to follow was sent and this was shown on the Firebase database. Test was successful.

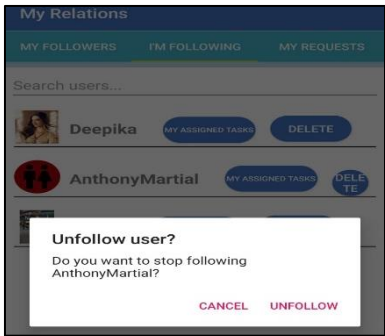


Delete follower

Test: Check if deleting a follower deletes

Process: I will delete a random follower from an account and see if it's reflected on the firebase database

Outcome: User was considered deleted in the firebase and the user who deleted can't be seen in the other user's followers list. Test was successful.

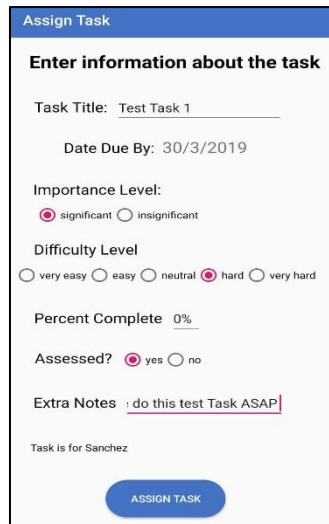


Assign Task

Test: Check if a user is able to assign another user a task

Process: I will use person A's account to assign a task to person B. Then I will log in to person B's account and see if the new task is present.

Outcome: Person B is able to see the task assigned to them. *Test was successful.*



Assign Task

Enter information about the task

Task Title:

Date Due By:

Importance Level:
☒ significant ☐ insignificant

Difficulty Level:
☐ very easy ☐ easy ☐ neutral ☒ hard ☐ very hard

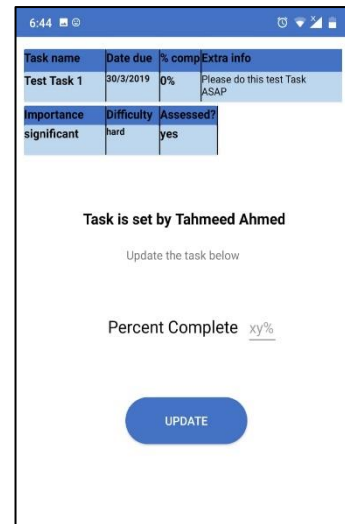
Percent Complete:

Assessed? ☒ yes ☐ no

Extra Notes:

Task is for Sanchez

ASSIGN TASK



Task is set by Tahmeed Ahmed

Update the task below

Percent Complete

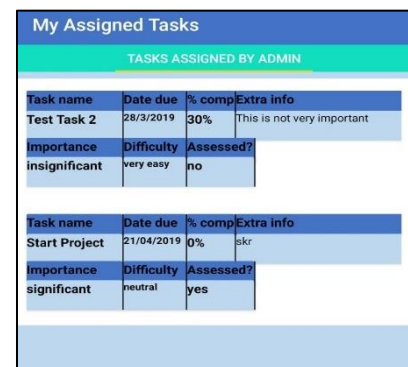
UPDATE

Assigned tasks

Test: Check if all tasks assigned by one user shows

Process: I will go to 'I'm Following' fragment in the 'View Relations' page. A user that has assigned multiple tasks to the logged in user will be chosen. The button 'My Assigned Tasks' will be clicked

Outcome: The page displays all the tasks that the other user has set for the logged in user. *Test was successful.*



My Assigned Tasks

TASKS ASSIGNED BY ADMIN

Task name	Date due	% comp	Extra info
Test Task 2	28/3/2019	30%	This is not very important
importance	Difficulty	Assessed?	
insignificant	very easy	no	

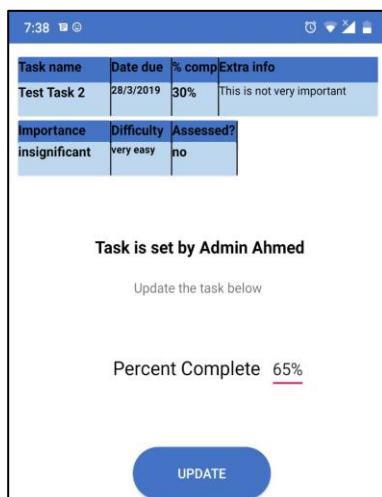
Task name	Date due	% comp	Extra info
Start Project	21/04/2019	0%	skr
importance	Difficulty	Assessed?	
significant	neutral	yes	

Update Task Completion

Test: Check whether user can update how much they have completed a task

Process: I will choose 'Test Task 2' and change the completion percentage. It is currently at 30%. The new percentage should be reflected on the Firebase database.

Outcome: The new percentage can be seen on the application as it was stored on Firebase and retrieved from there. *Test was successful.*



Task is set by Admin Ahmed

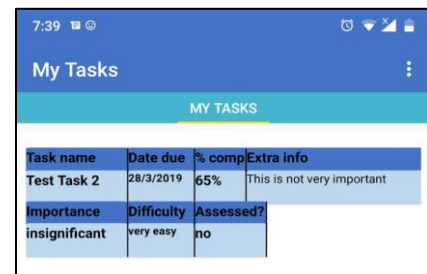
Update the task below

Percent Complete

UPDATE

```

Tasks
- LaCy0jNwR4rt47Rul6
- LaD1GRDvpl7RM8PTY3a
  - assessed: "no"
  - belongsTo: "4Bk6k8n5AWcMtbUsJmyBIBo9ct"
  - dateDueBy: "28/3/2019"
  - difficulty: "very easy"
  - extraNotes: "This is not very important"
  - importance: "insignificant"
  - percentComplete: "65%"
  - setBy: "Um5vH7GnF10qGNVTw3DfaNaFXf1"
  - taskTitle: "Test Task 2"
    
```



My Tasks

MY TASKS

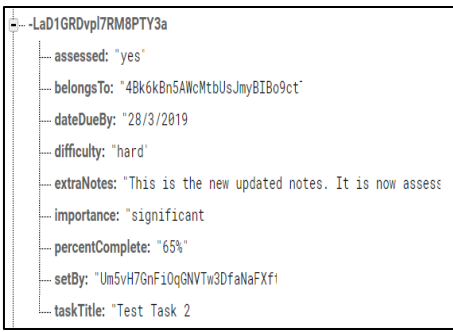
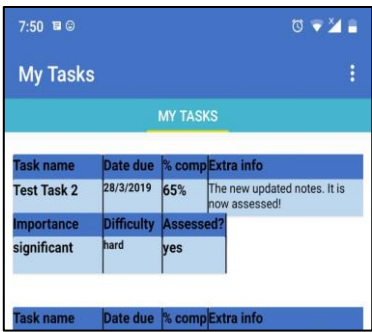
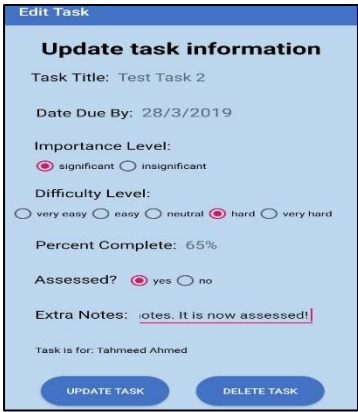
Task name	Date due	% comp	Extra info
Test Task 2	28/3/2019	65%	This is not very important
importance	Difficulty	Assessed?	
insignificant	very easy	no	

Edit task

Test: Check edits a task setter edited are shown to users

Process: As a task setter, I will update a task (change to an assessed task) and then log back into the user which the task belongs to and check if the updated information is present.

Outcome: The edited task information was shown in the user’s tasks section. User can see the task is now ‘assessed. *Test was successful.*

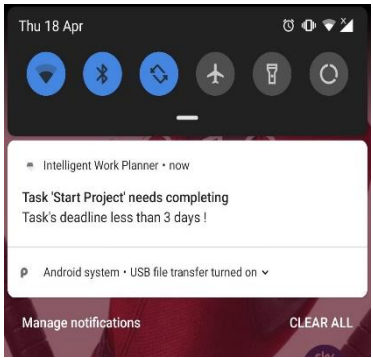


Receiving Notification at Specific Time

Test: Check if notification is received at a specific time per day, when memory score is high

Process: I will have a task, ‘Start Project’ assigned to a user with an upcoming deadline in the next three days. The user will also have a high memory score of 62. This test will be done whilst the application is not running,

Outcome: At the set time, the required method was called, and the notification was sent to the notification tray on the phone. *Test was successful.*

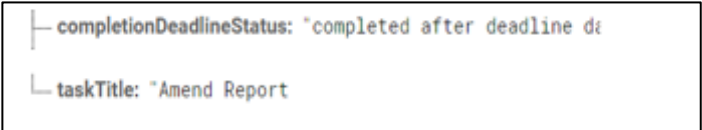
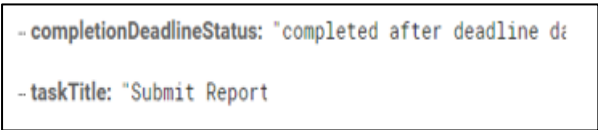
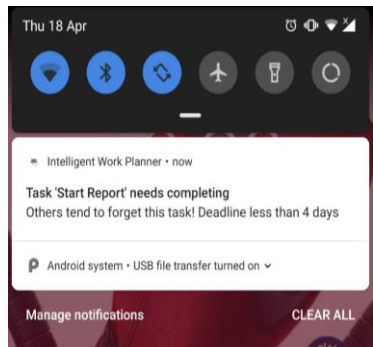


Receiving Notification Based on Similarity

Test: Check if notification is received for a task that is similar to another which people regularly don’t complete in time.

Process: I will have a few completed tasks in the database, with similar names ‘Amend Report’ and ‘Submit Report’, that were not completed on time. Then a new task with a similar name ‘Start Report’ will have an upcoming deadline. I will then check if the specific notification is sent because other similar tasks have not been completed in time.

Outcome: At the set time, the notification was sent, stating that similar people also miss tasks like the current one. *Test was successful.*

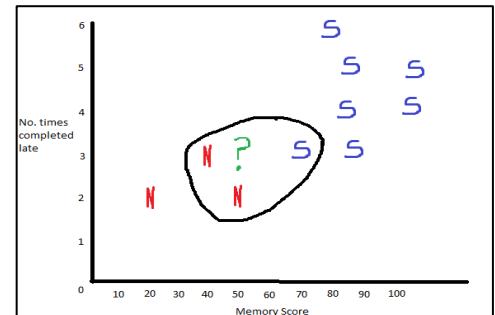


Testing KNN Algorithm

Test: Check if my KNN algorithm implementation works correctly

Process: I will create an array of data-structure type K-Nearest Neighbour' containing a few KNN objects. These objects will be KKN objects, each having a class 'sent' or 'notSent', x and y coordinates, which are memory scores and number of times tasks was submitted late by the task owner, respectively. I will then create a new KNN object/input containing a memory score and a number of times task was submitted late but no class/outcome. This object will be the test, using the method 'classifyNewPoint()'. From the diagram to the right, the class of the new input should be 'notSent'

Outcome: The outcome for the ? with coordinates (50, 3) was 'notSent'. To test correctness, I have also shown the outcome for other coordinates too. KNN algorithm works correctly. *Test was successful.*



```
KNearestNeighbour[] arr = new KNearestNeighbour[10];
arr[0] = new KNearestNeighbour( value: 1, x: 20, y: 2);
arr[1] = new KNearestNeighbour( value: 1, x: 40, y: 3);
arr[2] = new KNearestNeighbour( value: 1, x: 50, y: 2);
arr[3] = new KNearestNeighbour( value: 0, x: 70, y: 3);
arr[4] = new KNearestNeighbour( value: 0, x: 80, y: 4);
arr[5] = new KNearestNeighbour( value: 0, x: 85, y: 3);
arr[6] = new KNearestNeighbour( value: 0, x: 80, y: 5);
arr[7] = new KNearestNeighbour( value: 0, x: 75, y: 6);
arr[8] = new KNearestNeighbour( value: 0, x: 100, y: 5);
arr[9] = new KNearestNeighbour( value: 0, x: 95, y: 4);

KNearestNeighbour input = new KNearestNeighbour(x, y);
String outcome = Classifier.classifyNewPoint(arr, n: 10, k: 3, input);
System.out.println("The class of input with xy (" + input.x + ", " + input.y + ") is: " + outcome);
```

C:\Users\User\Documents>java KNN 50 3
The class of input with xy (50.0,3.0) is: notSent

C:\Users\User\Documents>java KNN 10 2
The class of input with xy (10.0,2.0) is: notSent

C:\Users\User\Documents>java KNN 90 5
The class of input with xy (90.0,5.0) is: sent

9.2 User Feedback with Changes

After my testing phase, I had decided to find volunteers to test my application. It helped get an insight into what they think of it, how easy it was for them to use and how I could improve.

- One feedback was that when opening certain pages such as 'View Relations', it took too long for the data to display. I've implemented by application using several Java handlers to delay the execution of some code. This is because the data needs to be retrieved from Firebase into, for example, array lists to display. Had I not used handlers, empty array lists would be passed on to adapters and no users will be displayed. Then the array list would get data added to it, but the adapter had already been set so the new data was not shown. And by delaying the execution of the adapter for a few seconds, it allowed the array list to be filled with data and then be passed on to the adapter once the timer has finished. But this method has been overly used through out many pages and my testers did not like it at all. As a result of this, I had to discover another way to get around this problem. I then decided to fill in all the data required for all pages, whether user may need it or not, immediately when the



application is opened. There is a loading page which stays for 3-4 seconds whilst all needed data is retrieved from Firebase. A class called 'ProcessDataActivity.java' handles all data retrieval. Once the 3-4 seconds is complete, only then can the user go to their desired pages. As soon as they go to their desired pages, the data is immediately shown which is great for the users. To the right is the loading screen I created.

- One tester suggested that there shouldn't be just only way to go to messages. Previously, before changes, the only way to message someone was to go to 'Social Space' page, find the user and click on it. This is not a problem, but the tester felt that if you are in the 'View Relations' page, you should be able to go directly to the chat from there. As a result, I made it that you can click on the item view (user/row) and that would lead straight to the chat. There is no message button, but now, if you click on the background, it directly goes to the chat. The code for this is below:

```
holder.itemView.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent(mContext, MessageActivity.class);
        intent.putExtra("userid", user.getId());
        mContext.startActivity(intent);
    }
});
```

- Many testers like the fact that on the 'Find People' page, you are able to type up letters and the page will display names starting with those letters. It was very convenient. But this feature had not been implemented onto other pages such as 'Social Space', 'View Relations' etc. So, I have now included a search box in these pages where the user can now search users by name, rather than having to scroll through the list of names which can be strenuous and time-consuming. The code below for this is for the 'Social Space' page is:

```
EditText search_users; ...
search_users = view.findViewById(R.id.search_users);
search_users.addTextChangedListener(new TextWatcher() {...
    @Override
    public void onTextChanged(CharSequence s, int start, int before, int count) {
        searchUsers(charSequence.toString().toLowerCase());
    }...
private void searchUsers(String s){
    Query query = FirebaseDatabase.getInstance().getReference("Users").orderByChild("search").startAt(s).endAt(s+"\uf8ff");
    query.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
            mUsers.clear();
            for (DataSnapshot snapshot : dataSnapshot.getChildren()){
                User user = snapshot.getValue(User.class);
                assert user != null; assert firebaseUser != null;
                if(!user.getId().equals(firebaseUser.getId()) && usersRelationsList.contains(user.getId()) ) mUsers.add(user); ...
                socialSpaceAdapter = new SocialSpaceAdapter(getContext(), mUsers); recyclerView.setAdapter(socialSpaceAdapter);
            }
        }
    });
}
```

- Testers also wanted an extra functionality. They wanted to have the feature where a user can set a task for themselves. After careful consideration, I decided this is a good idea to let people set personal tasks for them selves also. It doesn't a person of higher authority. A person can set a task such as 'Buy

milk tomorrow'. As a result, on 'My Tasks' page, there is a button which allows you to assign a new task but this time, the 'belongsTo' field is automatically set to the logged in user.

10.0 EVALUATION / EVALUATION

In this section, I will be evaluating and concluding my project. I will be providing my own personal view on how I felt the project was and giving an insight into whether I have been successful in meeting my project requirements and objectives. I will also be describing the problems I had encountered, how I overcame them and further work.

10.1 Meeting Requirements

Requirement	Met?
System must store all tasks for each user e.g. assignment, project etc.	✓
For each task, system should collect information - deadline, difficulty, importance and whether it is assessed or not	✓
System must sort all tasks suggesting which ones to do first	✓
User enters how much they have completed each task (percentage wise) and algorithm for suggesting which ones to do updates	✓
Users can mark off tasks when they have been completed	✓
There should be an algorithm that learns about each user (forgetfulness)	✓
User should receive reminder/notification about outstanding tasks – if algorithm suggests they may need to be reminded	✓
System should send reminders based on the ability and personality of user (in terms of forgetfulness)	✓
For similar tasks, system should use an algorithm to predict whether a person will forget that task based on other similar users	✓
Users enter times they are free, and system suggests when to do each task	✗
Users can choose level of reminders as user preferences	✗
There should be a social page where user can communicate with their task setter	✓
Each user must be identified with a unique username.	✓
User should be able to login with a username and an asterisk-protected password	✓
Incorrect login details should display a popup error message, prompting user to try again	✓
Users need to enter details e.g. name, dob etc, which can later be edited	✓
Task setters should be able to assign tasks to 1 or more other users	✓
Users should be able to receive assignments from multiple task setters	✓
There should be a 1-1 relationship between a user and a task setter	✓
Users should be notified when a new task is assigned for them	✗
System should be a mobile application (android)	✓

Justification of Meeting Requirements and Implementation of Complexities

From the table above, it can be seen that the most important requirements have been met. The application allows teacher-student and employer-employee relations to assign tasks to each other, allowing them to communicate. Intelligent Work Planner effectively manages all of the tasks a person has, suggesting which tasks each specific user should do. All tasks are listed in the order that they should be

attempted. This was done using my 'Manage Workload Algorithm'. Each task has its own contribution on being the most important task that needs to be done as soon as possible. The tasks with the highest contributions are more urgent and suggested to be done first.

Notifications have been implemented in 2 different ways. Notifications/reminders are sent to specific users if they need to be reminded. Each user has a memory score which is always fluctuating, depending on how well the user performs, in terms of completing tasks within the deadlines. Depending on the memory score, the respective may or not get a reminder. If they have a high memory score, it usually means the user tends to not complete tasks on time – so they will receive more notifications, compared to a user with a low memory score, as they tend to perform better.

Notifications are also sent using an algorithm based on similarity. The firebase database has a pool of tasks. IWP contains information about these tasks on which are the most common ones. The tasks that are most common, it is checked if a proportion of them were not completed on time. If so, it is concluded that a few people didn't complete this task on time, so the user might also not complete on time. So, it is worth reminding them.

Problems Encountered, Future Project Work and Improvement

Due to time constraints for this project, 3 requirements were not met. Suggesting a timetable when the user should do the tasks, having user preferences (for level of reminders too) and users being notified when a new task is assigned to them. It currently requires users to check daily for new tasks. If I did have more time, I would have attempted to implement this functionality, as well as the other two mentioned above.

A major problem that I faced was data retrieval from Firebase. Due to the fact that the data has to be retrieved in real time, it can take a while. The data can only be retrieved using a database listener. Whilst the listener is receiving data, the next fragments of code always gets executed. This was a major problem for me because I need the listener to finalise before moving on to the next piece of code. For example, when retrieving tasks from the database to a list via the listener, the next section of code would pass the list of tasks to the adapter for further processing. The problem was that the list would get passed onto the adapter before the listener has completed. So many times, before a task is even added to the list from the listener, the empty list is immediately passed onto the adapter. As a result, no tasks would be shown. I have faced many other similar problems regarding data retrieval from the firebase database. For example, retrieving user information. I was unable to find an 'on-complete-listener' which would have been called once the database listener has been completed. To work around this, I had to use a 'handler' to delay the execution of some code by a few seconds. For example, when retrieving the tasks, I delayed the execution of passing the list to the adapter by three to four seconds. This is poor practice because as the number of tasks increases in the database, three or four seconds might not be long enough to retrieve all data. It would lead to missing tasks. The number of seconds required to wait could increase to more than 10 seconds with lots of data. This would be terrible for user performance. As a result of this, if I was to do a similar project, I would research new ways to retrieve data from the firebase database. Or alternatively, I would use a different database service. Moreover, the same problem was faced when

combining my working K-Nearest algorithm with the firebase data retrieval, which has been explained before.

If I did have more time, there would be room for improvement in many areas. Regarding notifications, I would implement the sending of notifications of events such as a new relation request, being assigned a new task, a new message etc. Task setters should also be notified when one of their followers have finished their tasks. These are important features to include as the user needs to be notified on this whenever these events happen.

As briefly mentioned above, there aren't any user preferences. Initially, the idea was to have a settings page which would allow the user to make a few changes. Some of these were having a level of reminders. This is to prevent the user getting annoyed from too many reminders. Other options would have been available such as the choice to change the application colour scheme, privacy and security settings etc. It would be included if I did have more time. Moreover, people can't view other people's profiles. This could be an issue as people might not be who they say they are. A real profile page would need to be implemented.

Meeting IWP Objectives and my Personal Objectives

The objectives of my application were to provide to my target users in a work-environment, a mobile application to allow them to assign tasks, have the tasks being managed intelligently, send reminders to users when needed and tackle user forgetfulness.

I believe I have met all of these objectives as users can assign tasks to each other and communicate in real time. All tasks are managed with my own algorithm. Each user's memory and forgetfulness are modelled with my concept of Memory Scores, which change depending on how users perform. Based on this memory score and task similarity, reminders are sent intelligently, depending on whether a user needs to be reminded.

Regarding my own personal objectives, I have learnt a lot throughout the duration of this project. I have gained many skills. The most obvious one is the ability to use Android Studio to develop android applications. This is the first time I have done so and would like to do something similar in the future.

Moreover, I have learned to be persistent and patient when developing applications. There have been many times where the written code does not behave as expected. Debugging is a key skill I have gained from this project as a result. In addition to this, I have strengthened my skills in using Java, in conjunction with real time databases.

A major issue for me was time management. A lot has been rushed at the very end, especially nearer to exams. I have learned from this to manage time from an early stage, and always prepare for the unexpected.

Overall Conclusion

To conclude my project, I believe it has been a success as the key functional requirements of Intelligent Work planner have been met. I have gained valuable lessons and skills during the course of this project. User's expectations has been met, after demonstrating the application to many peers.

Development of my application further solidifies my interest in software engineering. This is the field of work I want to get into, learning new development languages and tools. The project has given me

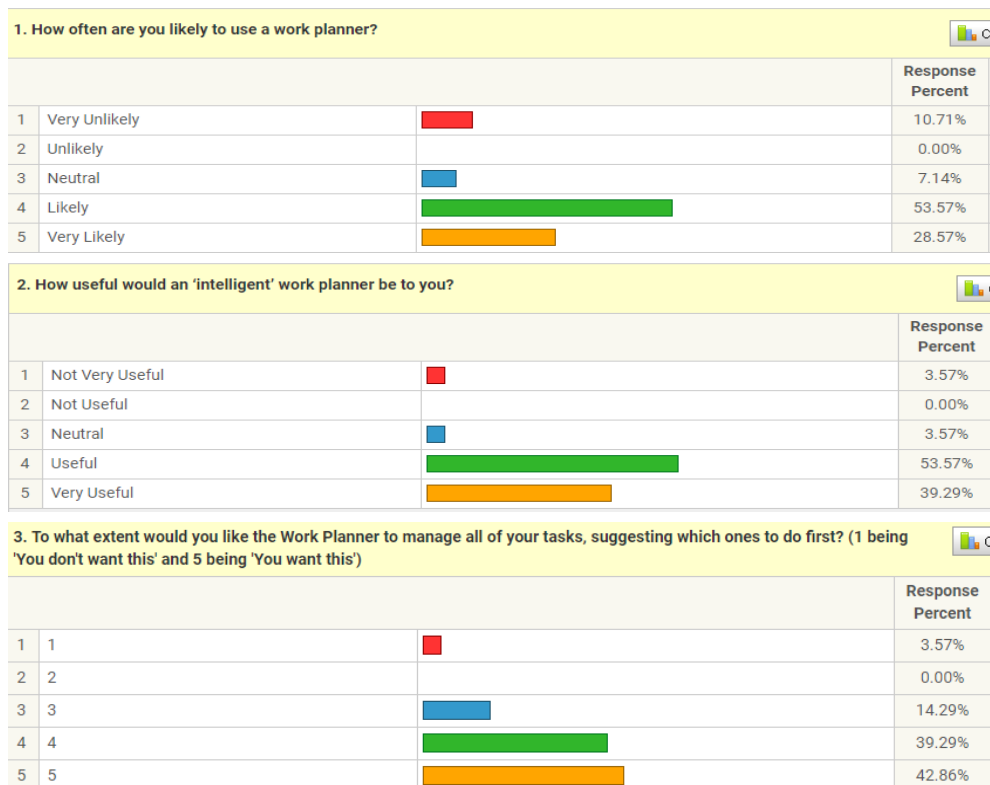
confidence in developing solutions to problems that can help in the real working world. I believe the project has prepared me for what is next to come.

11. REFERENCES

- [1] https://en.wikipedia.org/wiki/Waterfall_model
- [2] <https://searchcio.techtarget.com/definition/Prototyping-Model>
- [3] <https://play.google.com/store/apps/details?id=com.tasks.android>, Stephen Nottage
- [4] <https://play.google.com/store/apps/details?id=com.appxy.planner>, Appxy
- [5] https://en.wikipedia.org/wiki/Decision_tree_learning
- [6][7] <https://www.techwalla.com/articles/advantages-disadvantages-of-decision-trees>
- [8][10] https://en.wikipedia.org/wiki/Logistic_regression
- [9] https://en.wikipedia.org/wiki/Linear_regression
- [11] <https://www.quora.com/What-are-the-advantages-of-logistic-regression>
- [12] https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm
- [13][14] <https://www.fromthegenesis.com/pros-and-cons-of-k-nearest-neighbors/>
- [15] https://en.wikipedia.org/wiki/Euclidean_distance
- [16][17][18] https://en.wikipedia.org/wiki/Naive_Bayes_classifier

12.0 Appendix

12.1



4. The application would remind you about work-related tasks between you and task setter (e.g. submit analysis of findings by tomorrow). Would you also want the application to remind you on trivial daily tasks (e.g. purchase milk today)?				
			Response Percent	
1	Yes	<div></div>	32.14%	
2	No	<div></div>	67.86%	

5. The application will assess your memory, privately to you. Is this an issue?				
			Response Percent	
1	Yes	<div></div>	17.86%	
2	No	<div></div>	82.14%	

6. To what extent is including a small social space to communicate with your task setter appealing?				
			Response Percent	
1	Not Appealing	<div></div>	21.43%	
2	Appealing	<div></div>	78.57%	

7. Applicable to task setters: Are you willing to set up tasks for each user with important information e.g. deadline, prominence, difficulty etc?				
			Response Percent	
1	Yes	<div></div>	87.50%	
2	No	<div></div>	12.50%	

8. Are set reminders unappealing compared to intelligent predicted ones?				
			Response Percent	Response Total
1	No	<div></div>	17.86%	5
2	Yes	<div></div>	82.14%	23

12.2

```
public class LoginActivity extends AppCompatActivity { ...
    loginButton.setOnClickListener(new View.OnClickListener() { ...
        String txt_email = email.getText().toString();
        String txt_password = password.getText().toString();
        if (TextUtils.isEmpty(txt_email) || TextUtils.isEmpty(txt_password)){
            Toast.makeText(LoginActivity.this, "All fields required", Toast.LENGTH_SHORT).show();
        } else {
            auth.signInWithEmailAndPassword(txt_email, txt_password)
                .addOnCompleteListener(new OnCompleteListener<AuthResult>() {
                    ...
                    if (task.isSuccessful()){
                        Intent intent = new Intent(LoginActivity.this, ProcessDataActivity.class); ...
                        intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TASK | Intent.FLAG_ACTIVITY_NEW_TASK);
                    } else Toast.makeText(LoginActivity.this, "Authentication failed !", Toast.LENGTH_SHORT).show();
                }
            }
        }
    }
}
```

12.3

```
public class RegisterActivity extends AppCompatActivity { ...
    signUpButton.setOnClickListener(new View.OnClickListener() {...
        String txt_username = username.getText().toString(); ... String txt_dob = dob.getText().toString();
        if (TextUtils.isEmpty(txt_username) || ... || TextUtils.isEmpty(txt_dob) ){
            Toast.makeText(RegisterActivity.this, "All fields required", Toast.LENGTH_SHORT).show();
        } else if (txt_password.length() < 6 ){
            Toast.makeText(RegisterActivity.this, "password should be more than 6 chars", Toast.LENGTH_SHORT).show();
        } else register(txt_username, txt_email, txt_password, txt_fullName, txt_dob);
    }
}

...

public void register(final String username, String email, String password, final String fullName, final String dob){
    auth.createUserWithEmailAndPassword(email, password) ...
        if (task.isSuccessful()){
            FirebaseUser firebaseUser = auth.getCurrentUser();...
```

```
String userid = firebaseUser.getId();
String newUsername = changeStringFirstLetterToUpperCase(username);
String newFullName = capitaliseFullName(fullName);
reference = FirebaseDatabase.getInstance().getReference("Users").child(userid);
User user = new User(userid, newUsername, "default", dob, newFullName, "Male",
    username.toLowerCase(), new MemoryScore(0,0,0));
reference.setValue(user) ...
    if (task.isSuccessful()){
        Intent intent = new Intent(RegisterActivity.this, MemoryScoreTest.class); ...
    } else Toast.makeText(RegisterActivity.this, "can't register with this email or password", Toast.LENGTH_SHORT).show();
...

```

12.4

```
public class ResetPasswordActivity extends AppCompatActivity {...
    btn_reset.setOnClickListener(...
        String email = send_email.getText().toString();
        if (email.equals("")){
            Toast.makeText(ResetPasswordActivity.this, "Please enter a valid email address", Toast.LENGTH_SHORT).show();
        } else {
            firebaseAuth.sendPasswordResetEmail(email).addOnCompleteListener( ...
                if (task.isSuccessful()){
                    Toast.makeText(ResetPasswordActivity.this, "Please check your email to reset password", Toast.LENGTH_SHORT).show();
                    startActivity(new Intent(ResetPasswordActivity.this, LoginActivity.class));
                } else { ...
                    Toast.makeText(ResetPasswordActivity.this, "Error!", Toast.LENGTH_SHORT).show(); ...
                }
            }
        }
    }
}

```

12.5

```
public class Questions extends AppCompatActivity {...
    private int score = 50; ...
    process(0);
    ...
    private void process(final int qNum){
        questionNumber.setText("Question " + (qNum + 1) + " out of 10");
        a1.setVisibility(View.INVISIBLE); ... a4.setVisibility(View.INVISIBLE); ...
        extract.setVisibility(View.VISIBLE);
        updateExtract(qNum);
        int timeToWait = 0;
        if (qNum==0) timeToWait = 50000; if (qNum==1) timeToWait = 40000; if (qNum==2) ... if (qNum==9) timeToWait = 50000;

        new Handler().postDelayed(new Runnable() {...
            if (qNum==2 || qNum==4 || qNum==5) questionImage.setVisibility(View.INVISIBLE);
            updateQuestion(qNum);
            updateAnswerOptions(qNum);
            a1.setVisibility(View.VISIBLE); ... a4.setVisibility(View.VISIBLE);
        }, timeToWait);

        a1.setOnClickListener(new View.OnClickListener() {...
            if (a1.getText().toString().equals(questionSet.getAnswer(qNum))) score = score - 3;
            else score = score + 2;
            if (qNum == 9) finishUp();
            process(qNum+1); ...
        });

        a2.setOnClickListener(new View.OnClickListener() { ...
            if (a2.getText().toString().equals(questionSet.getAnswer(qNum))) score = score - 3;
            else score = score + 2;
            if (qNum == 9) finishUp();
            process(qNum+1); ...
        });

        private void updateQuestion(int questionNum){
            extract.setText(questionSet.getQuestion(questionNum)); ...
        }

        private void updateExtract(int extractNum){
            extract.setText(questionSet.extracts[extractNum]);
            if (extractNum == 2 || extractNum == 4 || extractNum == 5){
                questionImage.setImageResource(questionSet.getImage(extractNum));
            }
        }
    }
}

```

```
questionImage.setVisibility(View.VISIBLE); ...

private void updateAnswerOptions(int questionNum){
    a1.setText(questionSet.getChoice1(questionNum)); ... a4.setText(questionSet.getChoice4(questionNum));
    ...
}
```

12.6.1

```
...
TabLayout tabLayout = findViewById(R.id.tabLayout);
ViewPager viewPager = findViewById(R.id.viewPager);
ViewPagerAdapter viewPagerAdapter = new ViewPagerAdapter(getSupportFragmentManager());
viewPagerAdapter.addFragment(new TasksFragment(), "My Tasks");
viewPager.setAdapter(viewPagerAdapter);
tabLayout.setupWithViewPager(viewPager);
...
```

12.6.2

```
public class TasksFragment extends Fragment { ...
    public void sortTasksByDate() throws Exception{
        List<Date> dates = new ArrayList<>();
        for (int i=0; i<usersTasks.size(); i++){
            Date currentTasksDate = new SimpleDateFormat("dd/MM/yyyy").parse(usersTasks.get(i).getDateDueBy());
            dates.add(currentTasksDate);
        }
        for (int i=0; i<dates.size(); i++){
            for (int j=0; j<dates.size(); j++){
                if (dates.get(j).compareTo(dates.get(i))<0 ){
                    Date tempDate = dates.get(i);
                    dates.set(i, dates.get(j)); dates.set(j, tempDate);
                    Task tempTask = usersTasks.get(i);
                    usersTasks.set(i, usersTasks.get(j)); usersTasks.set(j, tempTask); ...
                }
            }
        }

        public void fillInUsersTasks(){ ...
            public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
                for(DataSnapshot snapshot : dataSnapshot.getChildren()){
                    Task = snapshot.getValue(Task.class);
                    if (task.getBelongsTo().equals(fuser.getUid())){ usersTasks.add(task); ...
                }
            }
        }
    }
}
```

12.6.3

```
public class TasksAdapter extends RecyclerView.Adapter<TasksAdapter.ViewHolder> {
    private Context mContext;    private List<Task> mTasks;
    ...
    @Override
    public void onBindViewHolder(@NonNull final ViewHolder viewHolder, int position) {
        final Task task = mTasks.get(position);
        viewHolder.taskTitleText.setText(task.getTaskTitle()); viewHolder.dateDueBy.setText(task.getDateDueBy()) ...
        viewHolder.assessed.setText(task.getAssessed());

        viewHolder.itemView.setOnClickListener(...
            Intent intent = new Intent(mContext, ManageATaskActivity.class);
            intent.putExtra("taskTitle", viewHolder.taskTitleText.getText().toString()); ...
            intent.putExtra("difficultyLevel", viewHolder.difficultyLevel.getText().toString());...
        );
    }

    public class ViewHolder extends RecyclerView.ViewHolder{
        private TextView taskTitleText; private TextView dateDueBy; private TextView percentComplete;
        private TextView extraNotes; private TextView importanceLevel; private TextView difficultyLevel; private TextView assessed;

        public ViewHolder (View itemView){
            super(itemView);
            taskTitleText = itemView.findViewById(R.id.task_title_text); ... assessed = itemView.findViewById(R.id.assessed_text);
            ...
        }
    }
}
```

12.7

```
public class ManageATaskActivity extends AppCompatActivity { ...
    taskTitleText.setText(getIntent().getStringExtra("taskTitle")); ... difficultyLevelText.setText(getIntent().getStringExtra("difficultyLevel"));
    ...
    updatePercCompButton.setOnClickListener(new View.OnClickListener() { ...
        char[] array = percCompAnswer.getText().toString().toCharArray();
        if (array.length==2){
            if (!Character.isDigit(array[0]) || array[1]!='%'){
                proceed = false;
                Toast.makeText(ManageATaskActivity.this, "Percent Complete not in valid format !", Toast.LENGTH_SHORT).show();
            }
            ...
        } else if (array.length==3){
            ...
        }
        if (proceed){
            percCompAnswer.setText(percCompAnswer.getText().toString());
            percCompAnswer.setInputType(InputType.TYPE_NULL);
            percCompAnswer.setTextIsSelectable(false);
            ...
            public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
                for (DataSnapshot snapshot : dataSnapshot.getChildren()){
                    String taskId = "";
                    Task task = snapshot.getValue(Task.class); ...
                    taskId = snapshot.getKey();
                    reference.child(taskId).child("percentComplete").setValue(percCompAnswer.getText().toString());
                    percentCompleteText.setText(percCompAnswer.getText().toString());
                }
            }
            ...
        }
    }
}
```

12.8

```
public class NavigationDrawerActivity extends AppCompatActivity implements NavigationView.OnNavigationItemSelectedListener { ...
    DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);
    ActionBarDrawerToggle toggle = new ActionBarDrawerToggle(
        this, drawer, toolbar, R.string.navigation_drawer_open, R.string.navigation_drawer_close);
    drawer.addDrawerListener(toggle); toggle.syncState(); ...
    navigationView.setNavigationItemSelectedListener(this);
    ...
    Fragment fragment = new HomepageFragment();
    FragmentManager fragmentManager = getSupportFragmentManager();
    FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
    fragmentTransaction.replace(R.id.screen_area, fragment); fragmentTransaction.commit();
    ...
    public boolean onNavigationItemSelectedListener(Menuitem item) { ...
        int id = item.getItemId();
        if (id == R.id.nav_home){
            intent = new Intent(NavigationDrawerActivity.this, HomepageActivity.class);
            startActivity(intent); ...
        }
    }
}
```

12.9.1

```
public class SocialSpaceRelationsFragment extends Fragment { ...
    private void readUsers() { ...
        for (DataSnapshot snapshot : dataSnapshot.getChildren()){ User user = snapshot.getValue(User.class);
            if (!user.getId().equals(firebaseUser.getId()) && usersRelationsList.contains(user.getId()) ) mUsers.add(user);
        }
        ...
    }
    public void fillInUsersRelationsList(){ ...
        for (DataSnapshot ds : dataSnapshot.getChildren()){ FriendRequest request = ds.getValue(FriendRequest.class);
            if (request.getAFollowedByB().equals("true") || request.getAFollowsB().equals("true")){
                usersRelationsList.add(ds.getKey()); ...
            }
        }
    }
}
```

12.9.2

```
public class SocialSpaceAdapter extends RecyclerView.Adapter<SocialSpaceAdapter.ViewHolder> { ...
    public void onBindViewHolder(@NonNull final ViewHolder holder, int position) {
        final User user = mUsers.get(position);
        holder.username.setText(user.getUsername());
    }
}
```

```

if (user.getImageURL().equals("default")) holder.profile_image.setImageResource(R.mipmap.ic_launcher);
else Glide.with(mContext).load(user.getImageURL()).into(holder.profile_image);

holder.itemView.setOnClickListener(new View.OnClickListener() { ...
    Intent intent = new Intent(mContext, MessageActivity.class); intent.putExtra("userid", user.getId()); ...

public class ViewHolder extends RecyclerView.ViewHolder{
    public TextView username;    public ImageView profile_image;
    public ViewHolder (View itemView){
        super(itemView);
    }
}

```

12.10

```

public class MessageActivity extends AppCompatActivity { ...
    final String userid = intent.getStringExtra("userid");
    sendButton.setOnClickListener(...
        String msg = text_send.getText().toString();
        if (!msg.equals(""))    sendMessage(fuser.getId(), userid, msg);
        else    Toast.makeText(MessageActivity.this, "You can't send empty messages!", Toast.LENGTH_SHORT).show();
        text_send.setText("");
    ...

    public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
        User user = dataSnapshot.getValue(User.class);    username.setText(user.getUsername());
        if (user.getImageURL().equals("default"))    profile_image.setImageResource(R.mipmap.ic_launcher);
        else    Glide.with(MessageActivity.this).load(user.getImageURL()).into(profile_image);
        readMessage(fuser.getId(), userid, user.getImageURL());
    }
    ...

    private void sendMessage(String sender, String receiver, String message){ ...
        HashMap<String, Object> hashMap = new HashMap<>();
        hashMap.put("sender", sender); hashMap.put("receiver", receiver); hashMap.put("message", message);
        reference.child("Chats").push().setValue(hashMap); ...

    public void readMessage(final String myid, final String userid, final String imageurl){ ...
        for (DataSnapshot snapshot : dataSnapshot.getChildren()){    Chat chat = snapshot.getValue(Chat.class);
            if (chat.getReceiver().equals(myid) && chat.getSender().equals(userid) ||
                chat.getReceiver().equals(userid) && chat.getSender().equals(myid))    mChat.add(chat);
            messageAdapter = new MessageAdapter(MessageActivity.this, mChat, imageurl);
            recyclerView.setAdapter(messageAdapter); ...

```

12.11.1

```

public class FindUsers extends AppCompatActivity {...
    ViewPagerAdapter viewPagerAdapter = new ViewPagerAdapter(getSupportFragmentManager());
    viewPagerAdapter.addFragment(new UsersFragment(), "Users");
    viewPager.setAdapter(viewPagerAdapter);
    tabLayout.setupWithViewPager(viewPager);
}

class ViewPagerAdapter extends FragmentPagerAdapter {
    private ArrayList<Fragment> fragments;
    private ArrayList<String> titles;
    ...

```

12.11.2

```

public class UsersFragment extends Fragment {...
    readUsers();
    search_users.addTextChangedListener(new TextWatcher() { ...
        public void onTextChanged(CharSequence charSequence, int start, int before, int count) {
            searchUsers(charSequence.toString().toLowerCase());
        } ...
    private void searchUsers(String s){
        Query query = FirebaseDatabase.getInstance().getReference("Users").orderByChild("search").startAt(s).endAt(s+"\uf8ff");
        query.addValueEventListener(new ValueEventListener() {
            ...
            for (DataSnapshot snapshot : dataSnapshot.getChildren()){    User user = snapshot.getValue(User.class); ...

```

```

        if(!user.getId().equals(fuser.getUid())){ mUsers.add(user); ... recyclerView.setAdapter(userAdapter);
...
private void readUsers() { ...
    public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
        if (search_users.getText().toString().equals("")) { ...
            for (DataSnapshot snapshot : dataSnapshot.getChildren()){
                User user = snapshot.getValue(User.class); ...
                if (!user.getId().equals(firebaseUser.getUid())) mUsers.add(user);
                ... recyclerView.setAdapter(userAdapter); ...
            }
        }
    }
}

```

12.12.1

```

public class FriendRequestsPage extends AppCompatActivity { ...
    viewPagerAdapter.addFragment(new UsersFollowersFragment(), "My Followers");
    viewPagerAdapter.addFragment(new UsersFollowingFragment(), "I'm Following");
    viewPagerAdapter.addFragment(new RelationRequestFragment(), "My Requests");...
}
class ViewPagerAdapter extends FragmentPagerAdapter {
    ...
}

```

12.12.2

```

public class UsersFollowersFragment extends Fragment {...
    readUsers();

    private void readUsers() { ...
        public void onDataChange(@NonNull DataSnapshot dataSnapshot) { ...
            for (DataSnapshot snapshot : dataSnapshot.getChildren()){
                User user = snapshot.getValue(User.class); ...
                if (!user.getId().equals(firebaseUser.getUid()) && usersFollowersList.contains(user.getId()) ) mUsers.add(user);
            } ...
            recyclerView.setAdapter(usersFollowersAdapter);
        }
    }

    public void fillInUsersFollowersList(){ ...
        public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
            for (DataSnapshot ds : dataSnapshot.getChildren()){ FriendRequest request = ds.getValue(FriendRequest.class);
                if (request.getAFollowedByB().equals("true")) usersFollowersList.add(ds.getKey());
            }
        }
    }
}

```

12.12.3

```

public class UsersFollowersAdapter extends RecyclerView.Adapter<UsersFollowersAdapter.ViewHolder> { ...
    public void onBindViewHolder(@NonNull final ViewHolder holder, int position) { ...
        holder.deleteFriendButton.setOnClickListener(...
            AlertDialog.Builder builder = new AlertDialog.Builder(mContext);
            builder.setMessage("Do you want to remove " + holder.username.getText() + " from your list of followers?").setTitle("Delete user?") ...
            .setPositiveButton("Delete", new DialogInterface.OnClickListener() { ...
                reff.child(firebaseUser.getUid()).child(holder.id).child("AFollowedByB").setValue("false");
                refff.child(holder.id).child(firebaseUser.getUid()).child("AFollowsB").setValue("false"); ...
            }.setNegativeButton("Cancel", new DialogInterface.OnClickListener() {...
            AlertDialog alertDialog = builder.create(); alertDialog.show();
        }
    }

    holder.actionButton.setOnClickListener(new View.OnClickListener() { ...
        Intent intent = new Intent(mContext, TaskAssignActivity.class);
        intent.putExtra("followersId", user.getId()); intent.putExtra("followersUsername", holder.username.getText());
        intent.putExtra("usersId", firebaseUser.getUid()); ...
    }
}

```

12.12.4

```

public class TaskAssignActivity extends AppCompatActivity {...
    tasksFor.setText("Task is for " + intent.getStringExtra("followersUsername")); ...
    dateDueByAns.setOnClickListener( ...
        Calendar calendar = Calendar.getInstance();
        int year = calendar.get(Calendar.YEAR); int month = calendar.get(Calendar.MONTH); int day = calendar.get(Calendar.DAY_OF_MONTH);
        DatePickerDialog dialog = new DatePickerDialog(

```



```

        TaskAssignActivity.this, android.R.style.Holo_Light_ButtonBar_AlertDialog, dateSetListener, year, month, day);
    ...
    dateSetListener = new DatePickerDialog.OnDateSetListener() {
        @Override
        public void onDateSet(DatePicker datePicker, int year, int month, int dayOfMonth) {
            month = month + 1;      String date = dayOfMonth + "/" + month + "/" + year;
            dateDueByAns.setText(date);
        }
    };
    ...
    assignTaskButton.setOnClickListener(new View.OnClickListener() { ...
        boolean proceed = true;
        char[] array = percentCompleteAns.getText().toString().toCharArray();    String extNot = extraNotesAns.getText().toString();
        int radio1 = RGImportLev.getCheckedRadioButtonId();
        RGImportLevAns = findViewById(radio1); ...

        if (taskTitleAns.getText().length() > 30){
            Toast.makeText(TaskAssignActivity.this, "Task Title is too long. Please shorten it !", Toast.LENGTH_SHORT).show();
            proceed = false;
        }
    });
    ...
    if (dateDueByAns.getText().toString().equals("Tap to choose a date")){
        Toast.makeText(TaskAssignActivity.this, "Choose a valid date !", Toast.LENGTH_SHORT).show();    proceed = false;    }
    if (array.length==2){
        if (!Character.isDigit(array[0]) || array[1]!='%'){
            Toast.makeText(TaskAssignActivity.this, "Percent Complete not in valid format !", Toast.LENGTH_SHORT).show();
            proceed = false;
        }
    }
    ...
    if (extNot.length() > 100){
        Toast.makeText(TaskAssignActivity.this, "Extra notes is too long !", Toast.LENGTH_SHORT).show();
        proceed = false;
    }
    if (proceed){
        HashMap<String, String> hashMap = new HashMap<>();
        hashMap.put("assessed", RGImportLevAns.getText().toString()); ...
        hashMap.put("taskTitle", taskTitleAns.getText().toString());
        reference.push().setValue(hashMap); ...
    }
}

```

12.12.5

```

public class UsersFollowingFragment extends Fragment { ...
    readUsers(); ...
    private void readUsers() {
        reference.addValueEventListener(...
            public void onDataChange(@NonNull DataSnapshot dataSnapshot) { ...
                if (!user.getId().equals(firebaseUser.getId()) && usersFollowingList.contains(user.getId())) mUsers.add(user);
                friendsAdapter = new UsersFollowingAdapter(getContext(), mUsers);    recyclerView.setAdapter(friendsAdapter);    ...
            }
        }
    }
    public void fillInUsersFollowingList(){...
        friendsRef.addValueEventListener(new ValueEventListener() { ...
            for (DataSnapshot ds : dataSnapshot.getChildren()){
                FriendRequest request = ds.getValue(FriendRequest.class);
                if (request.getAFollowsB().equals("true"))    usersFollowingList.add(ds.getKey());    ...
            }
        });
    }
}

```

12.12.6

```

public class UsersFollowingAdapter extends RecyclerView.Adapter<UsersFollowingAdapter.ViewHolder> { ...
    public UsersFollowingAdapter( ... ) { ... }
    @Override
    public void onBindViewHolder(@NonNull final ViewHolder holder, int position) {
        final User user = mUsers.get(position);
        holder.actionButton.setText("My assigned Tasks");    ...
        holder.deleteFriendButton.setOnClickListener(new View.OnClickListener() { ... });

        holder.actionButton.setOnClickListener(new View.OnClickListener() { ...
            Toast.makeText(mContext, "Loading data...", Toast.LENGTH_LONG).show();
            Intent intent = new Intent(mContext, TasksAssignedActivity.class);
            intent.putExtra("assignersUsername", holder.username.getText().toString());    intent.putExtra("taskSettersId", holder.id);    ...
        });
    }
}

```

12.12.7

```
public class TasksAssignedActivity extends AppCompatActivity {...
    Bundle bundle = new Bundle();
    bundle.putString("taskSettersId", taskSettersId);
    TasksAssignedFragment fragment = new TasksAssignedFragment();
    fragment.setArguments(bundle);
    viewPagerAdapter.addFragment(fragment, title);
    viewPager.setAdapter(viewPagerAdapter);
    tabLayout.setupWithViewPager(viewPager);
}
```

12.12.8

```
public class TasksAssignedFragment extends Fragment {...
    fillInUsersAssignedTasks(); ...
    sortTasksByDate();

    new Handler().postDelayed(new Runnable() {...
        taskAdapter = new TasksAdapter(getContext(), usersTasks);    recyclerView.setAdapter(taskAdapter);
    }, 3000); ...

    public void sortTasksByDate() throws Exception{    ...    }

    public static void fillInUsersAssignedTasks(){...
        reference.addListenerForSingleValueEvent(new ValueEventListener() {...
            for(DataSnapshot snapshot : dataSnapshot.getChildren()){
                Task task = snapshot.getValue(Task.class);
                if (task.getBelongsTo().equals(fuser.getUid()) && task.getSetBy().equals(taskSettersId)) {
                    usersTasks.add(task);    ...
                }
            }
        })
    }
}
```

12.12.9

```
public class RelationRequestFragment extends Fragment { ...
    getUsers(); ...
    private void getUsers() {...
        DatabaseReference reference = FirebaseDatabase.getInstance().getReference("Friends").child(firebaseUser.getUid());    ...
        public void onDataChange(@NonNull DataSnapshot dataSnapshot) {...
            for (DataSnapshot snapshot : dataSnapshot.getChildren()){
                FriendRequest friendRequest = snapshot.getValue(FriendRequest.class);
                final String requestersID = snapshot.getKey();
                if (friendRequest.getAFollowsB().equals("pending")){
                    for (int i = 0; i<allUsers.size(); i++)    if (allUsers.get(i).getId().equals(requestersID))    mUsers.add(allUsers.get(i));    ...
                }
            }
        }
    }
}
```

12.12.10

```
public class RelationRequestAdapter extends RecyclerView.Adapter<RelationRequestAdapter.ViewHolder>{ ...
    public RelationRequestAdapter(...) {... }    ...
    @Override
    public void onBindViewHolder(@NonNull final ViewHolder holder, int position) {...
        holder.itemView.setOnClickListener(new View.OnClickListener() {...
            AlertDialog.Builder builder = new AlertDialog.Builder(mContext);
            builder.setMessage("This user wants you to follow them. They will be able to assign you tasks to complete. Accept request?").setTitle("Friend Request").setCancelable(true)
                .setPositiveButton("Accept", new DialogInterface.OnClickListener() {...
                    Toast.makeText(mContext, "Accepted !", Toast.LENGTH_SHORT).show();...
                    reference.child("AFollowsB").setValue("true");...
                    requestersReference.child("AFollowedByB").setValue("true");
                })
            ...
            .setNegativeButton("Reject", new DialogInterface.OnClickListener() {...
                Toast.makeText(mContext, "Rejected !", Toast.LENGTH_SHORT).show();...
                reference.child("AFollowsB").setValue("false");...
                requestersReference.child("AFollowedByB").setValue("false");
            })
        }
    }
}
```

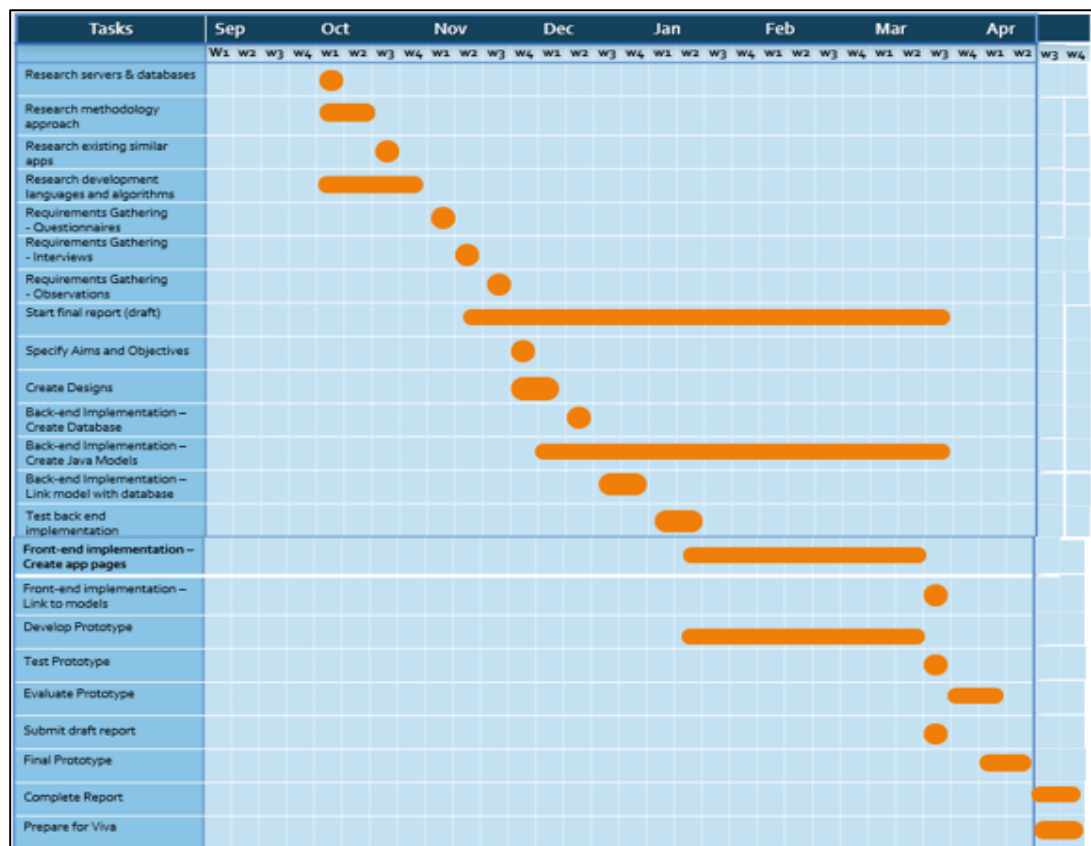
12.13.1

```
public class SuggestedTasksActivity extends AppCompatActivity { ...
    ViewPagerAdapter viewPagerAdapter = new ViewPagerAdapter(getSupportFragmentManager());
    viewPagerAdapter.addFragment(new SuggestedTasksFragment(), "Suggested Tasks");
    viewPager.setAdapter (viewPagerAdapter);
    tabLayout.setupWithViewPager(viewPager);
}
```

12.13.2

```
private static HashMap sortByValues(HashMap map) {
    List list = new LinkedList(map.entrySet());
    Collections.sort(list, new Comparator() {
        public int compare(Object o1, Object o2) {
            return ((Comparable) ((Map.Entry) (o1)).getValue())
                .compareTo(((Map.Entry) (o2)).getValue()); ...
        }
    });
    HashMap sortedHashMap = new LinkedHashMap();
    for (Iterator it = list.iterator(); it.hasNext(); ) {
        Map.Entry entry = (Map.Entry) it.next();
        sortedHashMap.put(entry.getKey(), entry.getValue());
    }
    for (Object key : sortedHashMap.keySet()){
        Task t = (Task)key;
    }
    return sortedHashMap; ...
}
```

12.14 Overall Project Gantt Chart



12.15 Decision Tree Algorithm

