# Designing a task management system that utilizes context

Lars Erik Strand

Master's Thesis
Master of Applied Computer Science
30 ECTS
Department of Computer Science and Media Technology
Gjøvik University College,

Avdeling for
informatikk og medieteknikk
Høgskolen i Gjøvik
Postboks 191
2802 Gjøvik

Department of Computer Science
and Media Technology
Gjøvik University College
Box 191
N-2802 Gjøvik
Norway

# Designing a task management system that utilizes context

Lars Erik Strand

2014/10/21

# Abstract

Task management systems such as Calendars and to-do lists are important tools for many people that helps them to stay organized. These systems are usually quite simple and does not go beyond simply keeping track of the user's events and tasks. This thesis looks at how the usefulness of such systems can be even further improved, by introducing the use of user contexts.

In order to make a task management system utilize contexts, there are many considerations and decisions that has to be made. This thesis study these considerations. We also design a proof of concept to-do list application for mobile devices that is able to collect and use context information. We experiment with the use of a recommender to recommend tasks and an ordering of tasks for a user, based on contexts and previous tasks.

# Preface

I would like to thank my two supervisors Mariusz Nowostawski and Rune Hjeldsvold for the help I received during the writing of this thesis. Their help has been invaluable. A special thanks to Mariusz Nowostawski for overseeing the entire development process and being easily available for questions regarding mobile development.

# Contents

**Abstract** . . . . . . . . . . i
**Preface** . . . . . . . . . . ii
**Contents** . . . . . . . . . . iii
**List of Figures** . . . . . . . . . . v
**List of Tables** . . . . . . . . . . vi
**1 Introduction** . . . . . . . . . . **1**
1.1 Topic . . . . . . . . . . 1
1.2 Problem description . . . . . . . . . . 1
1.3 Justification, motivation and benefits . . . . . . . . . . 2
1.4 Research questions . . . . . . . . . . 2
1.5 Thesis structure . . . . . . . . . . 2
**2 Related Work** . . . . . . . . . . **4**
2.1 Context awareness . . . . . . . . . . 4
2.2 Collecting contexts . . . . . . . . . . 4
2.3 Context history . . . . . . . . . . 4
2.4 Making predictions based on context . . . . . . . . . . 5
2.5 Context abstraction and modeling . . . . . . . . . . 5
2.6 Intelligent task and time management systems . . . . . . . . . . 5
2.6.1 Alternative recommendation algorithms . . . . . . . . . . 6
2.6.2 Probability calculations . . . . . . . . . . 7
**3 Methodology** . . . . . . . . . . **8**
3.1 User group . . . . . . . . . . 8
3.2 Application . . . . . . . . . . 9
3.2.1 General application design . . . . . . . . . . 9
3.2.2 Collecting context information . . . . . . . . . . 13
3.3 Context storage and modeling . . . . . . . . . . 14
3.4 Recommender . . . . . . . . . . 16
3.4.1 Chosen recommendation algorithm . . . . . . . . . . 17
3.5 Application distribution and user experiments . . . . . . . . . . 18
**4 Results** . . . . . . . . . . **19**
4.1 Application design . . . . . . . . . . 19
4.1.1 Student tasks . . . . . . . . . . 19
4.1.2 Collected student tasks . . . . . . . . . . 19
4.1.3 Design decisions . . . . . . . . . . 23
4.2 Context collection . . . . . . . . . . 24
**5 Discussion** . . . . . . . . . . **26**

5.1 Application design decisions . . . 26
5.1.1 Task collection issues and improvements . . . 27
5.2 Context collection . . . 27
5.3 Recommender and context usage . . . 28
5.4 Generalization of tasks . . . 29
**6 Conclusion . . . 30**
6.1 Research question 1 . . . 30
6.2 Research question 2 . . . 30
6.3 Research question 3 . . . 30
**7 Future work . . . 31**
**Bibliography . . . 32**
**A Student tasks questionnaire . . . 34**
**B Application GUI . . . 36**
**C Recommender algorithm . . . 37**

# List of Figures


1 Conceptual model . . . . . . . . . . 10
2 Main application screen . . . . . . . . . . 11
3 Application workflow . . . . . . . . . . 12
4 Screen for creating a new task . . . . . . . . . . 13
5 In-app representation of task . . . . . . . . . . 14
6 Context collection notification . . . . . . . . . . 14
7 Database model . . . . . . . . . . 15
8 External database model . . . . . . . . . . 16
9 To-do list usage . . . . . . . . . . 20
10 Student task regularity . . . . . . . . . . 20
11 Student task frequency . . . . . . . . . . 21
12 Collected tasks sample . . . . . . . . . . 22
13 Collected task type distribution . . . . . . . . . . 22
14 All tasks and tasks with collected contexts comparison . . . . . . . . . . 24
15 Collected contexts sample . . . . . . . . . . 24
16 Student tasks questionnaire part 1. . . . . . . . . . . 34
17 Student tasks questionnaire part 2. . . . . . . . . . . 35
18 Main application screen design showing tasks for “today”. . . . . . . . . . . 36
19 Main application screen design showing tasks for the “week”. . . . . . . . . . . 36

# List of Tables

1 Example recommendation probability hashmap . . . . . . . . . . . . . . . . . . . . 17
2 Location recommendation probability hashmap . . . . . . . . . . . . . . . . . . . 18
3 Resulting probability hashmap upon which the actual recommendation is made . 18
4 Other student tasks response . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
5 Time spent on tasks . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23

# 1 Introduction

## 1.1 Topic

Task and time management applications help users stay organized by keeping track of notes, meetings, tasks etc. Many applications exist that are directed towards this purpose, ranging from simple note-taking applications and to-do lists to more advanced ones like calendars and scheduling applications. A calendar typically holds events or appointments for a user while a to-do list will keep more detailed and lower level tasks. All of these applications have in common that they relieve the user of having to remember things, thus allowing the user to focus more deeply on other things.

Context awareness is also a field of research that has received more attention in recent years. The reason for this is the increasing number of mobile devices that are available and also the increasing functionality of these devices. When it comes to context awareness, the sensors in mobile devices play a huge role. There are more sensors packed into these devices now than ever before, which allows applications to collect very specific types of contextual information. This in turn allows for the development of applications that have very specific and tailored purposes.

By combining context awareness and task management, we open up for new types of applications. Smarter systems could be built that leverage contextual information, both past and present, to adapt its behavior to accommodate very specific situations. A calendar system for example, could evaluate a user's upcoming meeting and the current location of the user. Taking into account the distance between the user and meeting location, the system could then deliver a reminder at the appropriate time, allowing the user to catch the meeting. Another example would be a to-do list application that could leverage contextual information about previously performed tasks to provide task recommendations to the user.

## 1.2 Problem description

Task and time management applications are valuable tools that are used by many people. These applications can be especially valuable on mobile devices as the users can carry these devices with them, thus having the application data easily accessible. Concrete and popular examples of such applications are calendars such as Google Calendar[1] or to-do list applications such as Trello[2] or Todoist[3]. Even though these, and many similar applications are very useful, their functionality and usefulness could be further improved by integrating user context.

Utilization of contextual information in different scenarios have been widely researched. However, we have found no research that studies the usage of such information in the specific domain of task and time management systems. Many systems are improved by making them context-aware. One example of this is Learning Management Systems (LMS) that can suggest learning content based on the users current context[4], and in doing so increasing the level of learning for the user. Following examples such as this, it is believed that task management applications can also reap the benefits of context awareness.

This study will look into the area of utilizing contextual information in task management applications. We will design a proof of concept application that tries to make use of the user's contexts.

## 1.3 Justification, motivation and benefits

The complete envisioned application is a personal information manager (PIM) where the user only needs to inform the application of the tasks that he/she needs to perform. The application would then suggest a current task and provide an optimal ordering of the other tasks, taking into account the current user context. This optimization can for example be based on minimizing total time spent completing the tasks, minimizing total travel distance between tasks, thereby reducing travel costs and pollution, or a combination of these. With respect to time, this would mean that completing the tasks in any other order than that suggested by the system would lead to a longer time spenditure. In order to create such a system, many areas would need extensive research, more than that which is possible to complete within the scope of this thesis. However, sub-parts of the system can be identified and researched, providing building blocks for the realization of the entire system. These sub-parts include how to collect relevant context data, how to store those data, how to implement an overall design of such an application and how to evaluate past tasks and contexts to successfully produce task suggestions. Identifying and researching these parts of the system justifies the work to be conducted in this thesis, as it allows for working towards the higher-end goal. The motivation behind the thesis is to be able to contribute to to-do applications, and by extension task management applications in general, by further enhancing their task management capabilities.

## 1.4 Research questions

This thesis is three folded. Firstly, there is the area of context collection and representation. Secondly, we have the software engineering area, which relates to application design and functionality. The third area revolves around the context in which the previous two are placed for this thesis, namely students and the educational context for which the thesis is conducted in. The proposed solution will have to address all of these areas in detail. Therefore, these research questions have been formed for each of the areas:

- **Research question 1**: What contexts are relevant in a task management application, and how can they be acquired and modeled?
- **Research question 2**: How can a task management application be designed and what are the important design decisions when making such an application?
- **Research question 3**: How can context information be utilized in a task management application?

## 1.5 Thesis structure

Chapter 2 provides the necessary background that related to the work in this thesis. State-of-the-art task management applications are also covered here. Chapter 3 describes the whole process of designing the to-do list application and is also in its entirety the answer to research question 2 and 3. Chapter 4 presents the findings in this thesis and is together with Chapter 3 and 5 the answer to research question 1. Chapter 5 discusses the different approaches and decisions made in this thesis, as well as discussing alternatives for some of these decisions. The work is then concluded in Chapter 6 and futer work is discussed in Chapter 7.

# 2 Related Work

## 2.1 Context awareness

Context awareness is a large area of research. This has led to many proposed definitions of context and context awareness [5]. Context can refer to many different things, in fact, everything that happens in everyday life, happens in a context. Because of this, it is important to provide a specific definition of context when researching the area in order to avoid confusion. This thesis will use a widely used and commonly accepted definition of context proposed by Dey [6], stating that:

> “Context is any information that can be used to characterize the situ- ation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves.”

Following this, a context-aware application is therefore an application that gathers contexts, interprets them, and adapts its behavior accordingly during runtime to adjust to the users situations and needs.

## 2.2 Collecting contexts

In a mobile device there are typically many types of contexts that can be collected via the system and its sensors. Some of these are:

- Location
- Movement
- Time
- Activity
- Air pressure
- Ambient sound
- Humidity
- Orientation of device

## 2.3 Context history

Context-history refers to persistent storing of contextual information, in order to use this information for future purposes. The limitation of current to-do list applications is that they do not propose the usage of context-based history, and only considers current context. Utlilizing context-history usually involves reqognizing patterns in the stored context information. Studies involving context histories is also limited in the sence that most of them focus on building histories rather than utilizing them [7].

A study looking into the utilization of context-histories has been presented in [8]. This study tries to predict the preferences of the user by utilizing context-history. They implement a context management layer into their system where an inference agent infers the high level contexts from the raw, sensed contexts. These high level contexts are then stored as context-history represented by the OWL ontology language.

One way to extract usefull information through context histories would be deducting user habits from them. Ciaramella did such a study [9]. The study proposed a resource recommender that adapts to the habits of a specific user. This adaption was based on genetic algorithms (GA's). By tracking user behavior on a mobile device, they collected the context-history and utilized fuzzy linguistic variables to handle vagueness in the collected data. The study showed that adding context-histories and GA's to the calculation of recommended resources, improved both responsiveness and modeling capabilities of the recommender.

## 2.4 Making predictions based on context

Mayrhofer et al. discuss some issues regarding context-prediction [10]. When trying to predict a user context in order to proac1tively perform a task for the user, it is important that the predictions are accurate. This is generally a significant problem in the area of context prediction. This thesis is not focused on proactively performing tasks for the users. It is focused on providing task suggestions based on these predictions, thereby making the problem slightly less significant. However, the user experience will be related to the accuracy of these suggestions, meaning that it will still be of some importance.

When making predictions based on contexts, how the prediction is performed is equally important as that which is predicted. A study on how to proactively determine spatial data about the user is done in [11]. Here, a Predictive Context Object (PCO), modelled in UML, is proposed for modelling predictive data. A prediction algorithm is also proposed, consisting of concrete mathematical formulae.

## 2.5 Context abstraction and modeling

Many studies have looking into context modeling and representation *(reference(s) here)*. Context modeling and representation studies is often about providing proper abstraction from the raw context data. However, by using the integrated features for collecting contexts in Android, such abstractions are already built into the framework. Instead of getting raw contextual data from the framework, we can get data that are ready to be used and stored directly. For location contexts for example, we could get GPS coordinates, or even a specific address. For a user activity context we could get the data as a string indicating whether the user is *still*, *on foot* or *in vehicle* to name a few.

## 2.6 Intelligent task and time management systems

The SELF-PLANNER [12] proposed an intelligent Web-based calendar system where a users schedule can be created automatically by the system [13]. Research in intelligently scheduling a user's activities also touches the field of Artificial Intelligence (AI). The study investigates different types of user activities, such as activities that are dependent upon each other, interuptable activities, location-dependent activities, and activities of variable length. It also studies how these activities can relate to each other in order to create a model of the activities to be used in the scheduling algorithm. The work was based on the Squeaky Wheel Optimization framework, and shows that letting an intelligent system schedule user activities can generate effective and qualitative plans [13].

Norton et al. proposed a information management system that can autonomously rearrange a task list depending on the current user context [14]. The system is similiar to a to-do list application. The paper studies relationships between tasks based on their priorities, deadlines, and locations in order to autonomously rearrange tasks with respect to time optimization. However, the system does not suggest using context history for this purpose. The paper also focus more on a smart way to recieve location data in order to save battery power. Trending (history) is mentioned, but this is purely for location and to further optimize the battery life. For ordering the tasks the system normalizes the relevant factors and use a multi-dimensional Euclidean space to determine the ordering.

Research presenting automated task assistants have also been proposed. Towel [15] propose a intelligent to-do list system, allowing for simple user task to be automatically performed by a system agent. The users to-dos are integrated with the execution-agent, called Project Execution Assistant (PExA) [16]. The tasks that the agent can perform need to be rather simple tasks that the agen can interpret and fully comprehend in order to do the tasks correctly. Such tasks include sending emails, looking for available hotels and arranging meetings. By relieving the user of these trivial tasks, the user can then focus on more important task requiring human problem solving skills.

A similar study was presented in [17]. This study proposed a system architecture for task and time management systems. This system's purpose is also to relieve a user from routine tasks, so that the user can focus on more complex tasks, thus increasing productivity among knowledge workers. The utilization of PExA are more thoroughly described in this study. A possible shortcoming in the study is that the agent performing the user tasks, need to be given specific instructions on a high level of detail by the user. This means that a method for infering data about a user task is not proposed.

Driver and Clarke proposed something that they called context-aware trails [18], which is a list of scheduled activities. Activity scheduling based on context can help users in many different areas, such as a hospital worker needing to do patient rounds and administrative tasks. Generating these trails has some problem areas that are much like the Traveling Salesman Problem (TSP) [19]. The mechanism for generating the trails is based on context-based activity set reduction, where the activity set is the entire list of activities the user could do while using the application.

The reviewed proposals are relevant for task and time management systems. However, none of these systems have looked into the use of context-histories for their purposes.

### 2.6.1 Alternative recommendation algorithms

Neural networks describes the process in which the computer. . . (half a page with a couple of references. . . )

### 2.6.2 Probability calculations

The second approach is to use probability calculations to perform the recommendations. This is the approach that was decided to be used in this project. *(some general information + a couple of references. . . )*

# 3 Methodology

In order to address the research questions, we wanted to develop a proof of concept system that is able to investigate and utilize the concepts in question. It was decided that a to-do list application would be a good candidate for investigating these concepts. A to-do list application is a task management system in a simple form. The simplest version of it could be just a list of strings representing the user's tasks, or to-do items, that needs to be done. Another form of a task management system is a calendar. A calendar would also be a suitable candidate for looking into the use of contexts in task management. However, a calendar is more complex than a to-do list, even in its simplest form, and would require more design decisions than a to-do list if building a system from scratch. Considering this, a to-do list application was chosen.

By designing a to-do list application, we want to discover the important decisions that needs to be made when developing such an application, both in terms of architecture as well as graphical design. We also want to discover how to make such an application context-aware, and how contexts can be used towards a meaningful purpose in such a system. This involves how to collect the contexts, how to store them, and also how to represent them so that they can be used.

When building context-aware systems, the most natural choice of platform is mobile devices. Contextual information is readily available on such devices because of their cheap and integrated sensors, which now exists on nearly all modern handheld devices. It is easy to access information such as user activity, time and location. The two most popular operating systems for such devices are Android and iOS. The development frameworks for both of these have functionality that allows developers to easily collect contextual data in their applications. Even context abstraction and representation are handled by the frameworks, so that the developer no longer has to deal with the raw sensory data coming from the hardware. For this particular thesis, the Android Operating System was chosen for development. This is mostly because it holds the biggest market share by far, but also because an Android device was easily available for development.

## 3.1 User group

In our application we decided that using a recommender to recommend tasks from the users to-do list based on contexts was one way of utilizing contexts in such an application. For the recommender to be able to differ between user tasks, these tasks needs to be assigned different types and be categorized. However, categorizing typical and everyday tasks will require a lot of effort. It is known that modeling an individual's tasks and activities can be challenging [13]. Consider trying to generalize the tasks of an office worker and a carpenter. The tasks that these two typically do during a day, will vary greatly. Finding similarities among these tasks so that they could be generalized to fit a general population would be difficult. Therefore, in order to reduce complexity and the amount of work needed, it was decided that the scope should be narrowed down by selecting a specific user group. This will also allow for narrowing down the application itself, both in terms of general application design as well as the complexity of context collection and representation.

The user group that was selected was students, and more specifically students in an educational environment. There were two main reasons for this. One being that the tasks of such a narrowed down and specific user group can be much easier divided into separate types and categorized. We will then be able to predefine a set of tasks that should cover most of the different types of tasks that a student will be doing in an educational environment. The second reason for choosing students was that this user group was easily accessible, both for handing out questionnaires and also for seeking participants for an experiment.

Although students were selected as the target user group, typical educational tasks can still vary to a large degree between undergraduate and postgraduate students. This can also be highly dependent on what semester the student in question is currently in. A bachelor student might have very different tasks than a master student doing his/her master thesis. To determine these differences, and to find out whether or not such differences needed to be accounted for in the application design, a short questionnaire was made. Detailed results of the questionnaire can be found in Section 4.1.1. Through the questionnaire, we found that there were only minor differences between the tasks of the different types of students. It was therefore decided that the tasks should be categorized equally for all students.

The actual task categories used for the tasks in the application were:

- Attend lecture
- Read
- Write report
- Do course exercises
- Meeting
- Group work
- Practical work
- Other

These categories are based on the results of the short student task questionnaire we made. We provided a set of predefined tasks where the student had to answer how often they typically performed each task, both in terms of regularity (once a day, once a week etc.) and frequency (1-3 times a week, 4-6 times a week etc.). The reason we distinguished between frequency and regularity is that we wanted to catch if a user did a typical task irregularly (once a month), but then quite often during that period (multiple times per day, once a month, for example). The students were also able to fill in additional tasks, if the predefined tasks were not enough to cover all the tasks that they typically do.

## 3.2 Application

### 3.2.1 General application design

By choosing to create a todo-list application, the user should be able to perform actions that are normal for such applications. These actions include creating and storing tasks, as well editing

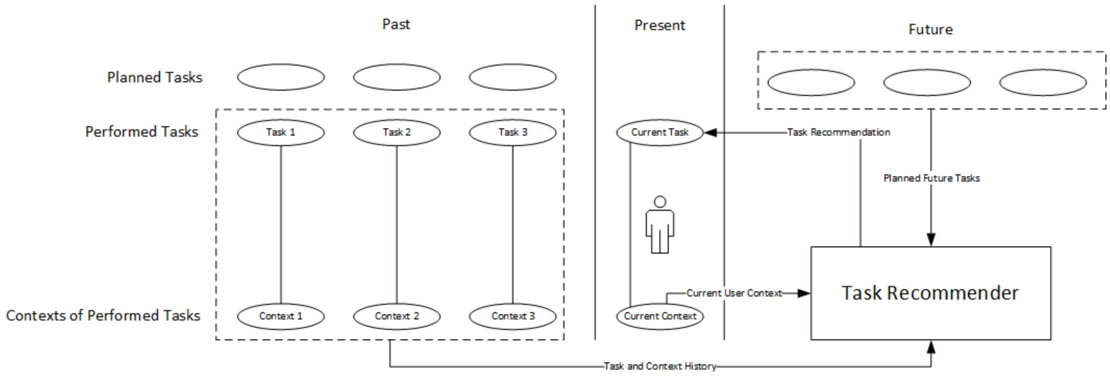



Figure 1: Conceptual model of the application.

and deleting them. It should also be possible to organize the tasks by arranging them into lists. By studying other to-do list applications [2, 3] we came up with a set of minimum requirements that the application would have to fulfill.

- Creating tasks.
- Editing/deleting tasks.
- Creating lists for tasks.
- Managing lists by editing/deleting them, or moving tasks between lists.

These are core features that are supported in popular to-do lists, and the users would expect a to-do list to have this functionality. The application in this study is developed in a way that supports all these aspects. When a todo-item or task is created, a date will be attached to that particular task, thereby organizing the tasks with similar dates into lists.

We want to utilize context features in our application in order to make task recommendations for the user. Therefore, we would have a few more requirements in addition to the ones mentioned above:

- Collection of contexts
- Modeling and representing the contexts in a way that they can be utilized.
- A recommender module to process the tasks and contextual information.

With the above requirements in mind, we designed a conceptual model of the application, shown in Figure 1. The figure shows the general idea behind the application. We have the user in the center with a current context. On the top right we have all the tasks that the user has planned to do in the application. When a user selects one of these tasks and starts doing the task, the mobile device will continuously collect information about the users contexts. Contexts are collected for as long as the user is actively performing the task. When the user tells the application that he/she is finished with a task, all the collected contexts are stored together with the task in the database, as shown on the left side of the figure (dotted lines). Here, the completed tasks and their related contexts make up the task and context history.

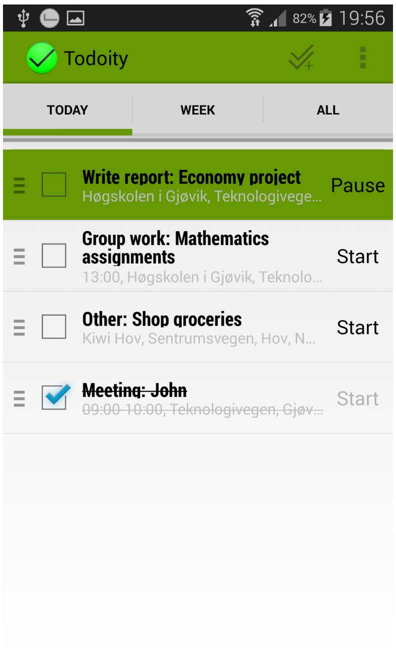



Figure 2: The main screen of the application, showing a few tasks where one is ongoing and one is finished.

The recommender is a separate and independent module in the application. The lines pointing towards the recommender represents its input, whereas the line pointing outward is the output, i.e. the actual task recommendation. The input for the recommender consists of three parts:

- The tasks that the user needs to do (planned tasks).
- The tasks that the user has previously done (task history) and the contexts related to these tasks (context history).
- The users current context.

By taking all these components into account, the recommender will be able process the information and suggest a task and a task ordering for the user. The recommender part of the application is discussed more thoroughly in Section 3.4.

In the overall design of the application, we decided to stay with the Android design principles[20] as much as possible. Popular apps on Google Play tend to follow similar design patterns, which means that Android OS users are accustomed to a certain look and feel. We wanted to provide a look and feel in our application that would be familiar to the users, in this case the students using the application. The graphical design of the main application screen is shown in Figure 2. Here we can see several similarities with the design guidelines. There is a toolbar with common actions at the top (new task button and an overflow menu with settings and about) which is also given a app-specific color that runs through several other parts of the application (also called branding). We also provided tabs to allow the users to easily change between viewing both fu-

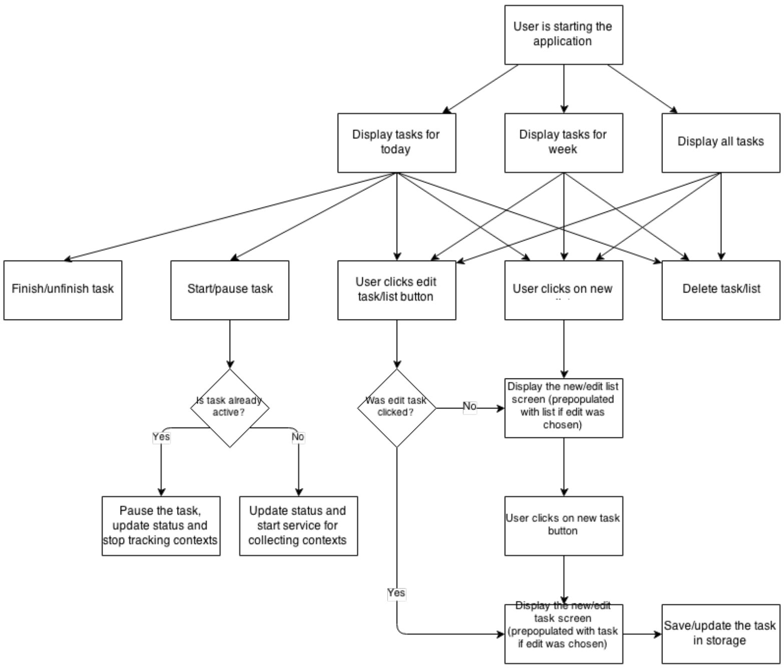



Figure 3: Application workflow.

ture planned tasks and past tasks. The *today* tab is the default tab, displaying the tasks that the user has planned for the current day. Tasks that have been started by the user is placed at the top of the list and given a background color to indicate that it is running. Finished tasks are automatically placed at the bottom. The user also have the option to reorder the tasks through drag and drop.

A slightly simplified version of the overall workflow of the application is shown in Figure 3. This depicts the basic functionality that is provided.

When the users launches the application, the *today* screen is shown by default. From here, the user can create a new list of tasks, add tasks to current lists or delete tasks or lists. Upon selecting a list, the user can edit and delete existing tasks within that list or create new tasks. When wanting to edit or create a new task the user is directed to the new task screen. This is where the actual information about the task is entered. The creation of a new task is shown in Figure 4. A category for the task has to be entered here. This is done by selecting one of our predefined tasks from a list. The user can also provide an optional description for the task. For this thesis, it was decided that a location should also be required by the users to provide. This is because we are designing a context-aware application that heavily relies on location. However, for a commercialized version of the application, a location would probably not be required as the users might get tired of having to input a location for every task. The location can either be input as text, where an autocomplete function was implemented to help users select real locations, or the map button can be clicked, which lets the user pinpoint the location from a map. The tasks can also have a fixed starting time and/or end time, which is optional.

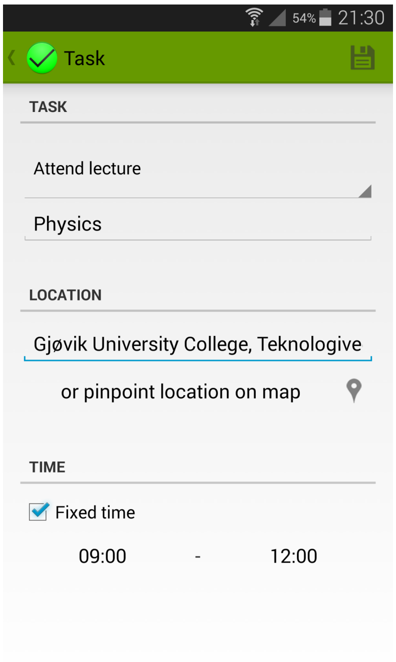



Figure 4: Creating a new task.

### 3.2.2 Collecting context information

By storing contextual information about how tasks are performed, the recommender will be able to not only provide recommendations based on current and planned contexts, but also take into account in what contexts tasks have been done previously. A certain task may work well in one context and poorly in another. For example, if a user always perform one specific task in one specific context, such as always doing *attend lecture* at campus, it would be meaningless to recommend this task if the user is at home. Before designing the overall schema of context representation, decisions on what contexts to actually use and collect in the application needs to be made.

Although a whole variety of different contexts may be collected by a mobile device, not all of them would be equally relevant to use in a to-do list application. While most contexts could be used to some degree, there are situations were some contexts would be redundant. Humidity, for example, could be a somewhat useful context for someone working on very specific tasks in very specific environments, say an environmentalist taking water samples near lakes and rivers. However, for students in an educational environment, such contexts are less relevant. Because of this, the actual contexts that was decided to be collected in this application were:

- *Location:* Both the planned location of the task as well as the actual location of where the task is performed is stored.
- *Activity:* The movement of the device while contexts are collected.

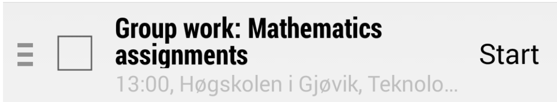



Figure 5: The in-app representation of a planned task.


Figure 6: On the left, the notification displayed when the app is collecting contexts in the background.

- *Time:* Each task is given timestamps both when they are started and ended. By doing this it is possible for the recommender to separate tasks that are done at specific times of the day, week or even month. Time spent doing a task is also tracked, as this may differ from the difference between the start and end times (users may pause doing a task).

These are the ones we considered to be the most relevant for students in an educational environment. While other contexts, such as ambient sound, may be quite relevant for some tasks, such as reading, we decided not to implement the use of these contexts in our application. It would be interesting from a researching point of view to look at as many different contexts as possible, but given the timeframe of this thesis and the amount of work needed to implement them, we reduced the number of used contexts to the ones mentioned above.

The actual logic behind the tracking of tasks and their contexts is a separate problem. It was decided that the tracking of tasks would have to be manually started and stopped by the users. This was done by providing a start button for each task, as shown in Figure 5. The process of collecting contexts when the user is performing a task would ideally be done autonomously by the application, without letting the user be responsible for starting and stopping the tasks. However, there are difficulties in making this happen, which is discussed in Section 5.2. When the user starts a task, the application indicates to the user that something is happening in the background. This is done by highlighting the tasks for which contexts are being collected (see Figure 2), as well as displaying a simple notification to indicate that the application is working in the background (see Figure 6).

## 3.3 Context storage and modeling

A database was needed to store both the collected contexts and the tasks. Both an internal and external database was chosen to hold the data. An internal database was chosen because it is easy to implement as well as it allows for the application to be used while being offline. However, in order to easily retrieve and analyze the collected tasks and contexts for the thesis results, an external database was chosen. While the internal database only holds information regarding the specific installation, the external database holds all the information about all installations, i.e. all tasks and all contexts from all users. We used the Android built-in SQLite database for the internal storage and Google's AppEngine datastore for the remote storage. We chose AppEngine to host our remote data, because it provides an API that allows for easy communication to the backend datastore through the use of endpoints. The endpoints handles the actual network connection

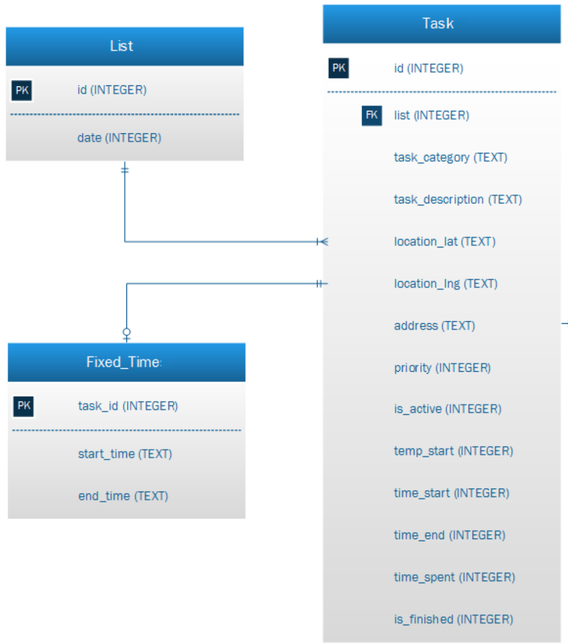



Figure 7: SQLite representation of the individual installations application data.

which relieves the developer of having to manually handle this. We did not provide functionality for synchronization of local and remote data. This is not critical, as the intended use was with students in an educational environment, where the mobile device would most of the time be connected to a network, either mobile or WiFi. Although this could potentially lead to some loss of data, this would be tolerable for this thesis as it would only represent a small fraction of the total amount of data.

For actually storing the data, we decided to go with a normalized database approach for the local database. The representation of the data for the application is shown in Figure 7 and Figure 8. The actual tasks are held in the table *Task*. Each task is tied to a specific list, represented by the *List* table. As a task can have an optional fixed starting and/or end time, we pulled these values out of the Task table and into the *Fixed Time* table. All the collected contexts related to the tasks are held in the *Context* table. The remote datastore is not normalized however, as there is no primary or foreign keys in the AppEngine datastore. This is not important, as the application only writes to the remote datastore.

The actual collection of the context data happens when the users starts a task from the application. A background service is then started and runs on the device until the task is paused by the user or completed. The service collects these contexts at certain intervals. A service to run in the background was needed in order to continuously collect contexts without disturbing the user. The user is also not actively using the mobile device when performing a task, so by implementing a service we could collect contexts even when the device is in an idle state. If a collected context, for example location, is already collected for a specific task then that context is not stored again. Multiple identical locations for a single task is not needed, so this prevents some information

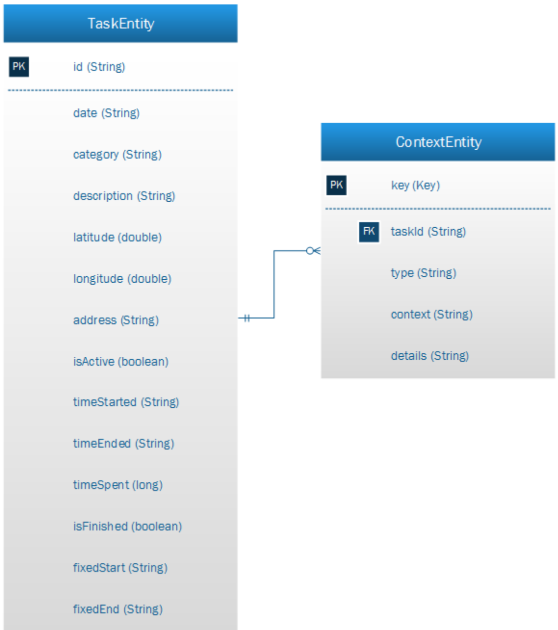



Figure 8: The Google AppEngine datastore representation for all installations application data.

overhead.

## 3.4 Recommender

When creating the recommender part of the application, several decisions needed to be made. First of all, we needed to decide what kind of recommendations to make. There where many possible ways to do this:

- Location proximity recommendations.
- Recommendations based on time of day.
- Recommendations based on time spent on previous tasks.
- Recommend tasks with fixed starting times.
- Recommend tasks based on the shortest traveling distance between tasks.
- Recommendations based on regularity of task occurrences.
- Combinations of the above.

After deciding what to recommend, the underlying logic also needed to be decided. A recommender will need some form of logic for comparison, so that it can know that it should recommend one task over another. Such logic already exist in some systems. We have seen Netflix[21] recommend movies and Amazon[22] recommend books to name a couple. It is potentially beneficial to analyze other systems for reusable recommendation algorithms.

| Task category | Probability |
|---|---|
| Attend lecture | 0.11 |
| Other | 0.24 |
| Practical work | 0.46 |
| Read | 0.14 |
| Write report | 0.05 |

Table 1: Example recommendation probability hashmap

### 3.4.1 Chosen recommendation algorithm

Implementing a recommender algorithm based on neural networks would require a lot of work, probably more than what can be achieved in the scope of this thesis. By using probability calculations however, we can create much simpler mathematical formulas to calculate the probability of different tasks. This is the approach that was used in this thesis. As mentioned earlier, the input for the recommender consists of the user's planned tasks, the current context, and the entire task history with related contexts. The recommender starts by looking at the categories of the user's planned tasks and tries to calculate a probability for each of them, representing how likely the task is going to be chosen next by the user. This calculation is done based on the tasks that the user has previously done and in what contexts. Probability calculations are made individually for each of the different contexts, but also for combinations of them. The probability for a specific task is calculated by following the following formula:

$$P(t(ca)) = \frac{\sum_{i=1}^{n} t_i(ca)}{\sum_{i=1}^{n} t_i(c)}$$

t(ca) represents a task that matches a given category and context and t(c) represents a task that matches the given context. We then get the probability P(TC) of a task category. In other words, the probability represents the relationship between all tasks of a given category and context, and all tasks with the given context.

As an example, the recommender could consider the current time of day. The recommender will then find other tasks that have been done during the same time of day previously. By dividing the total number of a particular task category performed at that time of day by the total number of tasks performed at the same time of day, we get the probability for a task category for that time of day. This is done for each of the task categories that the user has planned and the result is put into a map, as shown in Table 1.

The same process is executed for other contexts as well. For example, based on the current location of the user, the recommender finds all tasks that have been performed in that particular location. Probabilities are then calculated the same way as for the time of day. The task probability map based on location may look like Table 2.

Similar maps are calculated for each context and all combinations of them. This means:

- Time of day
- Day of week
- Location

| Task category | Probability |
|---|---|
| Attend lecture | 0.41 |
| Other | 0.13 |
| Practical work | 0.25 |
| Read | 0.16 |
| Write report | 0.05 |

Table 2: Location recommendation probability hashmap

| Task category | Probability |
|---|---|
| Attend lecture | 0.41 |
| Practical work | 0.46 |

Table 3: Resulting probability hashmap upon which the actual recommendation is made

- Time of day *and* location
- Time of day *and* day of week
- Day of week *and* location
- Time of day *and* day of week *and* location

The highest probability from each of these individual probability maps are then put into a resulting map, where the highest probability in this map will be the actual task recommendation made by the recommender. Looking at Table 1 and Table 2, the resulting map will look like Table 3. The actual recommendation made by the recommender in this example would be the task with the category "practical work".

## 3.5 Application distribution and user experiments

To test our design decision we needed users to test the application for a certain period of time. Because the application is based on task and context history, it would take some time and usage in order for history to build up. A large history of tasks and contexts is needed so that for the recommender can make reasonable recommendations. The recommendations from the recommender should increase in accuracy over time. By accuracy we mean tasks that matches the tasks that the users intend to do next. We could say that the application is "learning" the users behavior over time. For this experiment, we considered 2-3 weeks to be enough time to get some useful data from the application usage. Ideally the experiment would run over an even longer period, but having a timeframe for this thesis a maximum timeframe for the experimenting needed to be set.

The application was uploaded to Google Play for distributing. This made the application easily available for anyone who wished to test it. Requesting participants for the application testing was done by mailing students at Gjøvik University College. The application was found by following the link provided by mail, or searching for "Todoity" in Google Play.

# 4 Results

## 4.1 Application design

### 4.1.1 Student tasks

In order to find potential and noteworthy differences between the tasks of postgraduate students and undergraduate students a questionnaire was made. The questionnaire was also aimed at finding how much these students are using task management applications such as to-do lists or calendars, both in terms of regularity and frequency. The questionnaire yielded 105 responses where 62.9% of these were Bachelor degree students. The remaining 37.1% were Master or PhD students. This was determined by the first question in the questionnaire.

We also wanted to know whether or not these students typically make lists of the things that they have to do. The students were asked to reply to the following statement: *I make a list of the things I have to do during a day/week*. Possible responses to the statement were: *Never*, *1-2 times per week*, *Almost every day* and *Every day, multiple times*. The responses among all students are shown in Figure 9.

We also wanted to get an overview over the amount of students who were using task management applications at all. For this, we asked the question *Do you use task/time management applications such as calendars or to-do lists?*. 64.8% replied *Yes*, while the remaining 35.2% replied *No*.

An important part of the questionnaire was to find out the regularity at which students do certain types of tasks. For this, we listed the predefined tasks and asked the students to reply on a scale of how regular they did the tasks. The question were *How regular do you perform the following tasks?* The scale that was used in this case consisted of *Every day*, *Every week*, *Every month*, *Irregular, zero or few times per week* and *Irregular, zero or few times per month*. This question can let us know what kind of tasks are done on a regular basis and what tasks are done more randomly. The responses are shown in Figure 10

A very similar question was formed to provide feedback on how often the different tasks were being done. On the question *On average, how often do you perform the following tasks?* the students were asked to reply on a scale for each of the predefined tasks. The scale consisted of *Less than once per week*, *1-2 times per week*, *3-4 times per week*, *About once per day* and *Multiple times per day*. Responses to this question are shown in Figure 11.

In order to find out whether our predefined tasks were enough to cover all the tasks that a student typically does, we devised another question where the students were to input any other tasks that they typically did. The question and the responses are shown in Table 4

### 4.1.2 Collected student tasks

From the application user experimenting at the end of the thesis, a total of 46 tasks were logged in the external database. This is too few to make any strong conclusions upon the data, but it might be possible to make some small indications. A small sample of the collected tasks and its

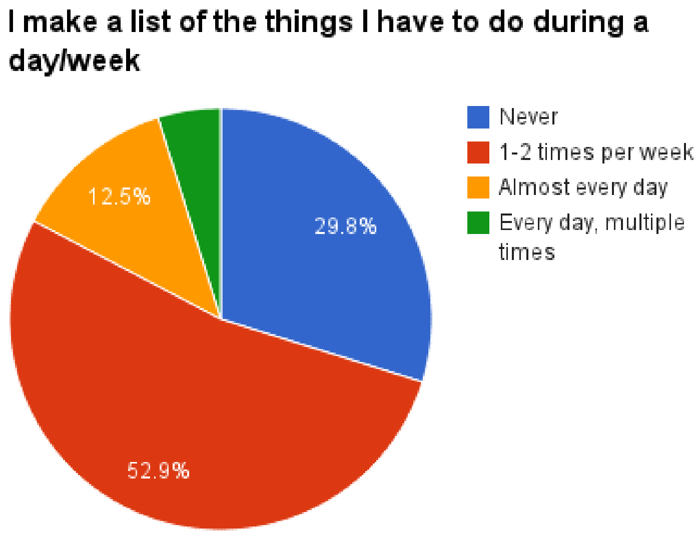



Figure 9: The distribution of how often students make to-do lists.

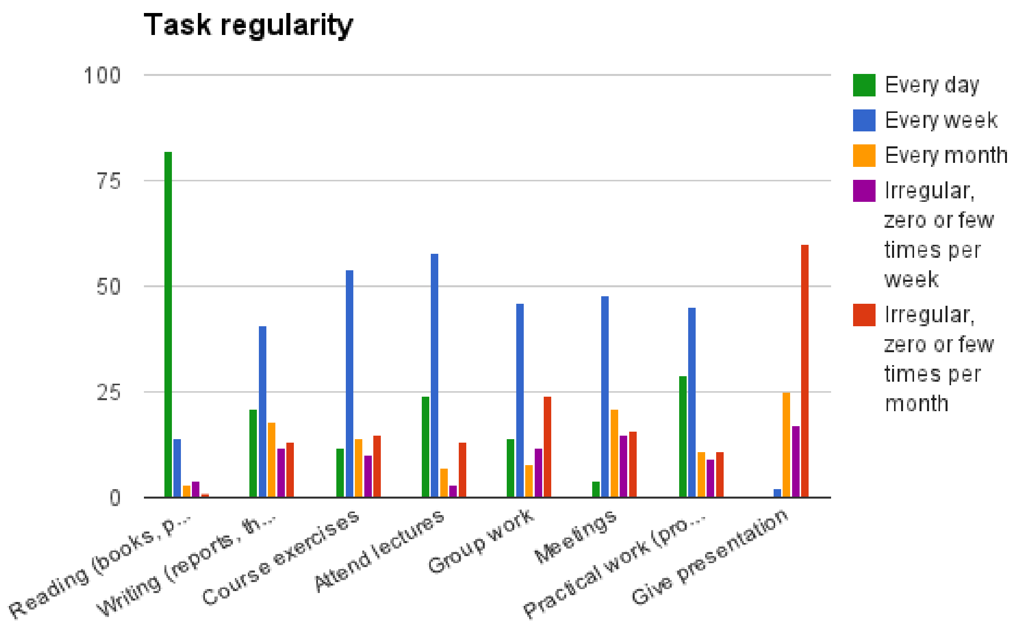



Figure 10: The distribution of how regular particular tasks are done.

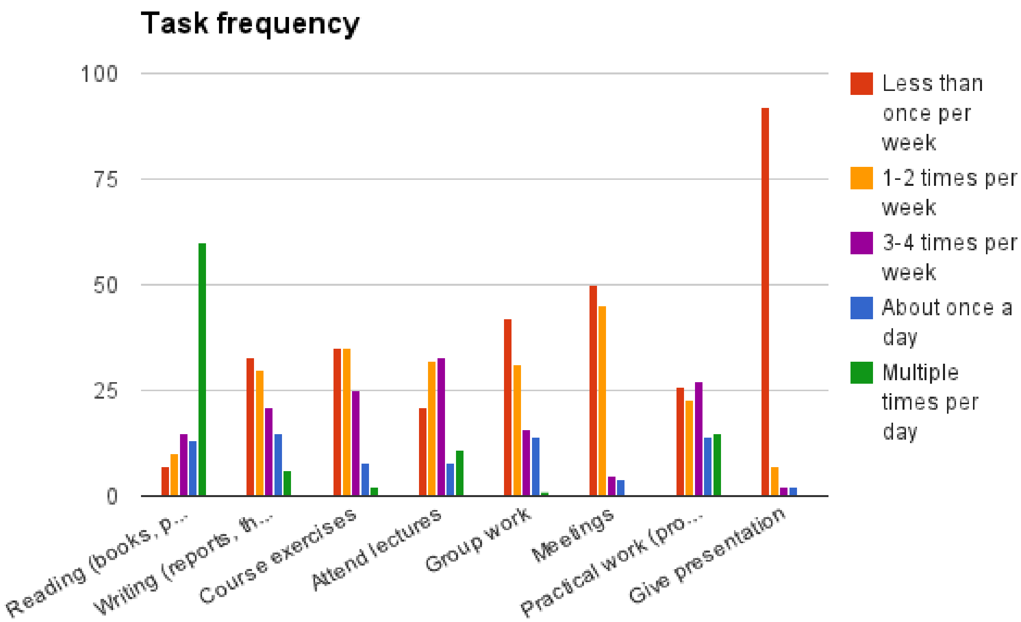



Figure 11: The distribution of how often (frequent) particular tasks are done.

| Task | Frequency |
|---|---|
| Reddit | All the time |
| Carry out experiment | Once a month |
| Part-time job (4) | 35"%" |
| Searching for articles (2) | |
| Gaming | |
| Workout/exercise | 4-5 times a week |

Table 4: Responses to the question *What other tasks that were not previously mentioned do you do, and how often?* The number in parenthesis shows how many students responded the same thing.

| ID/Name | address | category | date | description | fixedEnd | fixedStart | isActive | isFinished | latitude | longitude | priority | timeEnded | timeSpent | timeStarted |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| name=1efaad2a-5293-4c79-a9c8-0b2abb325767 1 | Krohnåsvegen, 5239 Bergen | Meeting | Fri, Oct 24, 2014 | Med Erna | | | FALSE | FALSE | 60.2973212601 | 5.329679735 | 1 | Fri, Oct 24, 2014 | 32104 | Fri, Oct 24, 2014 |
| name=244ad5aa-0a98-4ee9-ba0a-0a2c742f4810 1 | Gjøvik, Norway | Other | tor., okt. 30, 2014 | Halloween-fest!! | 00:00 | 19:00 | FALSE | FALSE | 60.7954302 | 10.6916303 | 1 | tor., jan. 01, 1970 | 0 | tor., jan. 01, 1970 |
| name=244ad5aa-0a98-4ee9-ba0a-0a2c742f4810 2 | | Group work | man., okt. 27, 2014 | Ingeniørrollen | 16:00 | 15:00 | FALSE | FALSE | 0.0 | 0.0 | 2 | tor., jan. 01, 1970 | 0 | tor., jan. 01, 1970 |
| name=244ad5aa-0a98-4ee9-ba0a-0a2c742f4810 3 | | Practical work | man., okt. 27, 2014 | Øving - gr.prog. | 19:00 | 16:00 | FALSE | FALSE | 0.0 | 0.0 | 1 | tor., jan. 01, 1970 | 0 | tor., jan. 01, 1970 |
| name=244ad5aa-0a98-4ee9-ba0a-0a2c742f4810 4 | Gjøvik, Norway | Other | tir., okt. 28, 2014 | Kiropraktor | 15:00 | 14:00 | FALSE | TRUE | 60.7954302 | 10.6916303 | 1 | tir., okt. 28, 2014 | 1745956 | tir., okt. 28, 2014 |
| name=244ad5aa-0a98-4ee9-ba0a-0a2c742f4810 5 | | Attend lecture | ons., okt. 29, 2014 | Gr.prog | 18:00 | 16:15 | FALSE | TRUE | 0.0 | 0.0 | 3 | tor., jan. 01, 1970 | 0 | tor., jan. 01, 1970 |
| name=244ad5aa-0a98-4ee9-ba0a-0a2c742f4810 6 | | Group work | ons., okt. 29, 2014 | Ingeniørrollen | 13:00 | 12:00 | FALSE | TRUE | 0.0 | 0.0 | 2 | tor., jan. 01, 1970 | 0 | tor., jan. 01, 1970 |
| name=244ad5aa-0a98-4ee9-ba0a-0a2c742f4810 7 | CC Gjøvik, Jernbanesvingen, Gjøvik, Norway | Other | ons., okt. 29, 2014 | Brodder, tannbørste, vippelim, | 20:00 | 18:30 | TRUE | FALSE | 0.0 | 0.0 | 1 | tor., jan. 01, 1970 | 0 | tor., jan. 01, 1970 |
| name=2ba71ded-364e-43d7-b9e8-af8d2c73e7a3 1 | Søndre Skøien veg 36A, 2730 Lunner | Read | man., okt. 13, 2014 | Service Managemebt | 21:47 | 19:47 | FALSE | FALSE | 60.2895961758 | 10.5570673943 | 2 | man., okt. 13, 2014 | 1865242 | man., okt. 13, 2014 |
| name=2ba71ded-364e-43d7-b9e8-af8d2c73e7a3 2 | Søndre Skøien veg, Lunner, Norway | Other | man., okt. 13, 2014 | Play FM | | | FALSE | FALSE | 60.289062 | 10.5559101 | 1 | man., okt. 13, 2014 | 1786380 | man., okt. 13, 2014 |
| name=2e687ced-9b43-4c21-b5f2-0e1b41e9a868 1 | Brurudbakka 2, Fall, Norway | Attend lecture | tir., okt. 21, 2014 | Pluralsight, C# course | | | FALSE | FALSE | 60.6722075 | 10.3656625 | 1 | tor., jan. 01, 1970 | 0 | tor., jan. 01, 1970 |
| name=2e687ced-9b43-4c21-b5f2-0e1b41e9a868 2 | Brurudbakka 2, Fall, Norway | Write report | tir., okt. 21, 2014 | Background | | | FALSE | FALSE | 60.6722075 | 10.3656625 | 2 | tor., jan. 01, 1970 | 0 | tor., jan. 01, 1970 |
| name=5dc1e32f-bd6c-4451-a338-796915b8b599 1 | Høgskolen i Gjøvik, Teknologivegen, Gjøvik, Nor | Meeting | man., okt. 13, 2014 | Mariusz | 15:00 | 14:00 | FALSE | FALSE | 60.7895337 | 10.6817954 | 1 | tor., jan. 01, 1970 | 0 | tor., jan. 01, 1970 |
| name=66d761a4-c2fb-45c3-9179-c8984fab67b2 1 | Høgskolen i Gjøvik, Teknologivegen, Gjøvik, Nor | Write report | fre., okt. 24, 2014 | | | | FALSE | FALSE | 60.7895337 | 10.6817954 | 1 | tor., jan. 01, 1970 | 0 | tor., jan. 01, 1970 |
| name=66d761a4-c2fb-45c3-9179-c8984fab67b2 2 | Gjøvik University College, Teknologivegen, Gjøvi | Practical work | fre., okt. 24, 2014 | Send email about job | | | FALSE | TRUE | 60.7895337 | 10.6817954 | 2 | tor., jan. 01, 1970 | 0 | tor., jan. 01, 1970 |
| name=66d761a4-c2fb-45c3-9179-c8984fab67b2 3 | Gjøvik University College, Teknologivegen, Gjøvi | Practical work | lør., okt. 25, 2014 | Work@uni | | | FALSE | FALSE | 60.7895337 | 10.6817954 | 1 | tor., jan. 01, 1970 | 0 | tor., jan. 01, 1970 |

Figure 12: Sample of collected tasks and its format.

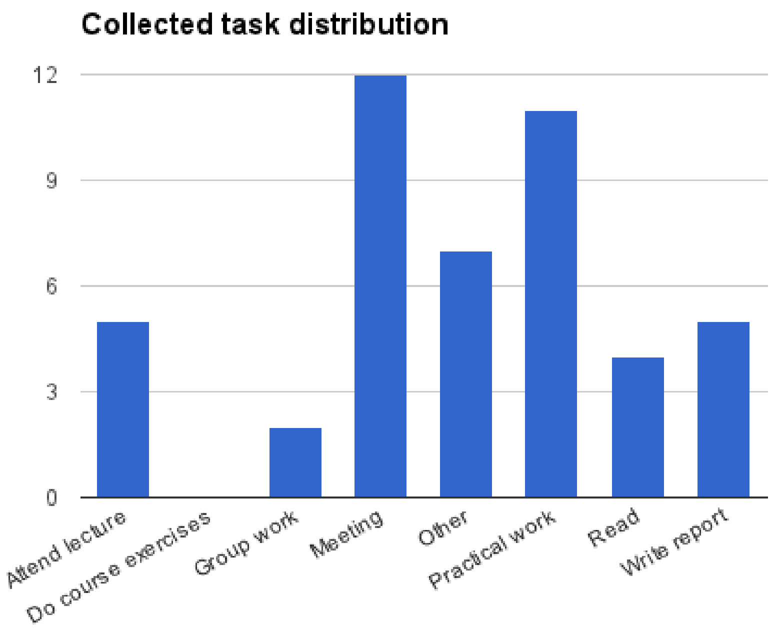



Figure 13: The distribution of the types of tasks among the tasks that were collected.

format is shown in Figure 12.

The distribution of the types of tasks that the students actually did is shown in Figure 13. This shows that *Meeting* was the type of task that were logged the most in the usage of the application, with 12 tasks being of this type. This might indicate that users typically log important tasks that they need to remember, and is therefore using the task entry as a reminder for the meeting.

Since we used a very limited set of predefined tasks, it would seem logical that the type *Other* would receive a large share of the total amount of tasks. Although the intended usage was in an educational environment, this is not something that was controlled during the application usage experiment. It is then likely that some students would use the application for purposes other than just educational ones. However, only 7 (15%) of the tasks were categorized as *Other*. This would indicate that most of the student actually used the application for educational purposes.

The indications that the students were using the application as a reminder of events is further

| Task category | Number of tracked tasks | Average time spent (minutes) |
|---|---|---|
| Attend lecture | 1 | 62.1 |
| Do course exercises | 0 | 0 |
| Group work | 0 | 0 |
| Meeting | 3 | 81.7 |
| Other | 2 | 29.4 |
| Practical work | 3 | 152.2 |
| Read | 2 | 31.4 |
| Write report | 2 | 81.9 |

Table 5: Number of tracked tasks in each category and the average time spent on those tasks.

strengthened by looking at the tasks that were created with specific start and end times. Tasks or events that has a specific starting time is generally something that we need to remember. Of the 46 total tasks that were logged, 24 (52%) of these were logged with both a starting time and an end time.

There are also a few other things that are interesting to look at. One is whether or not the users actually "ticked" or "checked" the tasks after completion as to indicate that the task is done. We could also check the amount of time spent on the different tasks, which would be available for the tasks that were actually tracked in the application.

Few of the tasks were ticked off as to indicate that they were finished. Of the 46 tasks, only 12 (26%) were said to be completed in the application. The students probably completed more of the tasks than this would suggest, so it is likely that they simply forgot or chose not to "tick off" the task in the application.

Very few tasks were tracked with actual contexts in the application. Only 13 (28%) had actual contexts related to them. The task categories that were tracked and the average time spent on these tasks is shown in Table 5. We can see that the distribution of the task categories for the tasks that have been tracked with contexts are quite similar to the distribution of all the tasks registered in the application (see Figure 14).

### 4.1.3 Design decisions

Through the application development process and user testing we learned that some design concepts worked well and some did not. One design decision that did not work as well as intended was the overall application workflow. The process of actually creating a new task in the application was slightly overcomplicated by having to go through a specific list in order to do so. This was confirmed by a couple of students who wrote this in an email as a response to the requesting of participants.

Another design decision that did not work well was how we chose to start and stop the collection of contexts in the background as the user performed a task. Whether or not contexts were collected for a task was entirely up to the user remembering to manually start the task from the application. By the number of tasks that contexts were collected for compared to the total number of tasks, it was evident that this was not a good design.

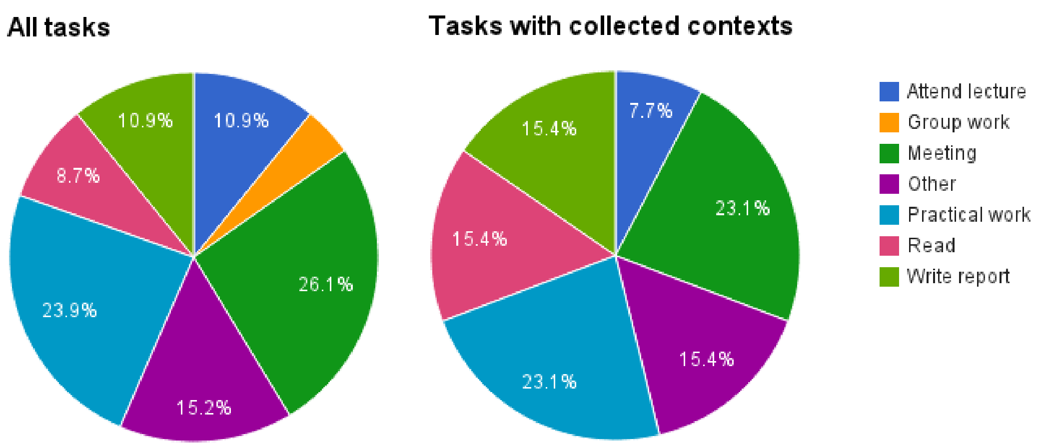



Figure 14: The distribution of the types of tasks among all registered tasks and the ones with collected contexts.

| ID/Name | context | details | taskId | type |
|---|---|---|---|---|
| id=4646334543953920 | Blåvarpvegen 11, 2843 Eina | 60.6307906 10.6027573 | d3c42003-06c6-4f1f-b2ac-ce416cb97a7d 3 | location |
| id=4760417733705728 | Hennungvegen 22, 2760 Brandbu | 60.4710483 10.5000547 | d3c42003-06c6-4f1f-b2ac-ce416cb97a7d 3 | location |
| id=4787072032309248 | Korsrudlinna 15-51, 2730 Lunner | 60.3094413 10.5824436 | d3c42003-06c6-4f1f-b2ac-ce416cb97a7d 3 | location |
| id=4788842901012480 | Sisselbergvegen 16A, 2843 Eina | 60.6229777 10.6014714 | d3c42003-06c6-4f1f-b2ac-ce416cb97a7d 3 | location |
| id=4794120610512896 | Jarenstranda 450, 2770 Jaren | 60.3921713 10.5441286 | d3c42003-06c6-4f1f-b2ac-ce416cb97a7d 3 | location |
| id=4803736505417728 | Hennungvegen 13, 2760 Brandbu | 60.4681806 10.5007398 | d3c42003-06c6-4f1f-b2ac-ce416cb97a7d 3 | location |
| id=4805985726103552 | Flubergvegen 217, 2860 | 60.7156083 10.3187252 | fbc80650-ba11-4c04-a50a-6c17b385cc49 8 | location |
| id=4806851430449152 | still | | 244ad5aa-0a98-4ee9-ba0a-0a2c742f4810 4 | activity |
| id=4813809814339584 | Grinilinna 5, 2750 Gran | 60.3459572 10.5681021 | d3c42003-06c6-4f1f-b2ac-ce416cb97a7d 3 | location |
| id=4817644414828544 | on foot | | d3c42003-06c6-4f1f-b2ac-ce416cb97a7d 4 | activity |
| id=4823184117334016 | Øvre Skiakersgutua 5, 2750 Gran | 60.3586841 10.5484704 | d3c42003-06c6-4f1f-b2ac-ce416cb97a7d 3 | location |
| id=4834703488057344 | Horgenmoen 26, 2750 Gran | 60.3682843 10.5677752 | d3c42003-06c6-4f1f-b2ac-ce416cb97a7d 3 | location |
| id=4837813581250560 | Bjørgevegen 92, 2730 Lunner | 60.3216622 10.575379 | d3c42003-06c6-4f1f-b2ac-ce416cb97a7d 3 | location |
| id=4858703228436480 | Bleikenvegen 112, 2760 Brandbu | 60.4684927 10.500618 | d3c42003-06c6-4f1f-b2ac-ce416cb97a7d 3 | location |

Figure 15: Sample of collected contexts and their format.

## 4.2 Context collection

Looking at the data collected from the users tasks in Section 4.1.2, we can see that few tasks have been logged with an actual “time spent” value. The tasks that do not have this value, are tasks that have not been logged with contexts. This is because the users did not press the in-app start button, which means that they either forgot to do so, or that they ignored that part of the intended work flow. This is an indication that having users tracking their own tasks in this manner opens for inconsistencies and inaccuracies in the logging. It is hard for users to remember to log a task prior to the execution of the task. A sample of the collected contexts and their format are shown in Figure 15.

We got a total of 206 collected contexts from all users for all tasks. 13 (6%) of these were time contexts. Time contexts were only logged when the users remembered to both start and stop/finish the tasks. The application gave the tasks timestamps a when the were started/paused/finished and from this we could also calculate the time spent on the tasks. 160 (78%) of the contexts were location contexts while the remaining 33 (16%) were activity contexts. Activity contexts were predefined in an abstracted textual representation, as provided by the Android framework:

- On foot
- On bicycle
- In vehicle
- Still
- Tilting (the activity is currently changing state, and the new state is not yet known)
- Unknown

As with location contexts, activity contexts would not be stored twice for a task if that specific context already existed for that task. This means that the maximum number of activity context for one task is 6. For location contexts, there are no such limitations by a set of definitions, so that all the locations the the user would be at during the tracking would be logged. This is why there are many more location contexts than activity contexts.

Another reason for the many location contexts is that several users seemed to leave tasks as active while being on the move, such as driving a car, probably because they forgot to turn tracking off when they stopped doing the task. This belief is reinforced by the fact that the tasks in question also had the activity context “in vehicle” attached to them. These tasks had many collected location contexts, one with as many as 99 different locations.

Another thing is to see if the actual location that the users performed their tasks coincides with the planned location of the tasks. This was true for only 5 (38%) of the 13 tasks. These 5 tasks had location contexts that matched the planned location of the task. 4 of the tasks had no location context, only activity contexts, logged. This leaves 4 tasks where the collected location context did not match the planned location of the task.

# 5 Discussion

## 5.1 Application design decisions

The initial goal of the thesis was to prove the value of utilizing context information in a task management system. In order to do that, we would need a lot of usage data to support such claims. In the ideal case, and that which was intended at first, we would have plenty of users using the application for a long period of time. Not just that, but the users would also follow the intended workflow of the application, and use it as was intended through the development process. If all of these would have been true, we could have looked at the collected tasks data, and possibly made some conclusions about those data. For example, it might have been possible to look at the average time spent on tasks, and see that this would decrease over time as the app was used more and the recommender learned more of its users behaviors. We could have also been able to conclude about app usage, such as the user becoming better to track their tasks the more time they spent using the application. Of course, the opposite may also be true, that we would have been unable to conclude these things, but the results would still be more valid as the collected amount of data upon which conclusions would have been made would have been much larger.

We could also have used different approaches to try to indicate the usefulness of context information in task management systems. For example, we could have designed a different experiment for testing the application. By using two applications instead of one, we could directly compare the usage and collected data from both of the applications. One application could then be the already created one, as described in this thesis, whereas the other one could be an identical application with the exception of using the recommender and the contexts. We would then be able to compare the two applications directly, provided that the users using the application, are from the same population.

The general application workflow mentioned in Section 3.2 might also not have been optimal. Although the choice of making lists and assigning new tasks to lists might be a good idea, our application forces the user to go through a specific list when a task is created. This might not be ideal, as creating a task is something the user does very often. An option to create a new task directly should have been added, so that the user would not have to go through a specific list. When creating a new task independent from a list, the task could default to *today*'s list.

Too much focus was given to usability principles and general design guidelines in the development of the application. These things are necessary to discuss and evaluate, but for a proof of concept application we could have chosen not to implement some of the principles. One example is the swiping between tabs in the main screen of the application. While this is a feature that users are familiar with in such applications, we could have instead created a simple menu for changing between displaying tasks for today, the week, and all tasks. Implementing easy and user friendly swiping led to some problems when it came to updating neighboring tabs. These issues and other similar issues stole some time, which could have been better used for getting more users for the application testing and also having a longer testing period. This way we could have had more conclusive data from the actual usage of the application.

### 5.1.1 Task collection issues and improvements

The application stores tasks on the users behalf as well as tracking and storing contexts related to those tasks. This is something that would need explicit concent from the user and would also need to be handled properly, in order to avoid violating privacy regulations. We then need to inform the users of the application of what is being collected and stored about the user and his/her tasks as well as why we need to store this information. In our application we have done this through an introductory dialog when the user first starts the application. The user has to accept the written policy in this dialog before he or she can use the application. Though this may be good enough for the purpose of this application, a full scale commercialized application would have to investigate this issue further.

Some considerations have been made towards privacy when storing the application data. On the local device database, the data is stored without obscuring it in any way, as others generally have no access to this data. However, on the remote database, we have applied some sort of anonymization. Here, the individual users are hidden by storing the installation id of the application rather than the users themselves. The contexts however, are not anonymized and could eventually be used to recognize certain users through pattern matching. This could have been handled in this thesis by applying some anonymization over at least the location context data. We would still get the same results as long as identical contexts are anonymized identically.

In a commercialized application there might not be a need for a remote database at all, but it could have some potential uses other than research purposes. For example, when recommending a certain ordering of tasks, one could look at the large pool of data in the remote datastore and calculate the average time spent on certain tasks. This could help the recommender make more informed decisions on the ordering of tasks, especially if the user does not have a large internal task and context history built up yet.

## 5.2 Context collection

There was one large problem when collecting context information when the users were doing a task: knowing *when* to collect the context data. For this application, we decided to implement a simple button to start and stop the tracking of contexts when doing a task. Issues regarding this approach were mentioned in Section 4.2. The ideal way to collect context would be for the user to not having to think about the collection of such data. The application would instead have an autonomous approach of handling this. Of course, such autonomy would be difficult to develop, and also reach outside of the scope of this thesis. However, one way that this could have been done is to let the application collect contexts all the time. The collected contexts would then be assigned to a task when the user finishes a task. Upon the user finishing a task, the application could look back in time a certain time interval and then determine that the contexts collected in that timeframe is the contexts related to the finished task. This approached is also somewhat flawed, as it assumes that the user tells the application that he/she has finished a task, i.e. "ticks off" the task, at the actual time the the user finishes it. In reality, many users would sometimes do this at a later point in time, meaning that the contexts assigned to a particular task could be the wrong ones. This error could be reduced over time however, as the application could "learn" the typical contexts related to individual tasks.

Another way to solve the problem could be to have the application ask the user to provide the timeframe in which he/she actually did the task when the user "ticks off" the task. This way we could assign the correct contexts to the correct tasks. The problem with this approach is usability. The users would not appreciate having to input times whenever they finish tasks. Unnecessary and repetitive operations should be avoided. This option would therefore not be suitable in a commercialized application, but could potentially be used for experimenting purposes.

As the application's user testing for this thesis only lasted for a little more than two weeks, we did not implement any restrictions to prevent old context data from being used. It is possible that, over time, some context information may eventually be outdated. Location contexts for example, could change dramatically if a user moves to a new home or apartment. The location contexts related to the previous place of residence is then no longer relevant. A mechanism for preventing such scenarios should be present in the application. For example, we could limit the task and context history to a month, meaning that tasks and contexts that are older than this will not be used. This way, the application could "relearn" new contexts over time.

There is also a limitation to how we stored the activity contexts for the tasks. By storing a particular activity only once, we prevent information overhead, but we also deny ourselves the ability to see the duration of activities when the user is doing a task. If we collect all activities regardless of whether or not an activity has been collected for a particular task earlier, we can see the approximate duration of the individual activities. This could be useful for research purposes.

## 5.3 Recommender and context usage

As a way to test the usefulness of contexts in a to-do list, we designed a recommender to utilize the collected context and task history. In this thesis, the recommender was somewhat simple and provided recommendations based on only the criteria mentioned in Section 3.4. Even though this way of making recommendations can be good in many cases, the recommender could be further improved by implementing more rules. These rules could be general, or task specific. An example of a task specific rule could be that the recommender assumes a task of category *meeting* to be ended if the location suddenly changes by a lot after the meeting has been already started. A more general rule could be the assumption that a task is finished if the activity changes into *in vehicle* and location updates starts coming in frequently. The user is then driving a car, and it is unlikely that he/she is actively performing a planned task.

More rules the the ones mentioned above could be implemented. However, it is more likely that the rules that would be feasible in a commercialized application is the general ones. This is because some of the predefined tasks are very specific for the user group in this thesis work, and would change if tasks were to be generalized.

Another thing that also should have been taken into account is the importance of the different contexts. When making a task recommendation based on contexts, some contexts may be more important than others. To make the recommender account for this, we could have added weights to the different contexts. For example, time contexts may be more relevant than location contexts for certain tasks. We could then write this into our recommendation algorithm. If one task was given a probability based on time context and another task was given the same probability based on location context, the recommender would choose the task whose context is more highly weighted. This is of course more complicated than only comparing equal probabilities. Mathematical expressions would have to be made that allow the comparison of task probabilities directly.

Instead of actually recommending an ordering of tasks with a most recommended task on top, the recommender could instead *not* recommend tasks. By that we mean that the recommender could look at all previous tasks with their related contexts, as before, and decide which tasks are *not* suitable given the current context of the user. For example, if a user always does one particular task at one specific location, this task can then be ignored if the user is at a different location. This way, the recommender is recommending tasks through a process of ruling out other tasks. A set of tasks that *are* relevant to the current context is then presented to the user. This way of recommending tasks may be especially useful if the user has a very long list of tasks that needs to be done.

## 5.4 Generalization of tasks

Too reach a wider range of users, it is possible to generalize the types of tasks available to the users. The task *attend lecture* for example, is highly relevant to a student, whereas to more general users it is highly irrelevant. However, generalizing tasks to a large extent could lead to other difficulties. We could make a very general category *work* that applies to most users, but as tasks within this category can differ greatly both in terms of what needs to be done as well as timespan, it will be very difficult for a recommender to compare tasks of this category, and thereby make reasonable recommendations to the user.

One approach that would help solve this, is that instead of trying to generalize tasks too much, we could expand the number of task categories greatly. The more specific a task category can be, the easier it is for the recommender to make good recommendations for the user. So instead of having a small set of general tasks, we could have a large library of predefined task categories, which could be divided into category sets. For example, the application could let the user select his occupation, and would thereby load a set of predefined task categories that matches the chosen occupation. This would require a lot of work however, as different occupations and their typical tasks would need to be matched together.

- Experiment desig flaws, not able to identify students or find the ones who answered what in the first questionnaire.

# 6 Conclusion

This study has looked into the process of designing a task management application that makes use of context information. Through this process we have found some of the aspects that are important to focus on when designing such an application. For actually utilizing the context information, we designed a recommender to recommend a task and an ordering of tasks for the user.

## 6.1 Research question 1

## 6.2 Research question 2

## 6.3 Research question 3

# 7 Future work

As this thesis covered rather broad area, several aspects will need to be looked in more detail. First of all, it would be interesting to see a similar study being done on a different user group, and also with a longer application testing period with more participants. Similar studies with other user groups are needed in order to be able to generalize the overall design of the application ti fit a wider and more general user group.

It would also be very useful too see a similar study being conducted on a larger task management application such as a calendar, rather than a to-do list. This would require some more work, as a calendar system is naturally larger and more complex than a simple to-do list, but by discovering that context information adds great value to a calendar system would be a big step towards making context valuable for all task management systems.

In our proof of concept application we decided that a recommender would be a good way to test the usefulness of context information in a to-do list. This was used to suggest tasks for the user. Investigating other ways of utilizing contexts in task management systems would also be useful. For example, one could use contexts to keep a calendar very much up to date with the users events. By learning the users behavior, the calendar could create entries in the calendar if the user has not already done so. The calendar could infer the user's activity by looking at contexts.

# Bibliography


[1] Google calendar. `https://www.google.com/calendar`.

[2] Trello. `https://trello.com/`.

[3] Todoist. `https://en.todoist.com/`.

[4] Verbert, K., Manouselis, N., Ochoa, X., Wolpers, M., Drachsler, H., Bosnic, I., & Duval, E. 2012. Context-aware recommender systems for learning: a survey and future challenges. *Learning Technologies, IEEE Transactions on*, 5(4), 318–335.

[5] Liu, W., Li, X., & Huang, D. 2011. A survey on context awareness. In *Computer Science and Service System (CSSS), 2011 International Conference on*, 144–147. IEEE.

[6] Dey, A. K. *Providing architectural support for building context-aware applications*. PhD thesis, Georgia Institute of Technology, 2000.

[7] Chalmers, M. 2004. A historical view of context. *Computer Supported Cooperative Work (CSCW)*, 13(3-4), 223–247.

[8] Hong, J., Suh, E.-H., Kim, J., & Kim, S. 2009. Context-aware system for proactive personalized service based on context history. *Expert Systems with Applications*, 36(4), 7448–7457.

[9] Ciaramella, A., Cimino, M. G., Lazzerini, B., & Marcelloni, F. 2010. Using context history to personalize a resource recommender via a genetic algorithm. In *Intelligent Systems Design and Applications (ISDA), 2010 10th International Conference on*, 965–970. IEEE.

[10] Mayrhofer, R. 2005. Context prediction based on context histories: Expected benefits, issues and current state-of-the-art. *COGNITIVE SCIENCE RESEARCH PAPER-UNIVERSITY OF SUSSEX CSRP*, 577, 31.

[11] Anagnostopoulos, C., Mpougiouris, P., & Hadjiefthymiades, S. 2005. Prediction intelligence in context-aware applications. In *Proceedings of the 6th international conference on Mobile data management*, 137–141. ACM.

[12] Refanidis, I. & Alexiadis, A. 2011. Deployment and evaluation of selfplanner, an automated individual task management system. *Computational Intelligence*, 27(1), 41–59.

[13] Refanidis, I. & Yorke-Smith, N. 2010. A constraint-based approach to scheduling an individual's activities. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 1(2), 12.

[14] Norton, N. & Lim, S. 2010. Towards a context-aware personal scheduler. In *International Conference of Impact on Ubiquitous IT Co-Design to Industry*, 21–23.

[15] Conley, K. & Carpenter, J. 2007. Towel: Towards an intelligent to-do list. In *AAAI Spring Symposium: Interaction Challenges for Intelligent Assistants*, 26–32.

[16] Myers, K. L. & Yorke-Smith, N. 2005. A cognitive framework for delegation to an assistive user agent. In *Proc. of AAAI 2005 Fall Symposium on Mixed-Initiative Problem-Solving Assistants*, 94–99.

[17] Myers, K., Berry, P., Blythe, J., Conley, K., Gervasio, M., McGuinness, D. L., Morley, D., Pfeffer, A., Pollack, M., & Tambe, M. 2007. An intelligent personal assistant for task and time management. *AI Magazine*, 28(2), 47.

[18] Driver, C. & Clarke, S. 2008. An application framework for mobile, context-aware trails. *Pervasive and Mobile Computing*, 4(5), 719–736.

[19] Lawler, E. L., Lenstra, J. K., Kan, A. R., & Shmoys, D. B. 1985. *The traveling salesman problem: a guided tour of combinatorial optimization*, volume 3. Wiley Chichester.

[20] Android design principles. `http://developer.android.com/design/index.html`.

[21] Netflix. `https://www.netflix.com`.

[22] Amazon. `https://www.amazon.com`.

# A Student tasks questionnaire

**What degree are you currently taking?**

- Bachelor degree
- Master degree
- Ph.D. degree

**I make a list of the things I have to do during a day/week**

- Never
- 1-2 times per week
- Almost every day
- Every day, multiple times

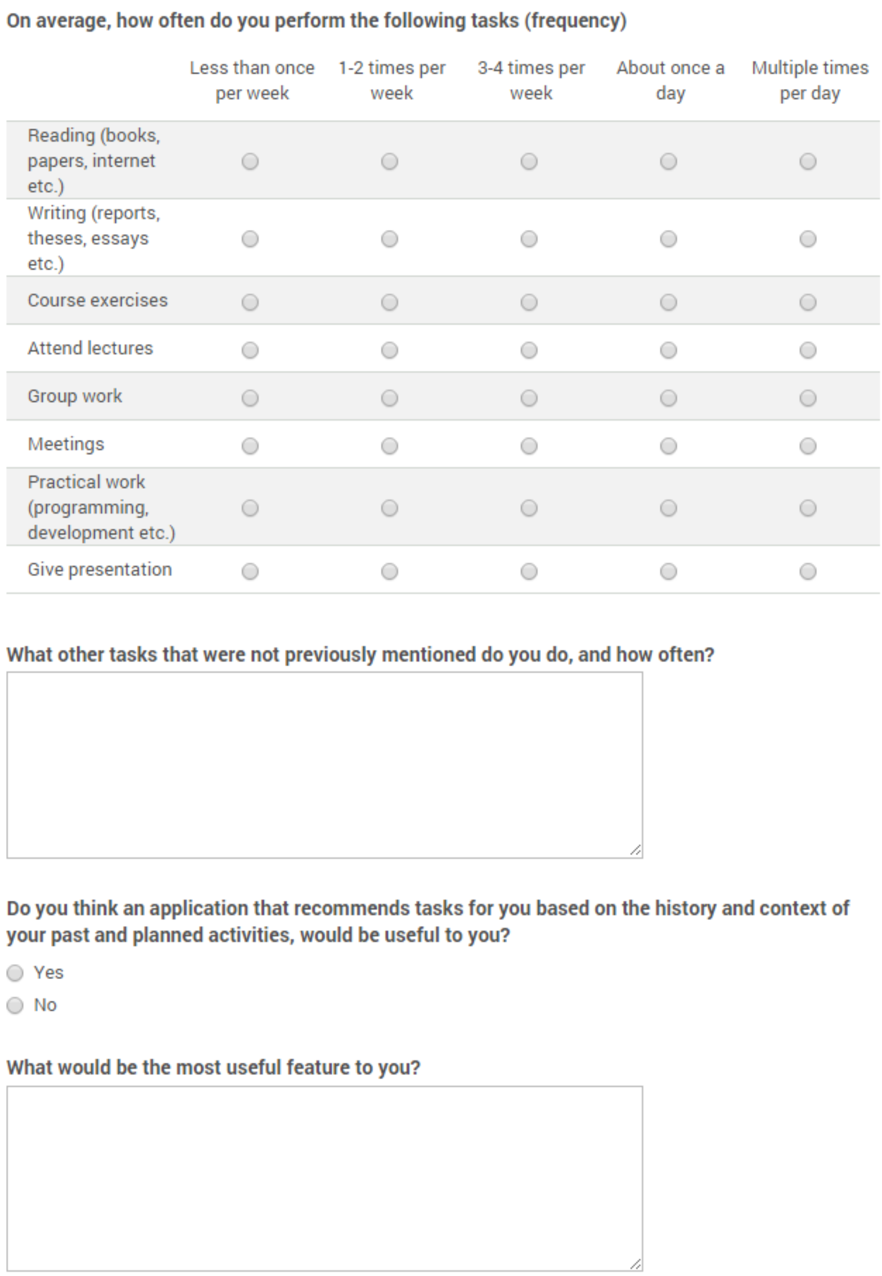
**Do you use task/time management applications such as calendars or todo-lists?**

- Yes
- No

**How regular do you perform the following tasks?**

| | Every day | Every week | Every month | Irregular, zero or few times per week | Irregular, zero or few times per month |
|---|---|---|---|---|---|
| Reading (books, papers, internet etc.) | | | | | |
| Writing (reports, theses, essays etc.) | | | | | |
| Course exercises | | | | | |
| Attend lectures | | | | | |
| Group work | | | | | |
| Meetings | | | | | |
| Practical work (programming, development etc.) | | | | | |
| Give presentation | | | | | |

Figure 16: Student tasks questionnaire part 1.



**On average, how often do you perform the following tasks (frequency)**

| | Less than once per week | 1-2 times per week | 3-4 times per week | About once a day | Multiple times per day |
|---|---|---|---|---|---|
| Reading (books, papers, internet etc.) | | | | | |
| Writing (reports, theses, essays etc.) | | | | | |
| Course exercises | | | | | |
| Attend lectures | | | | | |
| Group work | | | | | |
| Meetings | | | | | |
| Practical work (programming, development etc.) | | | | | |
| Give presentation | | | | | |

**What other tasks that were not previously mentioned do you do, and how often?**

**Do you think an application that recommends tasks for you based on the history and context of your past and planned activities, would be useful to you?**

- Yes
- No

**What would be the most useful feature to you?**

Figure 17: Student tasks questionnaire part 2.

# B Application GUI

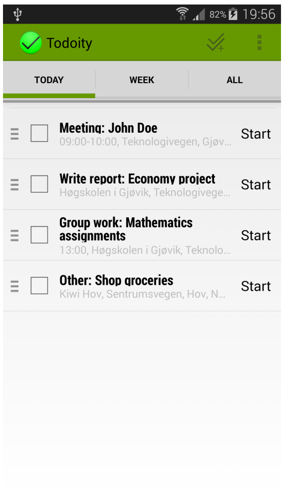



Figure 18: Main application screen design showing tasks for “today”.

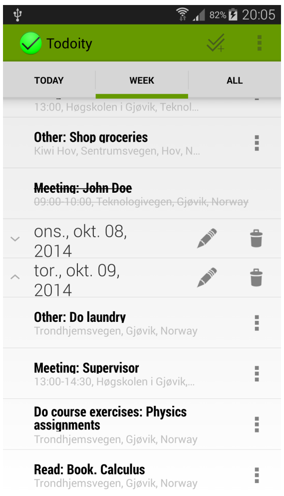



Figure 19: Main application screen design showing tasks for the “week”.

# C Recommender algorithm

```java
public final class Recommender {

    public Recommender() {}

    public static class RecommendTask extends AsyncTask<Void, Void, Void>
            implements ConnectionCallbacks, OnConnectionFailedListener {

        private TodayFragment mFragment;
        private Context mContext;
        private TasksDatabase mTasksDb;
        private LocationClient mLocationClient;

        private Location mLastKnownLocation;
        private long mTimeOfCalculation;
        private HashMap<String, Float> mRecommendationMap;

        private Task mRecommendedTask;
        private ArrayList<Task> mTaskHistory;
        private ArrayList<Task> mPlannedTasks;
        private ArrayList<Task> mRecommendedList;

        public RecommendTask(TodayFragment fragment, ArrayList<Task> tasks) {
            mFragment = fragment;
            mContext = fragment.getActivity();

            mTasksDb = TasksDatabase.getInstance(mContext);
            mTaskHistory = mTasksDb.getTaskHistory();
            mPlannedTasks = tasks;
        }

        @Override
        protected void onPreExecute() {
            mFragment.getProgressBar().setIndeterminate(true);
        }

        @Override
        protected Void doInBackground(Void... params) {
            mLocationClient = new LocationClient(mContext, this, this);
            mLocationClient.connect();
            mLastKnownLocation = null;

            // Get calculation time (since midnight).
            mTimeOfCalculation = Utilities.getTimeSinceMidnight(
                    Calendar.getInstance().getTimeInMillis());
            mRecommendationMap = new HashMap<String, Float>();
            mRecommendedTask = null;
            mRecommendedList = new ArrayList<Task>();

            // Put the currently active tasks first (should be on top of list).
            for (int i = mPlannedTasks.size() - 1; i >= 0; i--) {
                if (mPlannedTasks.get(i).isActive()) {
                    mRecommendedList.add(mPlannedTasks.get(i));
                    mPlannedTasks.remove(i);
                }
            }

            while (! mLocationClient.isConnected()) {}

            mLastKnownLocation = mLocationClient.getLastLocation();
            recommend(mTimeOfCalculation);
            updatePriorities();

            return null;
```

```
}

@Override
protected void onPostExecute(Void result) {
    mFragment.getProgressBar().setIndeterminate(false);
    Toast.makeText(mContext, mContext.getString(
            R.string.tasks_recommended), Toast.LENGTH_SHORT).show();
}

// Recommend a task for the time provided as parameter
private void recommend(long recommendationTime) {
    mRecommendedTask = null;
    mRecommendationMap.clear();
    timeOfDayRecommendation(recommendationTime);
    timeOfDayAndDayOfWeekRecommendation(recommendationTime);
    locationRecommendation();
    timeOfDayAndLocationRecommendation(recommendationTime);
    timeOfDayAndDayOfWeekAndLocationRecommendation(recommendationTime);

    float probability = 0;
    for (Entry<String, Float> entry : mRecommendationMap.entrySet()) {
        if (entry.getValue() > probability) {
            for (Task task : mPlannedTasks) {
                if (! task.isFinished() &&
                        task.getCategory().equals(entry.getKey())) {
                    mRecommendedTask = task;
                }
            }
        }
    }

    // A recommendation has been found.
    if (mRecommendedTask != null) {
        // Check if there is a task with a fixed time that may interfere
        //  with the task to be recommended.
        Task fixedTask = null;
        long fixedTaskStartTime = 0;
        for (Task task : mPlannedTasks) {
            if (! task.getFixedStart().isEmpty()) {
                long taskStartTime = Utilities.timeToMillis(
                        task.getFixedStart());

                // If the start time of the task is later than 'now' and if
                //  the task is sooner than a previously found task or if
                //  a task have not been found.
                if (taskStartTime - recommendationTime > 0 &&
                        ( taskStartTime < fixedTaskStartTime
                                || fixedTaskStartTime == 0 )) {
                    fixedTaskStartTime = taskStartTime;
                    fixedTask = task;
                }
            }
        }

        // If a task with a fixed start time has been found, we must
        //  account for this.
        long avgTimeSpent  = getAverageTimeSpentOnTask(mRecommendedTask);
        if (fixedTask != null) {
            long timeToFixedStart = fixedTaskStartTime - recommendationTime;

            // If able to find an average time and this time is less
            //  than the time until the fixed task is to be started.
            //  Recommend task and perform new recommendation with
            //  new time.
            if (avgTimeSpent > 0 && avgTimeSpent < timeToFixedStart) {
                mRecommendedList.add(mRecommendedTask);
                mPlannedTasks.remove(mRecommendedTask);
                mTimeOfCalculation += avgTimeSpent;
                recommend(mTimeOfCalculation);

                // If not able to find an average time, or there is not enough
                //  time before the fixed task.
            } else {
```

```java
                    mRecommendedList.add(fixedTask);
                    mPlannedTasks.remove(fixedTask);
                    if (! fixedTask.getFixedEnd().isEmpty()) {
                        long timeToNextTask = Utilities.timeToMillis(
                                fixedTask.getFixedEnd());
                        mTimeOfCalculation = timeToNextTask;
                    } else {
                        avgTimeSpent = getAverageTimeSpentOnTask(fixedTask);
                        mTimeOfCalculation += avgTimeSpent > 0 ?
                                avgTimeSpent : Constant.DEFAULT_AVERAGE_TIME;
                    }
                    recommend(mTimeOfCalculation);
                }
            } else {
                mRecommendedList.add(mRecommendedTask);
                mPlannedTasks.remove(mRecommendedTask);
                mTimeOfCalculation += avgTimeSpent > 0 ?
                        avgTimeSpent : Constant.DEFAULT_AVERAGE_TIME;
                recommend(mTimeOfCalculation);
            }
        }
    }

    /*
     * Calculate the type of task (category) that is completed most often
     *  at the time of day of the calculation.
     */
    private void timeOfDayRecommendation(long recommendationTime) {
        HashMap<String, Integer> categoryOccurrences =
                new HashMap<String, Integer>();

        long startTimeTask;
        long endTimeTask;
        for (Task task : mTaskHistory) {
            if (task.getTimeStarted() > 0) {
                // Get start and end time since midnight of the task.
                startTimeTask = Utilities.getTimeSinceMidnight(
                        task.getTimeStarted());
                endTimeTask = Utilities.getTimeSinceMidnight(
                        task.getTimeEnded());

                if (startTimeTask < recommendationTime &&
                        endTimeTask > recommendationTime) {
                    int occurrences = 1;
                    if (categoryOccurrences.containsKey(task.getCategory())) {
                        occurrences = categoryOccurrences
                                .get(task.getCategory()) + 1;
                    }
                    categoryOccurrences.put(task.getCategory(), occurrences);
                }
            }
        }
        addRecommendationsFromMap(categoryOccurrences);
    }

    private void timeOfDayAndDayOfWeekRecommendation(long recommendationTime) {
        HashMap<String, Integer> categoryOccurrences =
                new HashMap<String, Integer>();

        int dayNow = Calendar.getInstance().get(Calendar.DAY_OF_WEEK);
        long startTimeTask;
        long endTimeTask;
        for (Task task : mTaskHistory) {
            if (task.getTimeStarted() > 0) {
                // Get start and end time since midnight of the task.
                int dayTask = Utilities.getDayOfWeek(task.getTimeStarted());
                startTimeTask = Utilities.getTimeSinceMidnight(
                        task.getTimeStarted());
                endTimeTask = Utilities.getTimeSinceMidnight(
                        task.getTimeEnded());

                if (startTimeTask < recommendationTime &&
                        endTimeTask > recommendationTime &&
```

```
                    dayNow == dayTask) {
                int occurrences = 1;
                if (categoryOccurrences.containsKey(task.getCategory())) {
                    occurrences = categoryOccurrences
                            .get(task.getCategory()) + 1;
                }
                categoryOccurrences.put(task.getCategory(), occurrences);
            }
        }
    }
    addRecommendationsFromMap(categoryOccurrences);
}

private void locationRecommendation() {
    if (mLocationClient.isConnected()) {
        mLastKnownLocation = mLocationClient.getLastLocation();
    } else if (mLastKnownLocation == null) {
        return;
    }

    HashMap<String, Integer> categoryOccurrences =
            new HashMap<String, Integer>();

    ArrayList<TaskContext> taskContexts;
    for (Task task : mTaskHistory) {

        boolean hasSameLocation = false;
        taskContexts = mTasksDb.getContextByTaskId(task.getId(),
                ContextEntry.TYPE_LOCATION);

        for (TaskContext taskContext : taskContexts) {
            String[] latLng = taskContext.getDetails().split("\\s+");
            double latitude = Double.valueOf(latLng[0]);
            double longitude = Double.valueOf(latLng[1]);

            // Check if location where task was performed is the
            //  same as the current location.
            float[] result = new float[3];
            Location.distanceBetween(mLastKnownLocation.getLatitude(),
                    mLastKnownLocation.getLongitude(),
                    latitude, longitude, result);
            float distance = result[0];

            if (! hasSameLocation && distance <=
                    Constant.MAX_DISTANCE_LOCATION_RECOMMENDATION) {
                int occurrences = 1;
                if (categoryOccurrences.containsKey(task.getCategory())) {
                    occurrences = categoryOccurrences
                            .get(task.getCategory()) + 1;
                }
                categoryOccurrences.put(task.getCategory(), occurrences);
                hasSameLocation = true;
            }
        }
    }
    addRecommendationsFromMap(categoryOccurrences);
}

/*
 * Calculates and recommends the type of task (category) that is completed
 * most often at the time of the calculation and the current location.
 */
private void timeOfDayAndLocationRecommendation(long recommendationTime) {
    if (mLocationClient.isConnected()) {
        mLastKnownLocation = mLocationClient.getLastLocation();
    } else if (mLastKnownLocation == null) {
        return;
    }

    HashMap<String, Integer> categoryOccurrences =
            new HashMap<String, Integer>();

    long startTimeTask;
```

```java
        long endTimeTask;
        ArrayList<TaskContext> taskContexts;
        for (Task task : mTaskHistory) {

            boolean hasSameContext = false;
            taskContexts = mTasksDb.getContextByTaskId(task.getId(),
                    ContextEntry.TYPE_LOCATION);

            for (TaskContext taskContext : taskContexts) {
                String[] latLng = taskContext.getDetails().split("\\s+");
                double latitude = Double.valueOf(latLng[0]);
                double longitude = Double.valueOf(latLng[1]);

                // Check if location where task was performed is the
                //  same as the current location.
                float[] result = new float[3];
                Location.distanceBetween(mLastKnownLocation.getLatitude(),
                        mLastKnownLocation.getLongitude(),
                        latitude, longitude, result);
                float distance = result[0];

                if (! hasSameContext && distance <=
                        Constant.MAX_DISTANCE_LOCATION_RECOMMENDATION &&
                        task.getTimeStarted() > 0) {
                    // Get start and end time since midnight of the task.
                    startTimeTask = Utilities.getTimeSinceMidnight(
                            task.getTimeStarted());
                    endTimeTask = Utilities.getTimeSinceMidnight(
                            task.getTimeEnded());

                    if (startTimeTask < recommendationTime &&
                            endTimeTask > recommendationTime) {
                        int occurrences = 1;
                        if (categoryOccurrences.containsKey(task.getCategory())) {
                            occurrences = categoryOccurrences
                                    .get(task.getCategory()) + 1;
                        }
                        categoryOccurrences.put(task.getCategory(), occurrences);
                        hasSameContext = true;
                    }
                }
            }
        }
        addRecommendationsFromMap(categoryOccurrences);
    }

    private void timeOfDayAndDayOfWeekAndLocationRecommendation(
            long recommendationTime) {
        if (mLocationClient.isConnected()) {
            mLastKnownLocation = mLocationClient.getLastLocation();
        } else if (mLastKnownLocation == null) {
            return;
        }

        HashMap<String, Integer> categoryOccurrences = new HashMap<String, Integer>();

        int dayNow = Calendar.getInstance().get(Calendar.DAY_OF_WEEK);
        long startTimeTask;
        long endTimeTask;
        ArrayList<TaskContext> taskContexts;
        for (Task task : mTaskHistory) {

            boolean hasSameContext = false;
            taskContexts = mTasksDb.getContextByTaskId(task.getId(),
                    ContextEntry.TYPE_LOCATION);

            for (TaskContext taskContext : taskContexts) {
                String[] latLng = taskContext.getDetails().split("\\s+");
                double latitude = Double.valueOf(latLng[0]);
                double longitude = Double.valueOf(latLng[1]);

                // Check if location where task was performed is the
                //  same as the current location.
```

```
                float[] result = new float[3];
                Location.distanceBetween(mLastKnownLocation.getLatitude(),
                        mLastKnownLocation.getLongitude(),
                        latitude, longitude, result);
                float distance = result[0];

                if (! hasSameContext && distance <=
                        Constant.MAX_DISTANCE_LOCATION_RECOMMENDATION &&
                        task.getTimeStarted() > 0) {
                    // Get start and end time since midnight of the task.
                    int dayTask = Utilities.getDayOfWeek(task.getTimeStarted());
                    startTimeTask = Utilities.getTimeSinceMidnight(
                            task.getTimeStarted());
                    endTimeTask = Utilities.getTimeSinceMidnight(
                            task.getTimeEnded());

                    if (startTimeTask < recommendationTime &&
                            endTimeTask > recommendationTime &&
                            dayTask == dayNow) {
                        int occurrences = 1;
                        if (categoryOccurrences.containsKey(task.getCategory())) {
                            occurrences = categoryOccurrences
                                    .get(task.getCategory()) + 1;
                        }
                        categoryOccurrences.put(task.getCategory(), occurrences);
                        hasSameContext = true;
                    }
                }
            }
        }
        addRecommendationsFromMap(categoryOccurrences);
    }

    private void addRecommendationsFromMap(HashMap<String, Integer> map) {
        float total = 0;
        for (Entry<String, Integer> entry : map.entrySet()) {
            total += entry.getValue();
        }

        float probability;
        for (Entry<String, Integer> entry : map.entrySet()) {
            probability = (float) entry.getValue() / total;
            if (! mRecommendationMap.containsKey(entry.getKey()) ||
                    mRecommendationMap.get(entry.getKey()) < probability) {
                mRecommendationMap.put(entry.getKey(), probability);
            }
        }
    }

    private long getAverageTimeSpentOnTask(Task task) {
        long totalTimeSpent = 0;
        int numberOfTasks = 0;
        for (Task t : mTaskHistory) {
            if (t.getCategory().equals(task.getCategory()) &&
                    t.getTimeStarted() > 0) {
                numberOfTasks += 1;
                totalTimeSpent += task.getTimeSpent();
            }
        }
        return numberOfTasks > 0 ? totalTimeSpent / numberOfTasks : -1;
    }

    private void updatePriorities() {
        for (int i = 0; i < mRecommendedList.size(); i++) {
            mRecommendedList.get(i).setPriority(i+1);
            new DatabaseUtilities.UpdateTask(
                    mContext, mRecommendedList.get(i)).execute();
        }
        for (int i = 0; i < mPlannedTasks.size(); i++) {
            mPlannedTasks.get(i).setPriority(mRecommendedList.size() + i + 1);
            new DatabaseUtilities.UpdateTask(
                    mContext, mPlannedTasks.get(i)).execute();
        }
```

```java
        }

        @Override
        public void onConnectionFailed(ConnectionResult connectionResult) {
            cancel(true);
        }

        @Override
        public void onConnected(Bundle bundle) {}

        @Override
        public void onDisconnected() {
            mLocationClient = null;
            cancel(true);
        }
    }

    public static void recommend(TodayFragment fragment, ArrayList<Task> tasks) {
        new RecommendTask(fragment, tasks).execute();
    }
}
```