Using mobile sensing to detect indicators of alcohol intoxication

Boris Das

July 21, 2018

Abstract

Alcohol abuse is being one of the biggest causes of preventable deaths, therefore development of a tool that can help prevent alcohol abuse and aid in reducing alcohol abuse, can be beneficial to the global heath of the worlds population. In this thesis, a project is described that is aimed at collecting data, that could be indicators of alcohol intoxication. The project consists of an application that can be installed on a smartphone, which when used will unobtrusively collect data. The project makes extensive use of the data that is typed on the smartphone keyboard. It is designed to respect the user's privacy, and only extracts meta data from typed characters. Besides typing sensors, acceleration is also tracked to see if this can indicate alcohol intoxication.

Contents

Chapter 1

Introduction

This chapter briefly tries to state the problem and scope of this project. The project will try to identify the use and misuse of alcohol in order to either prevent long or short term problems. The findings in this project should be able to be used in research towards alcohol use disorder and to support projects that would need a way of non-intrusively identifying if a user is intoxicated. Since mobile phones are more and more accessible we can try to use these as a tool to identify patterns and indications of alcohol use. The research in this thesis tries to find these patterns of phone use to help identifying the use and misuse of alcohol. When we successfully do we can use this data to help people with an alcohol use disorder (AUD) to guide them to a more healthy lifestyle.

1.1 Background

Alcohol consumption has been identified as an important risk factor for illness disability and mortality. Even though alcohol has been proved to help with cardiovascular diseases in lower amounts of use, most of the burden in getting diseases seems to stem from regular heavier drinking. This is defined as more than 40 grams of pure alcohol for men and 20 gram for a day for women. In addition to regular drinking, irregular patterns of heavy binge drinking defined as more than 60 grams of pure alcohol are also significantly associated with the burden of the diseases associated with alcohol use[10]. Overall the following diseases and injuries are impacted by the use and misuse of alcohol:

- Infectious Disease
- Cancer
- Diabetes

- Neuropsychiatric disease
- Cardiovascular disease
- Liver and pancreas disease
- Unintentional and intentional injury

About 35% of the United States population abstains from the use of alcohol, about 60% are occasional to moderate drinkers, and about 5 to 7 % are diagnosable with alcohol abuse or dependence. Out of 16 million users that meet the diagnostic criteria for alcohol abuse or dependence, only 1.5 million seek and receive treatment[1]. Alcohol abuse is a big concern in the world, and in America alcohol misuse is the third biggest cause of preventable death causes [7]. This is of course a big health concern, which with some help could be easily prevented. Lancet states that around 2 billion people worldwide consume alcoholic beverages, and that over 76 million people have an alcohol use disorder (AUD)[11]. WHO estimated that about 2.3 million people die a premature death worldwide, and is responsible for 4.4% of the global burden of disease[11]. People with an alcohol use disorder can often prevent getting alcohol related diseases when taking the necessary precautions when drinking[12].

Besides the physical diseases and problems, there is also the mental state that can be influenced by alcohol use. In a study that combined several other studies, strong relationships between alcohol abuse and depression could be concluded. This relationship was either the alcohol use triggering the depression, or the alcohol was used as a "self-medication" kind of role [2]. The direction of which way the two are related is not always clear, but one could conclude alcohol use disorder could be used as an indicator for a depressive state. The problem with alcohol and the abuse is that often people do not dare to admit the problem, or do not want to do anything about it.

1.2 Smart phones as a tool

With the rise of access to smart phones and other technologies, new methodologies and approaches to problems can be developed. According to Google, Android had about 2 billion active users in 2017 and according to the CIA [1] about 7.1 billion active cellular phones exist in the world, and in nearly a third of the countries, the number of cell phones in use is greater than the

[1]https://www.cia.gov/library/publications/the-world-factbook/geos/xx.html

number of people living in those countries [4]. Ways of researching a problem that previously might have had a hard time to gather enough data, could now be approached in a way where a mobile phone can become a research tool. Since a lot of people use a smart phone on a daily basis, data could continuously be gathered in a non-intrusive way. This way participating in a research is less time consuming for the participants and the research is more likely to have more participants and data. Many tests will have the user answer prompts to check if a certain state of mind is present. But often in such tests a user could not always be answering the questions true. If after the research phase these prompts can be eliminated because sufficient data has been collected and analyzed, we can develop a more non-intrusive way to spot behavioral changes ahead of time. If we succeed in developing such a tool we can better understand what role a mobile phone is playing when it comes to the use and abuse of alcohol.

1.3 Motivation

Seeing the effects and impact of alcohol use and the availability lead me to believe there is a gap that can be filled by combining the technology with the problem of alcohol use disorder. Since it is the third biggest cause of most preventable diseases, I see there is a huge impact to be made in solving even the slightest part of the problem.

1.4 Problem Statement

Since AUDs are a huge factor when it comes to global disease around the world it would be desirable to identify the abuse before it actually becomes a problem. If we succeed in doing so we can get treatment in the places they are needed and in this way we can prevent alcohol use from becoming a health risk and in this way increase the overall worldwide health. While some people do get treatment for alcohol abuse, rehabilitation facilities do not always have the right checkup methods to see if a patient is really still sober. Sometimes people will lie about the relapse because the feel ashamed.

1.5 Research Questions

To investigate if we can collect any data that is relevant to finding out if someone is intoxicated or getting intoxicated the following research question has been defined.

Can we identify in a non-intrusive way if a person is intoxicated, with the use of smart phone technology?

The important part is that we want to make it as non-intrusive as possible as we don't want it to feel like a thing that take effort to do since this might harm the effectiveness of the research and overall goal. By collecting the data of a user we hope to be able to spot changes in data, of which we can conclude alcohol intoxication. This will mainly be focused on the use of typing messages and how many type corrections are done in the process. We expect users that are intoxicated to have different typing processes, than those that are not. To be able to use the smart phone technology as much as possible, we want to see if other sensor data might also give an indication of alcohol intoxication.

Supporting questions To backup the main research question one of the subquestions is:

Which tools and sensors on a mobile phone can be used to indicate alcohol use?

The purpose of this question is to focus not only on keystroke analysis, but also explore if other data can indicate the use of alcohol. Since most phones nowadays have a gps and accelerometer this data can also be used to analyze. During the research we might think of other methods we can exploit to further back up data that has already been gathered.

To conclude the research we want to try to formulate a metric that will give a quick indication whether a user is index intoxicated or not. Therefore we formulate another question as following:

Can we formulate a metric to indicate a certain level of intoxication?

This can either be a certain scale that either points to an exact level of alcohol, or to put a user in a certain "range" of likeliness to be intoxicated. The first one would probably be most unlikely because most people seems to react quite different at certain level of intoxication.

1.6 Structure of the Report

In the next few chapters the process of developing an application that can help identify intoxication will be described. Chapter 2 will clarify which approaches have already been done in a scientific field, and how we can combine several techniques to come to an application that can be used in the field of alcohol use detection. Chapter 3 describes how we designed our approach and how we implement the actual application. The software architecture shall be discussed and explained. In chapter 4 we try to formulate a test that will be conducted to get the first training data. When we have conducted these tests we will review the data and discuss it per test and see where significant data has been found. We will also test the usability of the application in the for of a questionnaire. In chapter 5 we discuss our final findings and try to formulate a conclusion based on the several test that have been done. We hope to find a certain metric that will tell us whether a user is being or getting intoxicated so we can start using the methods to actually help the user. The final chapter shall discuss how this application could be used to help or could be improved to better suit specific needs. After reading this thesis one can expect to have read about an application that can successfully collect data that can indicate alcohol intoxication, in a non-intrusive way, by using smartphone technology.

Chapter 2

State of the art

In this section an overview of several projects that have tried a similar approach to other or similar problems. The projects discussed shall mainly include projects that have also used mobile phones as a tool to identify states or help a user with similar problems as this project encounters. Not all cases are related to alcohol use, but can be used as an inspiration source for our research.

2.1 Tapsense

In the tapsense project researchers have conducted a study where they developed an Android application that would collect data about the user's texts input. The goal was to be able to identify at least 4 emotional states. The 4 states they identified were relaxed, happy, stressed, and sad. In this project the relaxed state seemed to be reported the most. The projected used an own implementation of the android keyboard in order to track the data. After certain amounts of typing and switching applications, the users would get a prompt to ask how a user was feeling at that moment. Short text messages were filtered out to make sure the actual typing data was relevant enough to be analyzed. Then by using machine learning they were able to connect certain ways of typing with emotions. With 22 participants to test and collect data they managed to obtain an average accuracy of 84% in classification. The precision of predicting a relaxed state was close to 80% and the other 3 states could be predicted with 60% and above. [6] This project seems promising when it comes to detecting emotional states with the use of a smart phone. Other papers have also claimed to be successful with identifying emotional states and stress levels [9, 8].

2.2 Deployment of keystroke analysis on a smart phone

Another paper [3] suggests that keystroke analysis can be used for user identification on a phone. This means that besides having a pin to unlock a phone, a continuous process could identify and authenticate a user based on typing. They suggest a possibility in identifying user by these biometrics. It is something the person is and not something that the person knows (e.g. a pincode). In their approach they used a method where a user would have to enter a password 20 times and then authenticate 10 times. Of course for a lot of user this would be to much repetitiveness. And while this project doesn't yet succeed that well in the implementation yet, it does clarify the importance of non-intrusiveness. Also in the paper they state that computational power was one of the biggest struggles since phone were not too capable at that time. With modern day technology we hope to further support the use of keystroke analysis.

2.3 Mobile phone based drunk driving detection

In a paper called mobile phone based drunk driving detection [5] researchers approached being intoxicated in another way than using key logging. While this project specifies being intoxicated while driving, this study can prove to be effective on a smaller scale as well. They researched the accelerations of a phone in a car to measure the jittering of speed, and being unable to maintain a steady speed, and also the ability to drive normal in a sense that the driver doesn't swerve. Which the latter they called abnormal curvilinear behavior. They state that with the method they used the never had any false negatives on both criteria of driving. And the curvilinear criteria they had only 0.49% false positives, and with the speed maintenance a percentage of 2.39%. Also some of the error they suggest, might come from turns that sway the phone to much, and thus bump it a against something. The paper suggest that using GPS in combination with this accelerometer technology could further support the errors in confirming whether or not is was a corner, in which the system could more likely expect false data.

2.4 Summary

Many of these approaches use a smartphone or other devices that can track a users way of interacting and identifying a certain state or uniqueness of interacting. In the tapsense project we see that they used prompts alongside data to learn the application how to interpret certain data. This seems like a very good lead into what we can and should do in our area of research. If emotional states can be concluded from typing data it can probably also be done with others states of mind. Depressive states have also be linked to alcohol use and vice versa [2]. We could use this to help analyzing the data.

If we can use an approach similar to the drunk driving project, but on a smaller scale, we can quite possibly back our other data findings up with additional data to support them. Assumptions for now can be made that when walking while intoxicated, some accelerating data is going to change in contrast to sober movements.

While each project discussed uses a different approach the combined data might help us to declare a more detailed metric to define intoxication levels. Since a phone hosts so many different ways of measuring data about a user we should look into the multiple options and combine these. If we look at the numbers the projects claim to have it seems that a combination of the approaches should give us a pretty clear indication of the intoxication levels of a user.

Chapter 3

Method

The goal of this project is to gather information about a user, in a way that is as unobtrusive as possible, in order find indicators that might suggest intoxication. In this project we specifically focus on the identification of alcohol intoxication. This includes the collection of the user's phone usage and gathering as much data as possible, within the regulations. What is relevant for identifying the intoxication can be very elusive, so the data gathered would be kept as broad as possible. This means nothing should be considered irrelevant until research might suggest otherwise. Doing so allows for a more accurate model for deciding whether a user is intoxicated. When deciding, based on the data that is available, it should be able to at least detect a slight difference in data, but also there should be caution in claiming the user to be intoxicated. The paper about TapSense suggests that a technology approach like theirs, might as well be effective in another field of cognitive problems [6]. Therefore the base of our approach is in the most fundamental part inspired on it. Since most people nowadays own a smartphone, this technology shall be used to track data of the user. While the TapSense was mainly focused on typing data, this project is also considering other factors that could possibly give a more detailed insight in what the user might do in different states of mind.

3.1 Design

This section will describe the requirements of the system and the design choices that were made.

3.1.1 Requirements

The project goal is to gather as much information as possible in order to come to a conclusion which factors are relevant for determining. Since there are a lot of factors and sensors are available, a MoSCoW analysis was done to define what the project has to do, and what it should be considered out of scope. In the MoSCoW analysis the following aspects are considered. *Must have*: The must haves are the minimum components and things the project should have or do. Without these aspects and components the project could be considered undeliverable. *Should have*: These are the components that would be described as desirable but the application should be able to achieve its goal. These components will be left out when the project runs out of time and will not be able to implement these. *Could have*: The requirements that are labeled as could have, are desirable for the project but are merely to improve the project in the sense of user experience or usability. The *Won't haves*: These are the components that are valued as the least important and will for this project be left out of the scope. These can later, or in another iteration still be implemented to further supplement the previously developed project.

Must have: This project focuses on the use of mobile equipment for data tracking. Therefore the application that is developed must run on at the most common devices. This includes the latest smartphones and smart watches. The application must at least be able to analyze the text a user is typing, but must not be too obtrusive while doing so. This means that the data collection should be done in the background while the user is able to use the phone in a way he or she wants to. While this application is mainly aimed at a research perspective, the data that is collected should be as anonymous as possible, and must not disclose too much information about the user. This means that, while typing data is gathered, the actual characters that are entered should not be visible to the researchers, as this will probably make a user feel uncomfortable. When the data has been collected on the phone locally, the data is not easily accessible to researchers. Therefore the data must be synchronized to a server, on which a researcher can access the data that is needed for analysis. The data should be in a clear and easy structure and will not need much explanation for it to be understood by a researcher. Once the application is running on a system, no additional steps should be taken in order for the application to keep doing its job.

Should have: The application in this project should have a solid frame of collecting data. This means that while a user might type more at one moment, he or she might type less in another. Therefore a user should have a clear incentive to make data available to be collected. This could be a simple question to answer or just a random text box every once in a while. Besides typing data the application should also include other data. This data can be accelerometer data, GPS data, and screen touch data. When it's possible to link this to the typing data, that data is suspected to help identify intoxication.

Could have: The system could have the possibility to also check word correctness. This would mean that if a word is spelled wrong, it could detect the fact that it was. Also when a user taps a word suggestion, it could make sense to see if a user typed a word using a suggested word. Also a feature like a game could eventually be implemented, so the user has more incentive to generate more data to be analyzed.

Will not have: While this project is aimed to be a setup towards machine learning applications, this project will not implement a way to do this. The project will collect the data that could make a machine learn, but it will not actually do it. The application will not include voice commands and speech recognition as it feels like that would hurt privacy way more than the data can ever satisfy, even if it perfectly would do its intended job.

3.2 Design choices

When designing the project the factors mentioned in section 3.1.1 were taken in consideration. The application will track a user's typing habits and keep data that could be relevant for researchers in developing a tool that can potentially identify alcohol intoxication. The user will only have to give a rating about how relevant the data is to the subject that will be studied. In this case that would be a question asking how intoxicated the user felt when the data was recorded. The data collection will be fully unobtrusive and will not necessarily involve any further user interaction or configuration on the researchers' side. This will also ensure data consistency because data will not be influenced by bad configurations. The tracked data will for now exist of

- Keystroke logging and analysis
- Acceleration and rotation logging
- Application sessions and connected data

The next section will describe in more detail what data will be used and how it will be used. A way to access data in a more usable way will be described in the sessions section 3.2.3.

3.2.1 Keystroke logging

When logging a users keystrokes there are different ways to approach the data tracking and to evaluate it. For example when a user types data into a textbox using the keyboard this is done by pressing several keys on the keyboard. The meaning of the individual characters typed into the field by themselves might hold very little data, besides the time they were typed and the specific character it was. When analysis on a character per character basis this will most likely fail to produce any useful data. Therefore this approach is considered useless for this project.

When we start looking at two characters at a time we can already extract more data since there will be a difference between the two of them. Where before we could only consider the character and its time, we can now compare the two times of typing and calculate how much time has elapsed between the two of them. This will define a type speed between the two of them, which could be a factor that is influenced by alcohol intoxication. Also by comparing the previous and current character a conclusion can be made of what character is entered. This is useful in the case of having pressed a backspace or a space, since a backspace is not an actual character itself but a modification to the text that has already been typed. Because the data framework that will be used 3.3.1, does not support direct key logging, we will have to use this approach for finding out if a backspace has been used. In this project we expect that based on the TapSense results the backspaces are a strong indicator of mood [6] and will probably also be affected by alcohol intoxication. This approach will be the main starting point for our analysis.

When comparing two characters more data becomes available to collect, but that still isn't covering all the data that can be collected from typing data. When we start collecting data over multiple characters we can also see the data that can be connected to words. When analyzing words, we can now also try to provide more context as to what is being typed how. This means we can go for a word per word analysis, which can include the typing speed per word and the correctness of the word. For example when typing a word, a person can correct the word by using a backspace and correcting it. This word will then be marked as a corrected/fixed word. If we can get access to the user's suggested words that are stored in the user's phone, we can check if the word was actually spelled correctly.

The context in which these words are used can provide data about the sentences and stories a user might have typed. This would be a sentence analysis which is most likely more privacy invading than the data needed has to be. Analysis of context like this will not be part of the scope. We probably also do not need it, because a per word analysis should already give a lot of data that is relevant.

What is a word

When defining what a word is based on characters it should be considered what for this project we define as a word. And in this project we consider a word a series of characters encapsulated by 2 spaces or line end or starts. This is because several cases will arise when looking at 2 character at a time and when a word is finished. In the case there has only be typed one character we will not consider this a word because very few data can be extracted from this. The possible scenarios for a word in our way of tracking key data are the following

- A new word in an empty text box
- A word after another word
- A word that is finished because the application was closed
- A deleted word

- A fixed word

New word: A new word is a case where no previous data was present in a text box. This means the first character that was entered should in almost every case be a word, but does not yet say anything about the typing data. The word is formed when a space is detected and no other space was recorded before that space.

Another word: When the first word has been recorded as being a word, other words can still be entered. These words will be identified in the same way as the new words. This word will again only be a word after the previous character was not a space.

Application closed word: When a user is typing a word and sends a message for example, it will not yet be registered as a word, because no space would have been detected. Therefore when the user closes the application it is currently typing in, the last registered key data should be saying that it was a word ending. Unless the last character typed was a space.

Deleted word: A deleted word should also be tracked, otherwise we would get a much bigger word count than an actual message might contain. Also the amount of deleted words can be a factor in determining when a user was intoxicated. Therefore when a word is deleted the currently inserted backspace character gets flagged as -1 word. This is because it is complicated to change the character that had previously made a word, to be flagged as not a word anymore. When the words for a session get calculated at the end of a typing session will still add up to the actual amount of words that were typed in the session. Also by checking which characters were flagged as -1 we can see how many words were deleted in a session. The conditions for a word to be flagged as deleted are when: the current entered character is flagged as being a backspace, has the value of space, and the previously entered character was not a space. Or when the current character is flagged as a backspace and the current text that was found in a text box is 0, and the previous text was larger than zero but not a space. This sounds a bit vague for now but will become clear when reading the code implementation. In which we look at what data is tracked and how it is saved.

Fixed word: A fixed word has the same conditions as a deleted word. The check for a fixed word is different from a deleted word in the sense that it does not have to be checked if the current text box is empty, because that would mean its already a deleted word, and that the previously entered character has to be a non space character instead of being a space.

Smiley: A smiley in this project also considered as a word. When a word is formed the application has to check if the word was in a predefined list of smileys. Each smiley is put in a category that could related to a certain mood. For this project the smileys have been divided into 5 categories, which are:

1. Happy smileys: ":)", ":-)", "=)"
2. Super happy smileys: ":D", ":-D", "=D"
3. Silly smileys: ":P", ":-P", "=P", "xD"
4. Sad smileys: ":(", ":-(", "=(", ":'(", ":'-(", "='("
5. Love smileys: "<3"

Summary: While the process of making words out of single characters is a relatively complicated process. It will likely be relevant, as the data of how many words are typed, fixed, and deleted gives more context to why a backspace or a space was used.

3.2.2 Acceleration and Rotation logging

In order to see if movements of the phone are relevant in defining if a user is intoxicated, acceleration and rotation of the phone are also used in the data collection. On a mobile device usually several sensors are available to track the phone's movements, and thus how a user is using his or her phone. The collection of data will be done on the background and will not impact the user at all. Although the application is constantly tracking the data, it is quite hard to know what happened when changes in data are present, therefore we for now only use the acceleration/rotation data that is collected during typing. This data will be coupled and simplified in a session. When we have defined what a session is we will know what is happening during the data changes, and we can then rate the data to a certain level of intoxication.

3.2.3 Sessions

In order for the application and researchers to make sense of the amount of data that is accessible, we will define a frame in which data is relevant to a certain level of intoxication. This frame is the data that is collected in each typing session. The sessions we will create, will be the main connecting component between a time frame and the collected data in it. Because the collected data individually is not connected to anything related to intoxication, the session will be rated by the user, based on how intoxicated he or she feels at that moment. This might be subjective for a user, but against the user's own 0 rated data, the changes can still be detected. This means that we are looking for changes in data between sessions that were rated 0, and sessions that had a higher rating.

Session definition: The sessions we define for our application are, when a user switches from one application to another, in which the user has also provided the application with enough typing data. This is to prevent the session data clogging with data that probably will not help the research, and to prevent the user from having to rate too many sessions.

Session data: The data the session holds, will be the typing data and acceleration/rotation data. The amount of words and the individual character data will be used to save data about average typing speed, and average typing speed per word. The session will also be tracking in what application the typing and movement will be done, and how long the application/session has been opened. Smiley count, and a smiley string will also be tracked. The smiley string will describe what smileys are present in a session, and how many. This will be in the format:

- *{smileytype}:{smileytypeamount};*

After each semicolon another part will be added in the same format, for each smiley type present.

Rating a session: When a new session is created the user will get a prompt with a slider, to give the session a rating about his or her intoxication level. When a session has not been rated it will either be rated based the previous rating the user gave, or with a flag indicating that the session has not been rated.

3.3 Implementation

For developing the application, the Android [1] platform was used. Android is a popular operating system primarily developed to run on mobile devices

[1]see: https://www.android.com

including smartphones and tablets. Applications that are developed for android make use of the Android Software Development Kit or short SDK. Nowadays at least two languages can be used that are supported by Google who is the main developer of Android. A newer language Kotlin [2] is supported natively, but for this project has been decided on the use of Java. Android Studio 3.1.2 has been used to develop the application. The collection of sensor data on the smartphone and synchronizing the collected data to a server/dashboard will be done using AWARE[3].

3.3.1 The AWARE framework

AWARE framework is a framework that makes sensors on a phone easy to use, and to easily collect the data from them. AWARE is available to everyone under the Apache Software License 2.0. The framework uses a client/server approach in which researchers can run their own data servers, and users can then join a study with the AWARE application. The framework can log data of several sensors and components which include

- Accelerometers
- Application data
- Keyboard data
- Rotation data
- Experience Sampling Methods (ESM)

Only the sensors and components used in this project are listed in this list but it has many more to be used.

The framework is available on android and on iOS but has limitations on the iOS systems. This is one of the main reason the application has been developed for an Android system. Especially because the keyboard sensor is only available for Android. The AWARE framework can either be used as an application that runs plug-ins or be used as a foundation for an standalone application. In this project is chosen to develop a plug-in instead of a standalone application. The two reasons for this are the simplicity of how a plug-in can be installed, and because the user has more control over what happens on his or her phone.

[2]see: https://kotlinlang.org/

[3]see: http://www.awareframework.com/

When the user joins a study, the data that is collected on the phone is periodically synchronized to a server environment. On this server environment researchers can easily access the collected data and make visualizations of the data with charts and diagrams. The way the AWARE framework works is illustrated in figure 3.3.1. To summarize the workings of the framework, these are the steps of the AWARE process. Step 1) In the first step researchers will create a new study on the server dashboard, where they can add the sensors and custom plug-ins they would like to use for their study. They will have to give a title and description for the study, so the users will be able see what kind of study they are participating in. Step 2) The next step is for the users to install the AWARE core application and join a study by scanning a QR code. When joining a study the AWARE application will ask the user to install the plug-ins needed for the study. When installed the plug-in will ask for the permissions it needs in order for the plug-in to be able to do it's job. When this is done the user will have to tap the "SIGN UP" button, that will now be enabled. The user will now be joined in the study. Step 3) As soon as the user has joined the study data collection has started and the data will be stored on the phone. Step 4) The AWARE application will periodically synchronize the collected data to the corresponding study on the server where the data will be stored in a MySQL database. Step 5) The researchers can now use the data for analysis and research. The core functionality of AWARE allows researchers to send broadcasts to the application on the users' phones. These can either trigger a force sync, or other messages. It is also possible to send ESM's or configurations to the AWARE application. The AWARE application on the users phone can then react to these messages, in order to execute parts of code, or sending back answers to ESMs. The ESMs will be further discussed in the ESM section 3.3.1.

The AWARE application

To give some more context this section will explain a bit more about what the actual application does and looks like. And what the relevant components to our project are doing.

The core application screens The core application that will be install on the users' phone will has several screens that will interface the application. The first screen the user will see is the sensor screen which is show in figure 3.2. This screen contains the core sensors that AWARE has available for data collection. In this screen, the sensors can be selected, which will then open the corresponding sensor's settings. In the settings screen you will be given the options for the sensor.

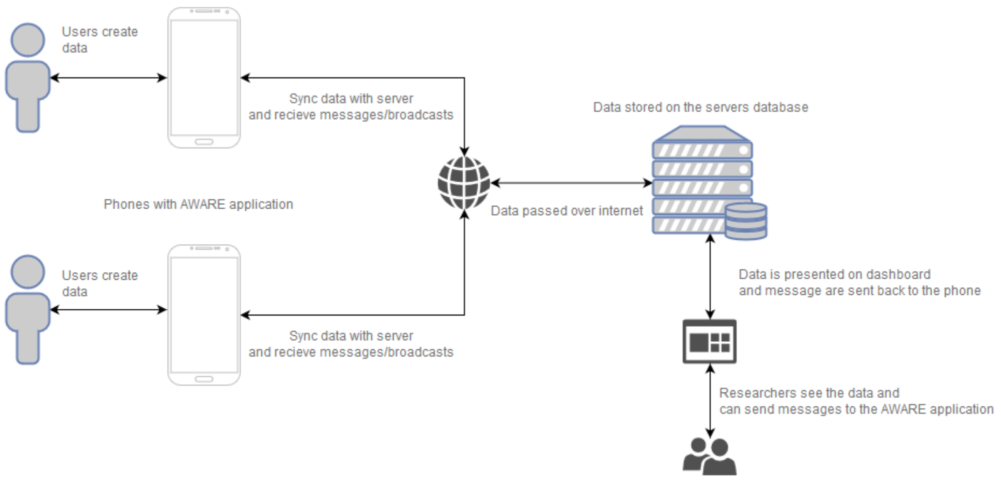
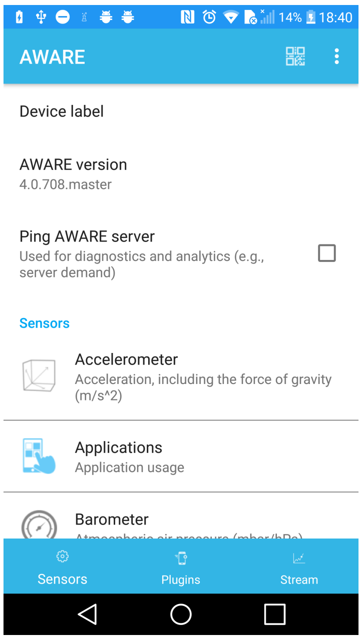

Users create data
Sync data with server and recieve messages/broadcasts
Data stored on the servers database
Phones with AWARE application
Data passed over internet
Users create data
Data is presented on dashboard and message are sent back to the phone
Sync data with server and recieve messages/broadcasts
Researchers see the data and can send messages to the AWARE application

Figure 3.1: AWARE framework workflow

In the setting screen shown in figure 3.3 the settings for the application screen are show. This sensor also hosts the keyboard setting which we need to develop our plug-in.


AWARE

Device label

AWARE version
4.0.708.master

Ping AWARE server
Used for diagnostics and analytics (e.g., server demand) ☐

Sensors

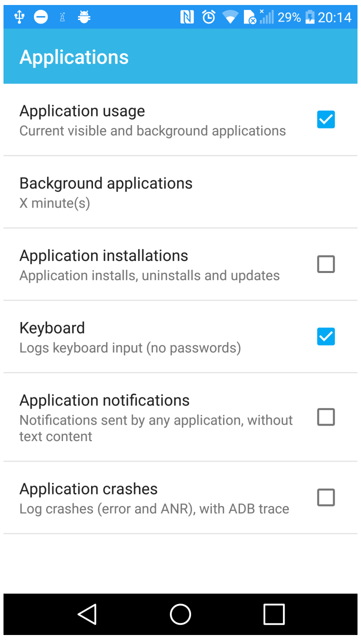
Accelerometer
Acceleration, including the force of gravity (m/s^2)

Applications
Application usage

Barometer

Sensors Plugins Stream

Figure 3.2: Sensor screen


Applications

Application usage
Current visible and background applications ☑

Background applications
X minute(s)

Application installations
Application installs, uninstalls and updates ☐

Keyboard
Logs keyboard input (no passwords) ☑

Application notifications
Notifications sent by any application, without text content ☐

Application crashes
Log crashes (error and ANR), with ADB trace ☐

Figure 3.3: Sensor settings

In the top navigation of the AWARE application there is a combination of icons possible, which are shown in figure 3.4. The options marked as 1 and 2 are not available when no study has been joined yet. When pressing the 3rd option android will open AWARE's QR scanning screen which is shown in figure 3.5. When scanning a QR code that is available on the AWARE dashboard, the application will open the "join study" screen of the corresponding study, of which an example is shown in figure 3.6. This screen will show a list of all additional plug-ins needed to join the study. These will then ask for the needed permissions in screen 3.7, after which the user will be able to press the "SIGN UP" button on the bottom of the screen, to join the study.

When the study has been joined action buttons 1 and 2 will be shown in the top bar of the AWARE application. Option 1 will bring the user back to the join study screen again, which will now show the option to leave the study when the user wants to. When conducting a study participants should feel they are in control of their privacy. Option 2 forces the data that has locally been collected on the phone, to be synchronized with the server. This can be useful when data from a certain phone needs to be updated to the server. On the dashboard a researcher can also force the data to be synchronized be sending a broadcast message to the system.

The final screen discussed in this section is the stream tab. The stream tab hosts an overview of all added plug-ins that are currently running. In this case, it will be a view in which our plug-in will display some data to the user, so he or she can check if data is collected or not. The stream screen is shown in figure 3.8. To provide some more information about several components that are used in the plug-in, a description of them will be given in the next few sections.

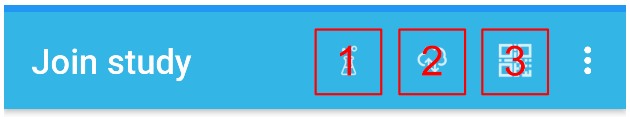


Figure 3.4: Available icons

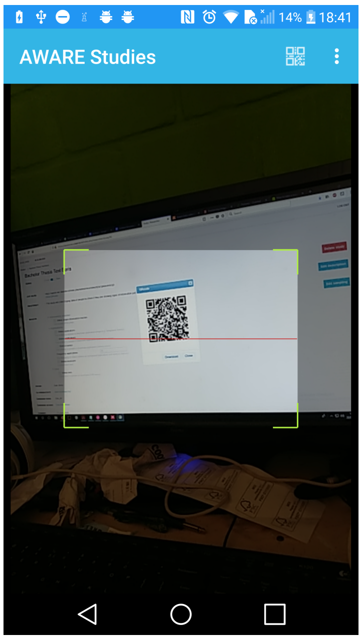
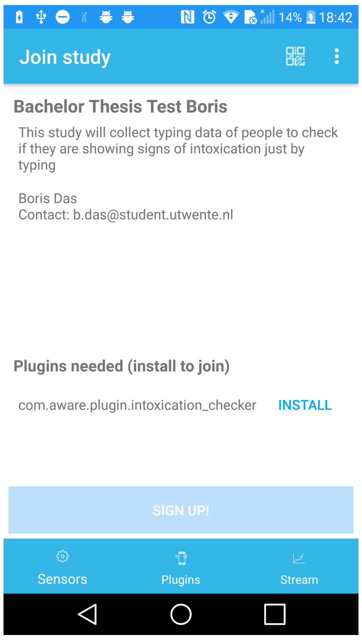


Figure 3.5: Joining a study


14% 18:42

Join study

Bachelor Thesis Test Boris

This study will collect typing data of people to check if they are showing signs of intoxication just by typing

Boris Das
Contact: b.das@student.utwente.nl

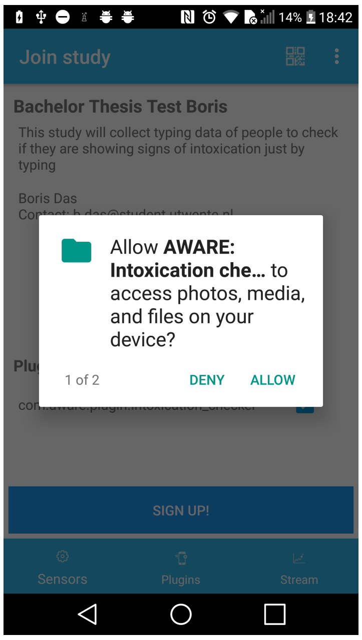
Plugins needed (install to join)

com.aware.plugin.intoxication_checker INSTALL

SIGN UP!

Sensors Plugins Stream

Figure 3.6: Install plug-ins


14% 18:42

Join study

Bachelor Thesis Test Boris

This study will collect typing data of people to check if they are showing signs of intoxication just by typing

Boris Das

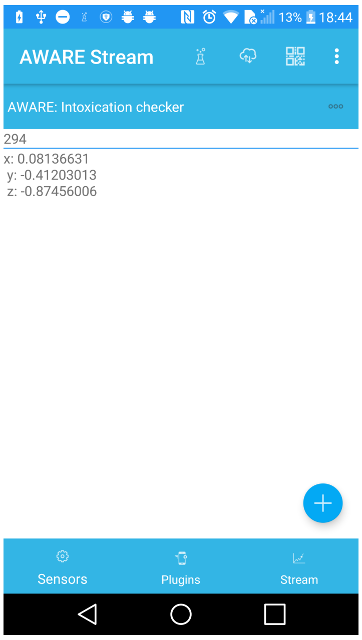
Allow **AWARE: Intoxication che...** to access photos, media, and files on your device?

1 of 2 DENY ALLOW

SIGN UP!

Sensors Plugins Stream

Figure 3.7: Permission check


13% 18:44

AWARE Stream

AWARE: Intoxication checker

294
x: 0.08136631
y: -0.41203013
z: -0.87456006

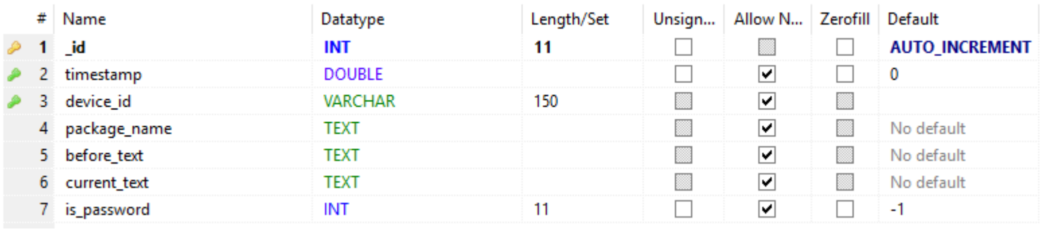
Sensors Plugins Stream

Figure 3.8: Context card

Keyboard sensor The keyboard sensor is one of the main components that is used in the development of this project. When the sensor is turned on, it collects data about what a users have been typing into text boxes. A user needs to have given permission for this to the android system, and has to enable androids accessibility services for the AWARE application. This is because with standard settings, android does not allow other applications to access what is typed into these boxes. However android made it possible for developers and users to make text in these boxes bigger, and develop applications that can help a user to make reading easier, or other applications that can help a user to make certain features more accessible for them. Since this is implemented this way by android, AWARE can use this data to track the keyboard data. Passwords will not be shown by the android system, which also helps us make a more privacy safe application. The way data is stored on the phone by AWARE is shown in figure 3.9. Some example data is shown in figure 3.10. Note that the current text column has the current text surrounded by brackets. Later on in developing the application we have to take care of how the data is saved. Whenever a new entry is added to the data table on the phone, AWARE sends out a broadcast message with the flag

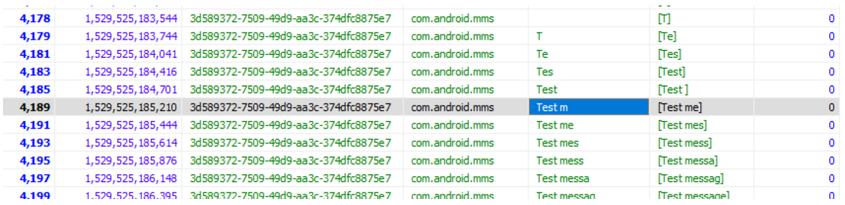
```
Keyboard.ACTION_AWARE_KEYBOARD.
```

We can then listen to this broadcast message, to see that new data was added to the keyboard data. More about how we use this data to get the relevant data will be discussed in the code implementation section 3.4



#	Name	Datatype	Length/Set	Unsign...	Allow N...	Zerofill	Default
1	_id	INT	11	☐	▨	☐	AUTO_INCREMENT
2	timestamp	DOUBLE		☐	✓	☐	0
3	device_id	VARCHAR	150	▨	✓	▨	
4	package_name	TEXT		▨	✓	▨	No default
5	before_text	TEXT		▨	✓	▨	No default
6	current_text	TEXT		▨	✓	▨	No default
7	is_password	INT	11	☐	✓	☐	-1

Figure 3.9: SQL table for keyboard data



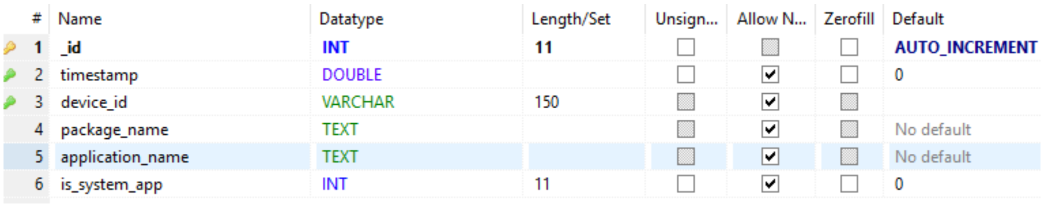
4,178	1,529,525,183,544	3d589372-7509-49d9-aa3c-374dfc8875e7	com.android.mms		[T]	0
4,179	1,529,525,183,744	3d589372-7509-49d9-aa3c-374dfc8875e7	com.android.mms	T	[Te]	0
4,181	1,529,525,184,041	3d589372-7509-49d9-aa3c-374dfc8875e7	com.android.mms	Te	[Tes]	0
4,183	1,529,525,184,416	3d589372-7509-49d9-aa3c-374dfc8875e7	com.android.mms	Tes	[Test]	0
4,185	1,529,525,184,701	3d589372-7509-49d9-aa3c-374dfc8875e7	com.android.mms	Test	[Test]	0
4,189	1,529,525,185,210	3d589372-7509-49d9-aa3c-374dfc8875e7	com.android.mms	Test m	[Test me]	0
4,191	1,529,525,185,444	3d589372-7509-49d9-aa3c-374dfc8875e7	com.android.mms	Test me	[Test mes]	0
4,193	1,529,525,185,614	3d589372-7509-49d9-aa3c-374dfc8875e7	com.android.mms	Test mes	[Test mess]	0
4,195	1,529,525,185,876	3d589372-7509-49d9-aa3c-374dfc8875e7	com.android.mms	Test mess	[Test messa]	0
4,197	1,529,525,186,148	3d589372-7509-49d9-aa3c-374dfc8875e7	com.android.mms	Test messa	[Test messag]	0
4,199	1,529,525,186,395	3d589372-7509-49d9-aa3c-374dfc8875e7	com.android.mms	Test messag	[Test message]	0

Figure 3.10: Data from keyboard data

Application sensor The application sensor is the sensor that allows us to see what applications are opened on the system. The plug-in needs this information to create the sessions discussed earlier in section 3.2.3. When the Android system brings forward a new application, a new row of data will be added to the application foreground data table. The structure of the application foreground data table is show in figure 3.11 and example data is provided in figure 3.12. When new data is added to the application foreground table a broadcast with the flag

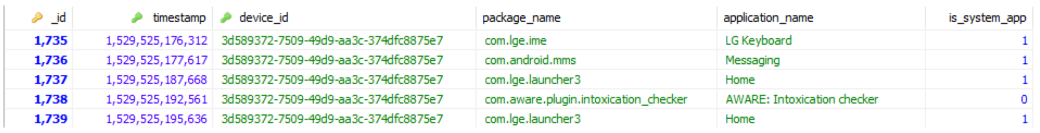
```
Applications.ACTION_AWARE_APPLICATIONS_FOREGROUND
```

will be sent to the system. We will need to listen to this broadcast to start our data processing.



#	Name	Datatype	Length/Set	Unsign...	Allow N...	Zerofill	Default
1	_id	INT	11	☐	▨	☐	AUTO_INCREMENT
2	timestamp	DOUBLE		☐	✓	☐	0
3	device_id	VARCHAR	150	▨	✓	▨	
4	package_name	TEXT		▨	✓	▨	No default
5	application_name	TEXT		▨	✓	▨	No default
6	is_system_app	INT	11	☐	✓	☐	0

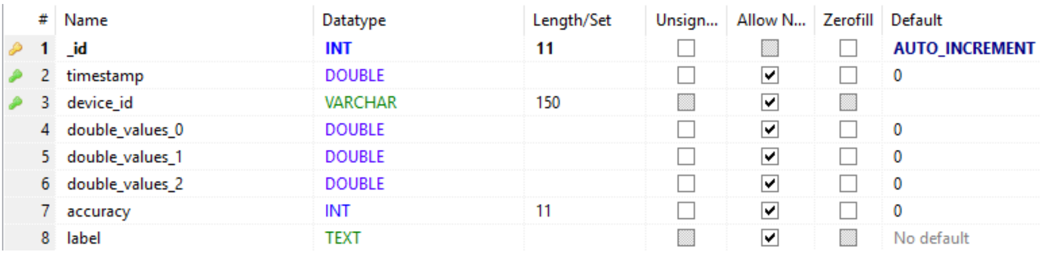
Figure 3.11: SQL table for application data



_id	timestamp	device_id	package_name	application_name	is_system_app
1,735	1,529,525,176,312	3d589372-7509-49d9-aa3c-374dfc8875e7	com.lge.ime	LG Keyboard	1
1,736	1,529,525,177,617	3d589372-7509-49d9-aa3c-374dfc8875e7	com.android.mms	Messaging	1
1,737	1,529,525,187,668	3d589372-7509-49d9-aa3c-374dfc8875e7	com.lge.launcher3	Home	1
1,738	1,529,525,192,561	3d589372-7509-49d9-aa3c-374dfc8875e7	com.aware.plugin.intoxication_checker	AWARE: Intoxication checker	0
1,739	1,529,525,195,636	3d589372-7509-49d9-aa3c-374dfc8875e7	com.lge.launcher3	Home	1

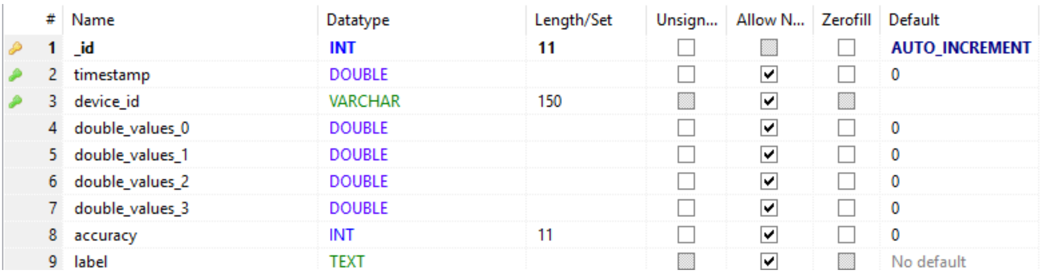
Figure 3.12: Data from application data

Acceleration and rotation sensors The acceleration and rotation sensors track data of the accelerations and rotations that are made with the phone. While the interpretation of this data might be a little bit more tricky, for this project we are merely interested in the changes of data when comparing sober measurements to intoxicated measurements for each session. Again a structure of the data is show in figure 3.13 and figure 3.14 for the rotation. The columns with the name "double_values_0", "double_values_1", "double_values_2", correspond to the X,Y and Z axis of their corresponding measurement. The double_values_3 column on the rotation data table is an optional value that some Android systems have, which will not be used in the development of our plug-in. In figure 3.17 the orientation axis of the accelerometer and rotation meter are displayed.



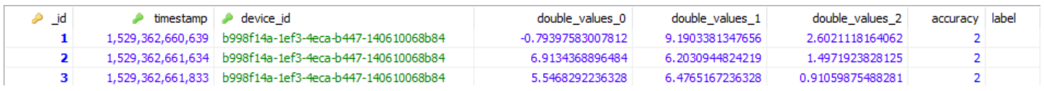
#	Name	Datatype	Length/Set	Unsign...	Allow N...	Zerofill	Default
1	_id	INT	11	☐	▨	☐	AUTO_INCREMENT
2	timestamp	DOUBLE		☐	✓	☐	0
3	device_id	VARCHAR	150	▨	✓	▨	
4	double_values_0	DOUBLE		☐	✓	☐	0
5	double_values_1	DOUBLE		☐	✓	☐	0
6	double_values_2	DOUBLE		☐	✓	☐	0
7	accuracy	INT	11	☐	✓	☐	0
8	label	TEXT		▨	✓	▨	No default

Figure 3.13: SQL table for accelerometer data



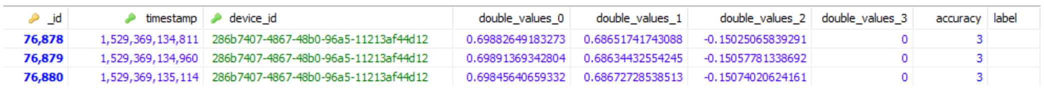
#	Name	Datatype	Length/Set	Unsign...	Allow N...	Zerofill	Default
1	_id	INT	11	☐	▨	☐	AUTO_INCREMENT
2	timestamp	DOUBLE		☐	✓	☐	0
3	device_id	VARCHAR	150	▨	✓	▨	
4	double_values_0	DOUBLE		☐	✓	☐	0
5	double_values_1	DOUBLE		☐	✓	☐	0
6	double_values_2	DOUBLE		☐	✓	☐	0
7	double_values_3	DOUBLE		☐	✓	☐	0
8	accuracy	INT	11	☐	✓	☐	0
9	label	TEXT		▨	✓	▨	No default

Figure 3.14: SQL table for rotation data



_id	timestamp	device_id	double_values_0	double_values_1	double_values_2	accuracy	label
1	1,529,362,660,639	b998f14a-1ef3-4eca-b447-140610068b84	-0.79397583007812	9.1903381347656	2.6021118164062	2	
2	1,529,362,661,634	b998f14a-1ef3-4eca-b447-140610068b84	6.9134368896484	6.2030944824219	1.4971923828125	2	
3	1,529,362,661,833	b998f14a-1ef3-4eca-b447-140610068b84	5.5468292236328	6.4765167236328	0.91059875488281	2	

Figure 3.15: Data from rotation data



_id	timestamp	device_id	double_values_0	double_values_1	double_values_2	double_values_3	accuracy	label
76,878	1,529,369,134,811	286b7407-4867-48b0-96a5-11213af44d12	0.69882649183273	0.68651741743088	-0.15025065839291	0	3	
76,879	1,529,369,134,960	286b7407-4867-48b0-96a5-11213af44d12	0.69891369342804	0.68634432554245	-0.15057781338692	0	3	
76,880	1,529,369,135,114	286b7407-4867-48b0-96a5-11213af44d12	0.69845640659332	0.68672728538513	-0.15074020624161	0	3	

Figure 3.16: Data from rotation data

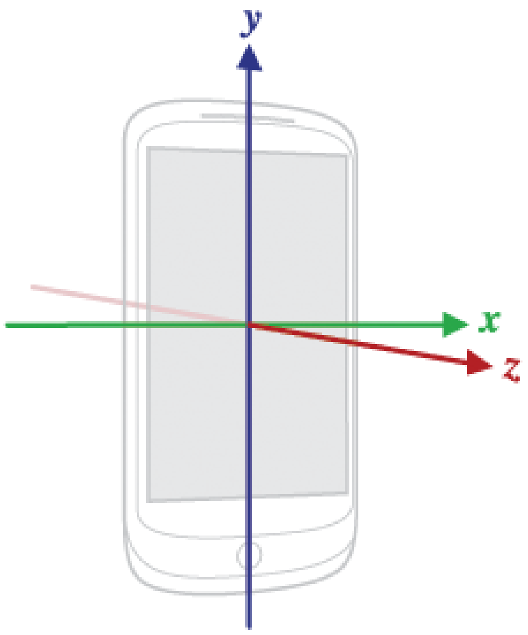


Figure 3.17: Rotation and acceleration axis on the android device

ESMs To connect the data of the application to a certain level of intoxication, the AWARE framework lets the application make use of the Experience Sampling method in short ESM. This allows the application and researchers ask the user questions in several ways. The AWARE framework standard allows the following types of ESMs

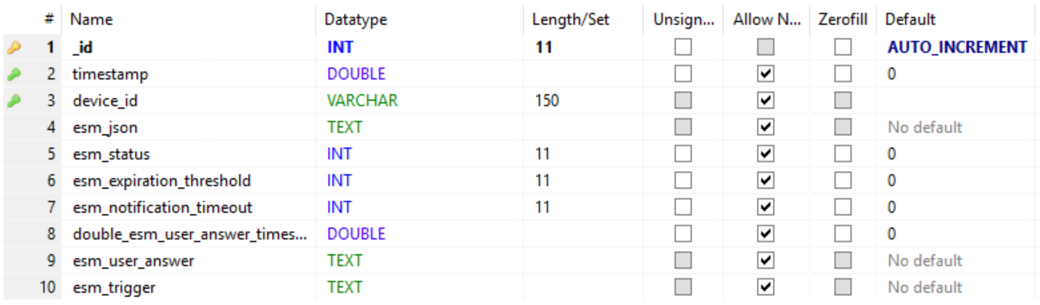
- Free text ESM, used for asking open question like personal opinions.
- Radio box ESM, used to ask to select one option out of a list of options.
- Check box ESM, used to ask the user to select multiple options.
- Likert ESM, used to ask a user to rate some thing with a 0 to 5 or 7 star rating.
- Quick ESM, typically used to ask for a simple yes/no answer.
- Numeric ESM, used to ask for numeric input.

- Scale ESM, used to ask for a rating based on a scale slider.

In this project the application will use a Scale ESM to ask the user to rate his or her level of intoxication, in order to rate the typing sessions. The begin, end and increment values of the ESM can easily be set and labeled. The data table of the scale ESMs is shown in figure 3.18 and example data is shown in figure 3.19.

The creation of a table in the JAVA code is.

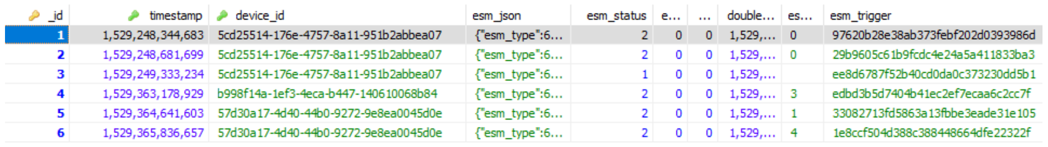
```
try {
     ESMFactory factory = new ESMFactory();
     //define ESM question
     ESM_Scale esmScale = new ESM_Scale();
     esmScale.setScaleMax(10)
     .setScaleMin(0)
     .setScaleStart(0)
     .setScaleMaxLabel("Sober")
     .setScaleMinLabel("Drunk")
     .setScaleStep(1)
     .setTitle("Intoxication rating")
     .setInstructions("Please rate your intoxication level")
     .setExpirationThreshold(15)
     .setSubmitButton("OK");

     //add them to the factory
     factory.addESM(esmScale);

  ESM.queueESM(context, factory.build());
} catch (JSONException e) {
     e.printStackTrace();
}
```



#	Name	Datatype	Length/Set	Unsign...	Allow N...	Zerofill	Default
1	_id	INT	11	☐	▨	☐	AUTO_INCREMENT
2	timestamp	DOUBLE		☐	✓	☐	0
3	device_id	VARCHAR	150	▨	✓	▨	
4	esm_json	TEXT		▨	✓	▨	No default
5	esm_status	INT	11	☐	✓	☐	0
6	esm_expiration_threshold	INT	11	☐	✓	☐	0
7	esm_notification_timeout	INT	11	☐	✓	☐	0
8	double_esm_user_answer_times...	DOUBLE		☐	✓	☐	0
9	esm_user_answer	TEXT		▨	✓	▨	No default
10	esm_trigger	TEXT		▨	✓	▨	No default

Figure 3.18: SQL table for scale ESM data

Summary These are the components from the AWARE framework that are used in the project development. The next section is about the actual use of these components in the application that has been developed along side the code itself.



_id	timestamp	device_id	esm_json	esm_status	e...	...	double...	es...	esm_trigger
1	1,529,248,344,683	5cd25514-176e-4757-8a11-951b2abbea07	{"esm_type":6...	2	0	0	1,529,...	0	97620b28e38ab373febf202d0393986d
2	1,529,248,681,699	5cd25514-176e-4757-8a11-951b2abbea07	{"esm_type":6...	2	0	0	1,529,...	0	29b9605c61b9fcdc4e24a5a411833ba3
3	1,529,249,333,234	5cd25514-176e-4757-8a11-951b2abbea07	{"esm_type":6...	1	0	0	1,529,...		ee8d6787f52b40cd0da0c373230dd5b1
4	1,529,363,178,929	b998f14a-1ef3-4eca-b447-140610068b84	{"esm_type":6...	2	0	0	1,529,...	3	edbd3b5d7404b41ec2ef7ecaa6c2cc7f
5	1,529,364,641,603	57d30a17-4d40-44b0-9272-9e8ea0045d0e	{"esm_type":6...	2	0	0	1,529,...	1	33082713fd5863a13fbbe3eade31e105
6	1,529,365,836,657	57d30a17-4d40-44b0-9272-9e8ea0045d0e	{"esm_type":6...	2	0	0	1,529,...	4	1e8ccf504d388c388448664dfe22322f

Figure 3.19: SQL table for scale ESM data

3.4 Developing the plug-in

Based on the requirements and the design of the application, and the restrictions the AWARE framework has on developing the project, a certain way of creating new context data has been imposed. This means that for the development specific patterns and data flows have been used to create the plug-in.

As mentioned before in section 3.3.1 the AWARE framework and its components send broadcasts to the Android system. When developing the application, it can listen to these broadcasts and then execute bits of code. The Android documentation states that operations should not be more than 10 seconds [4] in the onRecieve of these BroadcastRecievers, but the code this plug-in will need to use, is relatively simple so this does not impact our way of handling new data.

3.4.1 plug-in application flow

The way the developed plug-in works, is relying on Android broadcasts. Changes in collected data of the AWARE framework is communicated through these broadcasts. This gives us an application that is described in figure 3.20. A larger version of this diagram can be found in figure 3.29 at the end of this chapter. The databases in the diagram, are the databases that are collected on the phone.

[4]see: https://developer.android.com/reference/android/content/BroadcastReceiver

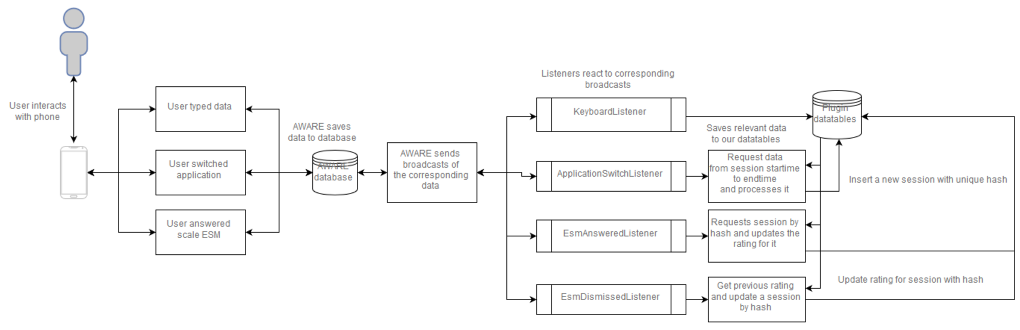

User interacts with phone
User typed data
User switched application
User answered scale ESM
AWARE saves data to database
AWARE database
AWARE sends broadcasts of the corresponding data
Listeners react to corresponding broadcasts
KeyboardListener
ApplicationSwitchListener
EsmAnsweredListener
EsmDismissedListener
Saves relevant data to our datatables
Plugin datatables
Request data from session startime to endtime and processes it
Requests session by hash and updates the rating for it
Get previous rating and update a session by hash
Insert a new session with unique hash
Update rating for session with hash

Figure 3.20: Dataflow of the AWARE plug-in

The plug-in can be split into two parts, the first part is the part where AWARE collects and saves data to device's local database, and then sends a broadcast message to the android system. This process has been described in the previous section.

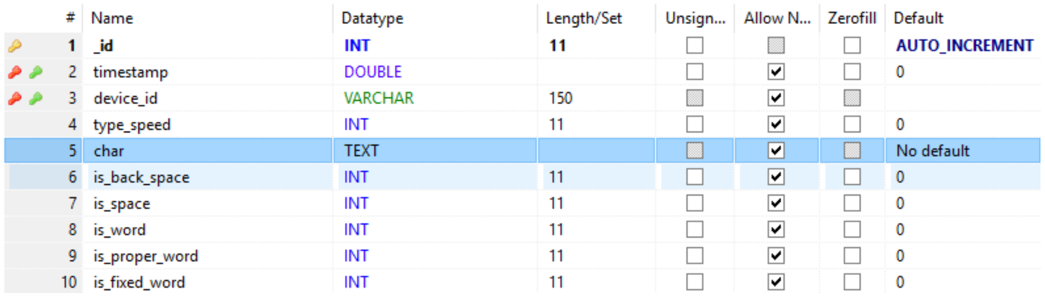
The second part is where the plug-in analyzes and creates new data, from the data that AWARE has collected. This part can again be split into 3 processes that operate separate from each other, but will together put the collected data in the intended data tables. These parts are the individual listeners to the broadcasts that have been sent by AWARE when inserting the collected data. The three groups of listeners are the following:

- The KeyboardListener, which listens to changes in keyboard data.
- The ApplicationSwitchListener, which listens to when an application has been opened.
- The ESMListeners, there are two listeners, one that responds to answered ESM and one that listens to dismissed/canceled ESM responses.

For each listener group, a description will be given about what they do when they receive the message that they are listening to.

3.4.2 KeyboardListener

The keyboard listener listens to changes in keyboard data. It saves new data in a new table called plug-in_intoxication_data_table, the data in the table is shown in figure 3.21. For each character that has been typed a new row is added with the data. Several cases of what a word is have been discussed in the what is a word section 3.2.1. The type speed is based on the timestamp of the currently inserted character and the previous one. The table holds a character column to debug data in the first few phases of test the plug-in, it is removed when a full scale research is conducted.



#	Name	Datatype	Length/Set	Unsign...	Allow N...	Zerofill	Default
1	_id	INT	11	☐	▨	☐	AUTO_INCREMENT
2	timestamp	DOUBLE		☐	☑	☐	0
3	device_id	VARCHAR	150	▨	☑	▨	
4	type_speed	INT	11	☐	☑	☐	0
5	char	TEXT		▨	☑	▨	No default
6	is_back_space	INT	11	☐	☑	☐	0
7	is_space	INT	11	☐	☑	☐	0
8	is_word	INT	11	☐	☑	☐	0
9	is_proper_word	INT	11	☐	☑	☐	0
10	is_fixed_word	INT	11	☐	☑	☐	0

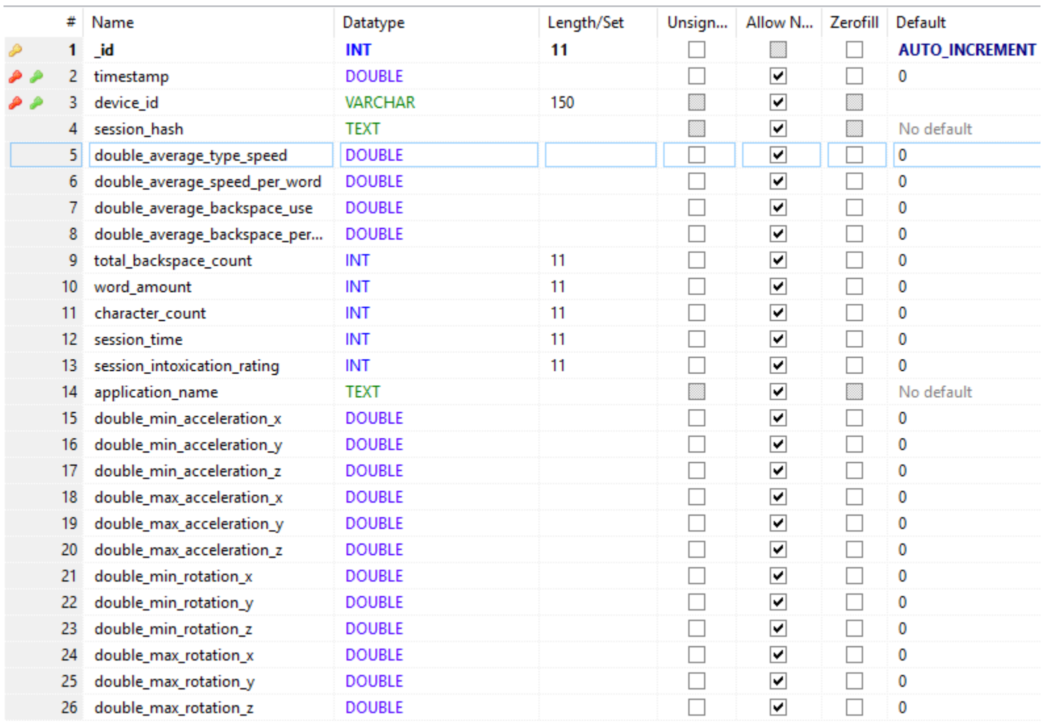
Figure 3.21: Character meta data

3.4.3 ApplicationSwitchListener

The application switch listener listens to the broadcast that is emitted when AWARE saves new data about an application that has been opened on the Android system. When a new application is opened, the listener will ask for the timestamps of the current and the previously opened application. This will be the time frame of a new session. When a new time frame is found the application listener will request all entries of the plug-in_intoxication_data_table between these times. When no more than 0 or 1 rows have been found, it means that no data has been typed in the time frame, and thus in the application that was opened at that time. Therefore nothing will be done with these sessions. When the data returned does return more than 1 character of data, the listener will calculate relevant data for insertion into the session. This session data will be saved in a table called plug-in_session_data_table. The data contained in this table is shown in figure 3.22. Most columns in this table are named after the data it holds but some need more explanation.

The session_hash is a unique has that is created when inserting the data. This hash will also be used to query the session table data when inserting the rating that a user entered when answering the scale ESM. The rating will then be inserted in the session_intoxication_rating column.

When a new row has been entered in the session table, a scale ESM will be created, that will be prompted to the user. This ESM will have the session_hash value of the session as the trigger for the ESM.



#	Name	Datatype	Length/Set	Unsign...	Allow N...	Zerofill	Default
1	_id	INT	11	☐	▨	☐	AUTO_INCREMENT
2	timestamp	DOUBLE		☐	☑	☐	0
3	device_id	VARCHAR	150	▨	☑	▨	
4	session_hash	TEXT		▨	☑	▨	No default
5	double_average_type_speed	DOUBLE		☐	☑	☐	0
6	double_average_speed_per_word	DOUBLE		☐	☑	☐	0
7	double_average_backspace_use	DOUBLE		☐	☑	☐	0
8	double_average_backspace_per...	DOUBLE		☐	☑	☐	0
9	total_backspace_count	INT	11	☐	☑	☐	0
10	word_amount	INT	11	☐	☑	☐	0
11	character_count	INT	11	☐	☑	☐	0
12	session_time	INT	11	☐	☑	☐	0
13	session_intoxication_rating	INT	11	☐	☑	☐	0
14	application_name	TEXT		▨	☑	▨	No default
15	double_min_acceleration_x	DOUBLE		☐	☑	☐	0
16	double_min_acceleration_y	DOUBLE		☐	☑	☐	0
17	double_min_acceleration_z	DOUBLE		☐	☑	☐	0
18	double_max_acceleration_x	DOUBLE		☐	☑	☐	0
19	double_max_acceleration_y	DOUBLE		☐	☑	☐	0
20	double_max_acceleration_z	DOUBLE		☐	☑	☐	0
21	double_min_rotation_x	DOUBLE		☐	☑	☐	0
22	double_min_rotation_y	DOUBLE		☐	☑	☐	0
23	double_min_rotation_z	DOUBLE		☐	☑	☐	0
24	double_max_rotation_x	DOUBLE		☐	☑	☐	0
25	double_max_rotation_y	DOUBLE		☐	☑	☐	0
26	double_max_rotation_z	DOUBLE		☐	☑	☐	0

Figure 3.22: Session data

3.4.4 ESMAnsweredListener & ESMDismissedListener

When a new session has been inserted the SessionListener also created an ESM for the user to be answered. When the ESM has either been answered or dismissed/canceled the AWARE broadcasts a corresponding message across the Android platform.

ESMAnsweredListener This listener is triggered when the user answered the ESM and pressed the submit button on the prompt. The listener then requests ESM's data and the session that was associated with the ESM via the esm_trigger field in the ESM's data table that AWARE manages. This session's session_intoxication_rating is then updated with the esm_user_answer field of the ESM's data.

ESMDismissedListener This listener listens to the ESMs that are either dismissed or expired. When this happens the listener will query the ESM data table to find the first answered ESM and uses that answer to update the associated session in the same way the ESMAnsweredLister does.

3.4.5 Summary

When putting all this data together researchers can start analyzing the data that has been made available through the individual character analysis and the session data.

3.5 Code implementations

This section will explain the plug-in's actual implementations discussed in the previous section, with small code snippets. If a full working of the system is desired, it is advised took look at the available code and the comments in it, alongside this report.

3.5.1 The Java classes

The main Java classes used in this development of this plug-in are the following.

- Plugin.java: this is the main class that AWARE uses as a basis of the plug-in.
- IntoxicationProvider.java: This class manages the saving and querying the data related to this plug-in.
- ContextCard.java: This is the class that manages the data that is shown in the stream view of the AWARE application, it is mainly used to show that the plug-in is actually running and working.
- TypeSession.java: This is a utility class that allows for easier session saving.
- MyUtils.java: This class contains a few convenience methods that make it easier to write consisted and cleaner code.

Plugin.java The Plugin class is the core starting point of an AWARE plug-in. It hosts the lifecycle callbacks of the plug-in in the Android environment. It contains the command:

```java
public int onStartCommand(Intent intent, int flags, int startId);
```

This command is run every 5 minutes by AWARE to ensure the plug-in is still running. When the permission that are needed for the plug-in are flagged as PERMISSIONS_OK the application can start enabling the several sensors that are needed for the plug-in. The sensors and the corresponding listeners are initialize with the code snippet shown in 3.23

In the onDestroy method of this class listeners are removed again to avoid memory leaks. This is shown in snippet 3.24

The plug-in class also contains the definitions of the listeners that are used in the plug-in as inner classes. These listeners are described in section 3.4 and should be understood by reading the .java files of these classes.

The IntoxicationProvider.java This class manages the saving and querying data the plug-in creates. It creates the database tables and holds all the information of the table data columns. It also defines what columns should be returned and saved when querying and inserting data. To avoid having too much code in this document only smaller snippets of the most critical parts are shown.

In this class the table names are defined as a public final string, so they can consistently be used across the multiple classes in this project. This is also done for the column names of the tables, they are defined in an inner class of the IntoxicationProvider class. These inner classes also hold content uri that android uses to request the data via the ContentResolver. The constants are defined as can be seen in snippet 3.25.

In the onCreate of the provider class certain uris are matched with an id and datamaps are created to be used when querying data, so the database knows what columns to request. When querying the data through Android's ContentResolver a uri has to be provided so it knows what data to get. These uris have to match with the uris that are listened to in the IntoxicationProvider class. When a uri matches a switch block is triggered to see what data is requested, and then either a projection map is assigned for querying the data or a custom query is executed to get the sum and averages of columns. This is shown in code snippet 3.26 Most of the other code in the IntoxicationProvider is done according to Android's standards in ContentProviders, so for more information on that it's easier to look up in the source files.

The ContextCard class and the TypeSession class are not very complicated and are explained in the source files.

MyUtils.java This class contains a few functions that make clean and consistent code more easy to write. Because of the importance of some of these functions, they will briefly be discussed. The first function is the getIntoxicationTypeSpeedDataForSession, which is defined in snippet 3.27. This will query the IntoxicationProvider for the sum, average and row count, of the plug-in_intoxication_data_table 3.21 between the start and end time. This is used for getting the right data when inserting a new session. Some similar functions are defined to query other data like backspaces word data and acceleration/rotation as well.

The utility class also contains a function for creating a hash, which now only uses a timestamp, but for better uniqueness could also be used with more than just the timestamp. This is the function that creates the hash for querying back sessions that have their corresponding ESMs answered. The createIntoxicationESM function is to more quickly and consistently create a session feedback ESM. This is shown in snippet 3.28

3.5.2 Summary

The code implementation as described above describes the plug-in as it was made only partly. Reading the code inside a popular code environment is highly recommended to fully understand what is happening. In combination with chapter 3 of this report everything will be made clear. Full understanding of what is happening inside the plug-in is not needed for the use in research project as it is developed as an "as is" plug-in for AWARE.

Figure 3.23: Plugin.java listener initialization

```
//Initialize our plug-in's settings
Aware.setSetting(this, Settings.STATUS_plug-in_TEMPLATE, true);
//Initialize the accelerometer.
Aware.setSetting(this, Aware_Preferences.STATUS_ACCELEROMETER,
    true);
Aware.startAccelerometer(this);
//Send broadcasts so the data can be read to be displayed at the
    context card/stream view.
Accelerometer.setSensorObserver(new
    Accelerometer.AWARESensorObserver() {
    @Override
    public void onAccelerometerChanged(ContentValues contentValues) {
        sendBroadcast(new
            Intent("ACCELERATION_DATA").putExtra("data",
            contentValues));
    }
});
Aware.setSetting(this, Aware_Preferences.STATUS_ROTATION, true);
Aware.startRotation(this);
Rotation.setSensorObserver(new Rotation.AWARESensorObserver() {
    @Override
    public void onRotationChanged(ContentValues contentValues) {
        sendBroadcast(new Intent("ROTATION_DATA").putExtra("data",
            contentValues));
    }
});
//Start the keyboard data logging and Keyboard listener.
Aware.startKeyboard(this);
Aware.setSetting(this, Aware_Preferences.STATUS_KEYBOARD, true);
IntentFilter filter = new IntentFilter();
filter.addAction(Keyboard.ACTION_AWARE_KEYBOARD);
registerReceiver(keyboardListener, filter);
//Start and register ESM listeners
Aware.setSetting(this, Aware_Preferences.STATUS_ESM, true);
IntentFilter esmFilter = new IntentFilter();
esmFilter.addAction(ESM.ACTION_AWARE_ESM_ANSWERED);
registerReceiver(esmAnsweredListener, esmFilter);
IntentFilter esmDismissFilter = new IntentFilter();
esmDismissFilter.addAction(ESM.ACTION_AWARE_ESM_DISMISSED);
esmDismissFilter.addAction(ESM.ACTION_AWARE_ESM_EXPIRED);
registerReceiver(esmDismissedListener, esmDismissFilter);
//Start application logging and set listener.
Aware.setSetting(this, Aware_Preferences.STATUS_APPLICATIONS, true);
IntentFilter applicationFilter = new IntentFilter();
applicationFilter.addAction(Applications.ACTION_AWARE_APPLICATIONS_FOREGROUND);
registerReceiver(applicationSwitchListener, applicationFilter);
```

Figure 3.24: Plugin.java onDestroy method

```
super.onDestroy();
//Turn off the sync-adapter if part of a study
if (Aware.isStudy(this) &&
    (getApplicationContext().getPackageName().equalsIgnoreCase("com.aware.phone")
    ||
    getApplicationContext().getResources().getBoolean(R.bool.standalone)))
    {
    ContentResolver.removePeriodicSync(
            Aware.getAWAREAccount(this),
            Provider.getAuthority(this),
            Bundle.EMPTY
    );
}
//Unregister the keyboard listener to prevent memory leaks.
if (keyboardListener != null)
    unregisterReceiver(keyboardListener);
if (applicationSwitchListener != null)
    unregisterReceiver(applicationSwitchListener);
if (esmAnsweredListener != null)
    unregisterReceiver(esmAnsweredListener);
if (esmDismissedListener != null) {
    unregisterReceiver(esmDismissedListener);
}
Aware.setSetting(this, Settings.STATUS_plug-in_TEMPLATE, false);
//Stop AWARE instance in plug-in
Aware.stopAWARE(this);
```

Figure 3.25: The intoxication provider constants

```
public static String INTOXICATION_TABLE_NAME =
    "plug-in_intoxication_data_table";
public static String SESSION_TABLE_NAME =
    "plug-in_session_data_table";
public static String DATABASE_NAME = "plug-in_intoxication_data.db";
public static final String[] DATABASE_TABLES =
    {INTOXICATION_TABLE_NAME, SESSION_TABLE_NAME};

public static final class Session_Data implements AWAREColumns {
   public static final Uri CONTENT_URI = Uri.parse("content://" +
       AUTHORITY + "/" + SESSION_TABLE_NAME);
   public static final String APPLICATION_NAME = "application_name";
   public static final String SESSION_HASH = "session_hash";
   public static final String AVERAGE_TYPE_SPEED =
       "double_average_type_speed";
}
```

Figure 3.26: IntoxicationProvider onCreate and onQuery methods

```
//Added in the onCreate
sUriMatcher.addURI(AUTHORITY, DATABASE_TABLES[0] + "/calculated/" +
    Intoxication_Keyboard_Data.TYPE_SPEED, CALCULATED_SPEED_DATA);

//Called in onQuery
switch (sUriMatcher.match(uri)) {
    case INTOXICATION_KEYBOARD_DATA:
    //Here a projection map is set to the query builder.
        qb.setTables(DATABASE_TABLES[0]);
        qb.setProjectionMap(intoxicationTypeDataMap);
        break;
    case SESSION_DATA:
        qb.setTables(DATABASE_TABLES[1]);
        qb.setProjectionMap(sessionDataMap);
        break;
    /*This query selects speed data from the
        plug-in_intoxication_data_table
      * between two timestamps. In this particular case speed above
          a certain threshold are ignored.
      * This is because the first character typed after a long wait
          time is most likely a wait that was not inbetween word,
          and there is no type speed for each first character of a
          message.
      */
    case CALCULATED_SPEED_DATA:
        cursor = databaseHelper.getReadableDatabase().rawQuery("" +
                "SELECT SUM(" + Intoxication_Keyboard_Data.TYPE_SPEED
                    + "), " +
                "AVG(" + Intoxication_Keyboard_Data.TYPE_SPEED + "),"
                    +
                "COUNT(" + AWAREColumns._ID + ") " +
                "FROM " +
                    IntoxicationProvider.INTOXICATION_TABLE_NAME +
                " WHERE (" + AWAREColumns.TIMESTAMP + " BETWEEN ? AND
                    ?) AND " + Intoxication_Keyboard_Data.TYPE_SPEED
                    + " < 3000", selectionArgs);
    break;
}
```

Figure 3.27: MyUtils getIntoxicationTypeSpeedDataForSession method

```
public static Cursor getIntoxicationTypeSpeedDataForSession(Context
    context, long start, long end) {
    return context.getContentResolver().query(
      Uri.parse(IntoxicationProvider.Intoxication_Keyboard_Data.CONTENT_URI
          + "/calculated/" +
          IntoxicationProvider.Intoxication_Keyboard_Data.TYPE_SPEED),
      null,
      null,
      new String[]{String.valueOf(start), String.valueOf(end)},
      null
   );
}
```

Figure 3.28: MyUtils createIntoxicationESM method

```
public static void createIntoxicationESM(Context context, String
    sessionHook) throws JSONException {
   //currenttime is initialized high so if there is no previous ESM
       record the check for the ESM interval time will always fail
       and thus never show an ESM.
   long currentEsmTime = 999999999;
   long previousEsmTime = 0;
   Cursor lastEsm =
       context.getContentResolver().query(ESM_Provider.ESM_Data.CONTENT_URI,
           null,
           null, null,
           "timestamp DESC LIMIT 1");
   if (lastEsm != null && lastEsm.moveToFirst()) {
       previousEsmTime =
           lastEsm.getLong(lastEsm.getColumnIndex("timestamp"));
       currentEsmTime = System.currentTimeMillis();
   }
   //break if last esm was to close to previous one.
   if ((currentEsmTime - previousEsmTime) < 2000) {
       return;
   }
   ESMFactory factory = new ESMFactory();
   ESM_Scale scaleESM = new ESM_Scale();
   scaleESM.setTitle(context.getString(R.string.intoxication_slider_title));
   scaleESM.setInstructions(context.getString(R.string.intoxication_slider_instructions)
   scaleESM.setScaleMax(10);
   scaleESM.setScaleStep(1);
   scaleESM.setScaleMin(0);
   scaleESM.setScaleStart(0);
   scaleESM.setExpirationThreshold(15);
   scaleESM.setScaleMinLabel(context.getString(R.string.scale_min_label));
   scaleESM.setScaleMaxLabel(context.getString(R.string.scale_max_label));
   scaleESM.setTrigger(sessionHook);
   scaleESM.setSubmitButton(context.getString(android.R.string.ok));
   factory.addESM(scaleESM);
   ESM.queueESM(context, factory.build());
}
```

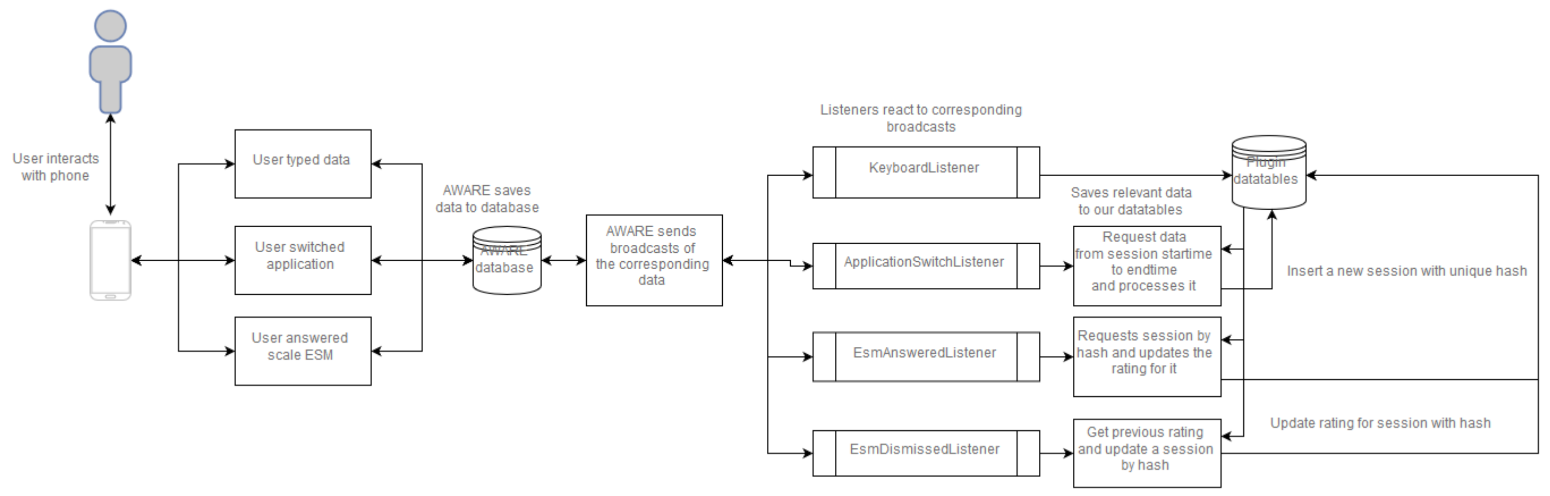

User interacts with phone
User typed data
User switched application
User answered scale ESM
AWARE saves data to database
AWARE database
AWARE sends broadcasts of the corresponding data
Listeners react to corresponding broadcasts
KeyboardListener
ApplicationSwitchListener
EsmAnsweredListener
EsmDismissedListener
Saves relevant data to our datatables
Plugin datatables
Request data from session startime to endtime and processes it
Requests session by hash and updates the rating for it
Get previous rating and update a session by hash
Insert a new session with unique hash
Update rating for session with hash

Figure 3.29: Dataflow of the AWARE plug-in

Chapter 4

Evaluation

When testing the application users were asked to install the aware application on their phones. Since google does not allow applications anymore that might have malicious use intentions, the AWARE application could not anymore be installed through the Google Play store. This is because the application is allowed track the user's input of text through the accessibility services of the Android system, as is mentioned before in section 3.3.1. While these features should not be seen as a security issue in this case, the general use of the accessibility features in an application has to be defended and explained when uploading to the Play store. Therefore the developers at the time had decided to distribute the AWARE application through a different platform that they maintained themselves. From this platform the application was downloaded to our own devices and installed.

4.1 Experimental setup

When the plug-in was tested and used by our test users the same version of the AWARE application was used. In this case that was version

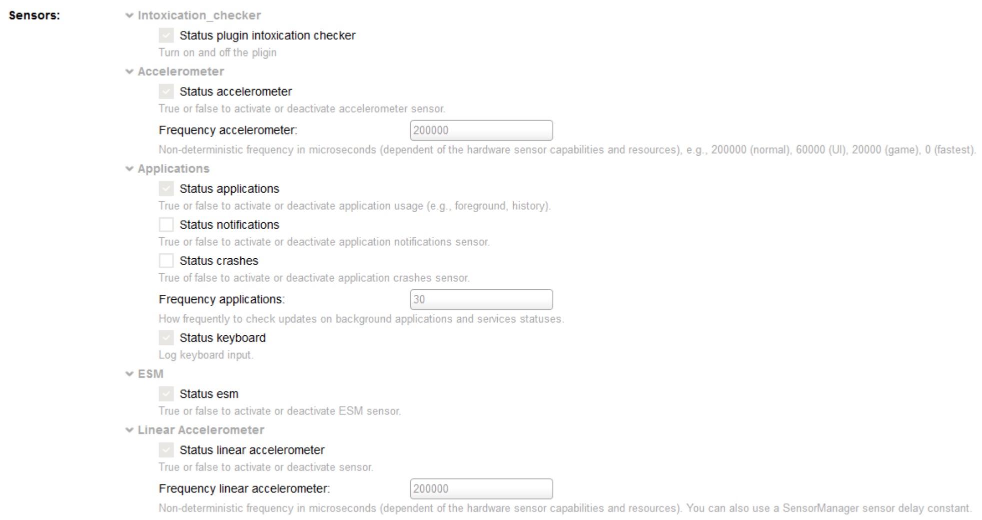
- AWARE version : 4.0.708.master

For convenience of our test users 2 methods of distribution were used that are commonly used by people with smartphones. The application was either installed by downloading the AWARE application with a Dropbox [1] download link, or by downloading it as an attachment through Email. When users installed the application they were asked to allow all permissions that application was asking for. In addition to this, the application needs accessibility services to be enabled, in order for the aware application to work. These

[1]https://www.dropbox.com/

features have been discussed in section 3.3.1. A study using the intoxication plug-in was then joined using the QR code scanner of the application. If the plug-in needed for this application had not been installed yet, it was be downloaded and installed through the study joining screen.

When all of this had been done a check was made to be sure the right sensors were enabled on the application. Sometimes because the application didn't have accessibility features on, the keyboard sensor of the application would not be enabled and thus had to be enabled manually. A check was done to see if all needed sensors were enabled. Checking the sensors would include turning on the, acceleration sensor, linear acceleration sensor, the application sensor, and in this menu also the keyboard sensor. When these sensors are selected on the dashboard on the server, they should be turned on automatically, but it helps to check if the sensors are actually turned on in the application. In figure 4.1 is shown what the dashboard should look like when the study is set up. When the application is running as it is supposed to, on the stream screen, it will display a value showing the last recorded type speed (if any data was typed since the plug-in was running) and the live values of the accelerometer.



Sensors:

- Intoxication_checker
  - ✓ Status plugin intoxication checker
    Turn on and off the pligin
- Accelerometer
  - ✓ Status accelerometer
    True or false to activate or deactivate accelerometer sensor.
  - Frequency accelerometer: 200000
    Non-deterministic frequency in microseconds (dependent of the hardware sensor capabilities and resources), e.g., 200000 (normal), 60000 (UI), 20000 (game), 0 (fastest).
- Applications
  - ✓ Status applications
    True or false to activate or deactivate application usage (e.g., foreground, history).
  - ☐ Status notifications
    True or false to activate or deactivate application notifications sensor.
  - ☐ Status crashes
    True of false to activate or deactivate application crashes sensor.
  - Frequency applications: 30
    How frequently to check updates on background applications and services statuses.
  - ✓ Status keyboard
    Log keyboard input.
- ESM
  - ✓ Status esm
    True or false to activate or deactivate ESM sensor.
- Linear Accelerometer
  - ✓ Status linear accelerometer
    True or false to activate or deactivate sensor.
  - Frequency linear accelerometer: 200000
    Non-deterministic frequency in microseconds (dependent of the hardware sensor capabilities and resources). You can also use a SensorManager sensor delay constant.

Figure 4.1: Enabled sensors on the dashboard

The test group The group of people used to test the application and gather data with consists of 5 people aged between 18 and 30 years. The group consisted of 3 male users and 2 female. All of the users were students at the time of conducting the test. All of the users had prior experience with drinking, and smart phones. When analyzing the collected data, gender and age should not matter because different people might already have different typing patterns than others. Therefore it is mostly relevant to know how each user individually types sober, as opposed to how he or she types intoxicated. However when testing the application and how it is perceived by different age groups, it might make a difference to include age and gender.

The setup When testing the application, an environment was used that might feel familiar to most of the users. This means that the tests were not conducted in an environment where a user would not normally drink. These locations where either at home, a bar or another social event. When testing the plug-in, beers or other alcoholic beverages were consumed, but they were never motivated or forced to drink anything. Users were voluntary cooperating in drinking, and had their own decision of the speed in which they would like to drink. The duration of a session was never defined, as in real use cases of the plug-in, a duration is usually irrelevant as well.

Besides the sessions that were planned with people, the application has also been running for longer times on each phone it was installed. This is because this way we could collect more data and thus have a better insight of what the application was doing. Because a rating of intoxication is always added to a type session, not knowing when a person was drinking should not influence data collection. Of course when people deliberately try to ruin the data this can not be said, but in a serious study we can assume this would not be an issue.

4.2 Evaluation

In this section an evaluation is given based on how users perceived the plug-in when using it on their own phones. The questions that were asked in the questionnaire were the following:

- Do you think you type different when you are intoxicated by alcohol?
- Do you think an application that uses typing analysis would be able to track intoxication signs?
- Do you feel like this application respects your privacy?

These questions were answered with the options, yes, no, and maybe.

In addition to these questions a SUS [2] usability questionnaire was added as well. This questionnaire aims to identify the usability of the developed

[2]https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html

plug-in, and can be used for other systems too. The full name of the test is System Usability Scale. It gives a global view of subjective assessments of usability. The users are given 10 questions that have to be rated 1 to 5, where 1 is strongly disagree with the statement, and 5 strongly agree with the statement. A score is then calculated as following. First sum the score contributions from each item. Each item's score contribution will range from 0 to 4. For items 1, 3, 5, 7,and 9 the score contribution is the scale position minus 1. For items 2, 4, 6, 8 and 10, the contribution is 5 minus the scale position. Multiply the sum of the scores by 2.5 to obtain the overall value of SU. SUS scores have a range of 0 to 100. Scores above 68 can be considered above average, and anything below can be seen as below average. It is suggested that scores are normalized and then looked at percentiles, to make sure the scoring makes sense as to what it is trying to explain. In this case that would be the usability of the plug-in. The SUS questionnaire consists of the following questions:

1. I think that I would like to use this system frequently
2. I found the system unnecessarily complex
3. I thought the system was easy to use
4. I think that I would need the support of a technical person to be able to use this system
5. I found the various functions in this system were well integrated
6. I thought there was too much inconsistency in this system
7. I would imagine that most people would learn to use this system very quickly
8. I found the system very cumbersome to use
9. I felt very confident using the system
10. I needed to learn a lot of things before I could get going with this system

Because sometimes it is hard to predict where users might have problem with in an application that runs in the background an open question was added to ask the users for any feedback they might have with the application. The question was formulated as following:

- What would you like to say about the system that could be relevant to evaluating the system?

The questionnaire was conducted through an online platform, so users could do it anytime they liked.

4.3 Results

In this section we show some of the results that came out of the tests that were conducted. Two aspects are covered in the tests. In the first part we discuss the usability of the application, and the other part we show the results of the data that is collected, and if the data complies with what we wanted to achieve.

4.3.1 Usability evaluation

The usability questionnaire explains us more about how the application was perceived by the users. This is a subjective measure of each user's experience with the application, its intended use. Results of the first part of the questionnaire are shown in figure 4.2

As can be seen in the first chart 4.3, most people feel like they type differently when they are intoxicated. This is an important factor when assessing if the approach we used to collect data to alcohol intoxication, is an appropriate one. It indicates that most people feel like they will type different based on the level of intoxication. The second chart shown in figure 4.4 seems to support this claim as well. As we suspected, in this project other people feel that data about typing could possibly be used to detect alcohol intoxication indicators.

The third chart shown in figure 4.5, gives us an insight about the user's sense of privacy invasion. Due to the low amount of testing users, it should be suggested that 40% of the users saying maybe, might be a possible indication that users are not clear about what exactly is being tracked. When assessing the issue of privacy, this data should be taken in consideration with importance. In the answers to the open question that was added to the questionnaire was mentioned: *I didn't really know what happened with messages and how my privacy wouldn't be infiltrated. I would like to know more about that in the future.* This should definitely be taken into consideration, because it suggests unclarity of how data is processed by the application. This is likely not because the data is not respected, but it seems that users need some feedback in order to trust that the system is doing what is says it is doing.

Figure 4.2: Questionnaire pie charts

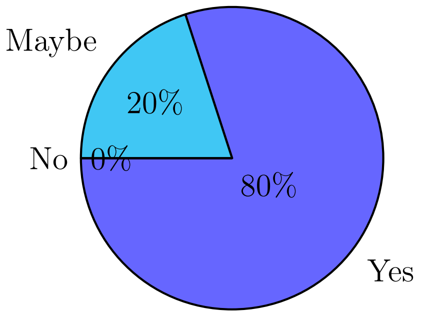


Figure 4.3: Do you think you type different when you are intoxicated by alcohol?

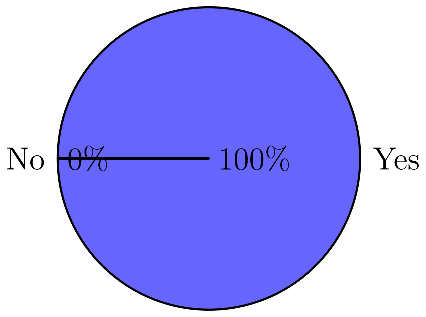


Figure 4.4: Do you think an application that uses typing analysis would be able to track intoxication signs?

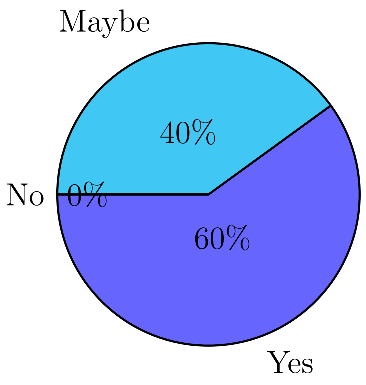


Figure 4.5: Do you feel like this application respects your privacy?

The open question asking

What would you like to say about the system that could be relevant to evaluating the system?

was only answered by 3 of the 5 people interviewed about the tested application. Some of the users had also been involved in testing earlier versions in testing so some of the comments could refer to an earlier test version of the application. The answers are the following:

- "Firstly, sometimes I would get a double amount of notifications, so I had to answer the question multiple times for just 1 session, which was annoying/cumbersome. Secondly, I had to rate whenever I closed the keyboard to be able to see the whole screen. So I got a notification before I ended my session, which resulted in multiple notifications in a short amount of time. That also was cumbersome. Lastly, I didn't really know what happened with messages and how my privacy wouldn't be infiltrated. I would like to know more about that in the future."
- "It's a work in progress"
- "My phone ran out of memory after installing the application"

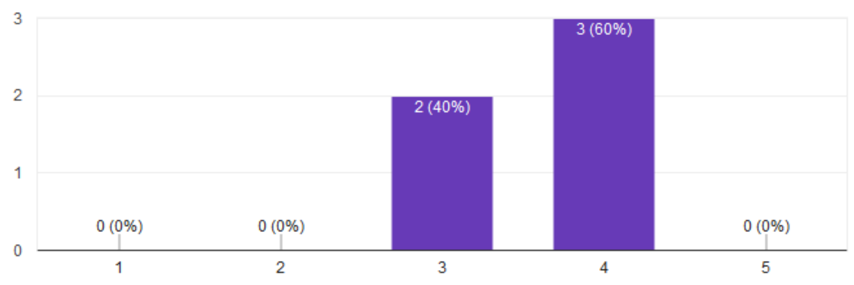
Some of the comments made were not foreseen in developing the application, but they will be mentioned in our discussion to make our conclusions about the developed plug-in.

The results of the SUS questionnaire gave us an average score of 81,5. This should mean that our application was usable for the tested group of people. We expect others in this demographic, to be also able to use the application. This does mean that, as mentioned before in the section 4.1, people should know how to use a smartphone in the first place. The results of the SUS questionnaire are shown in figure 4.6

Figure 4.6: SUS questionnaire answers

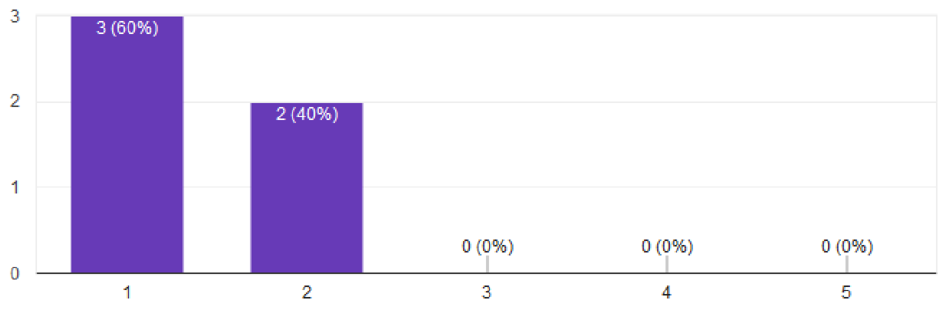
I think that I would like to use this system frequently

5 responses



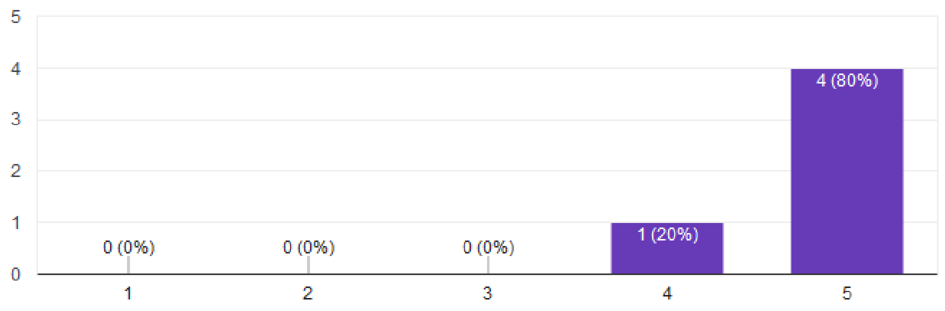
I found the system unnecessarily complex

5 responses



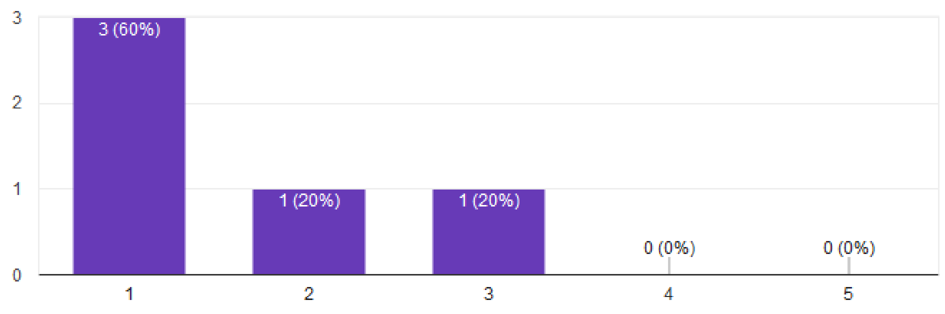
I thought the system was easy to use

5 responses



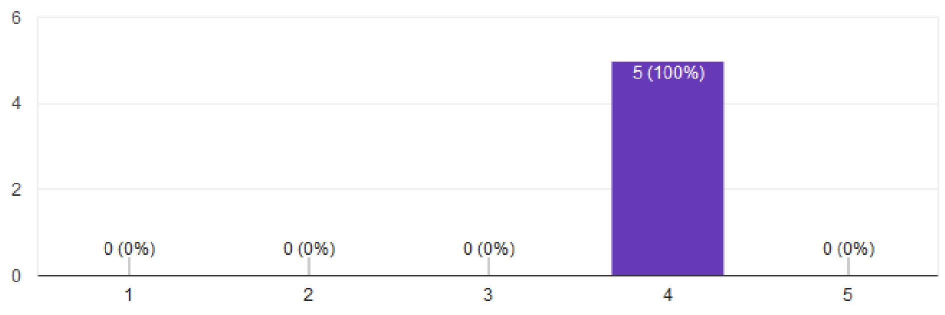
I think that I would need the support of a technical person to be able to use this system

5 responses



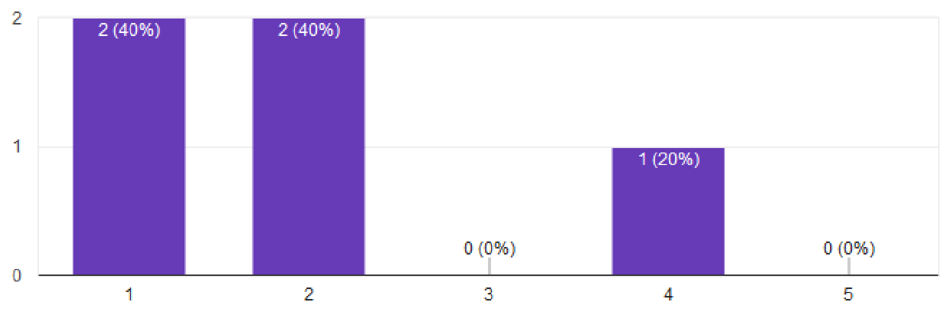
I found the various functions in this system were wel integrated

5 responses



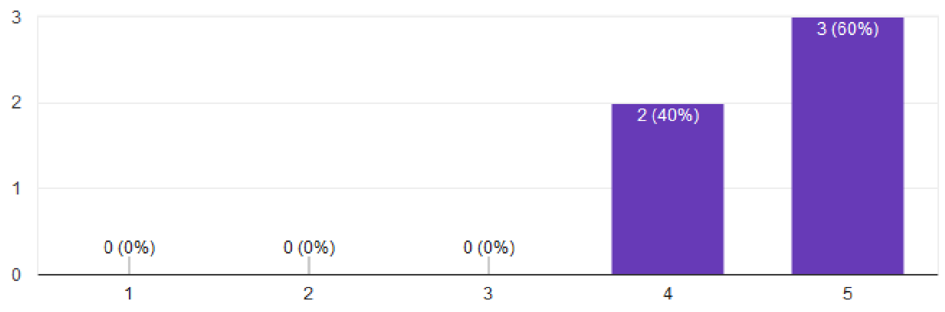
I thought there was too much inconsistency in this system

5 responses



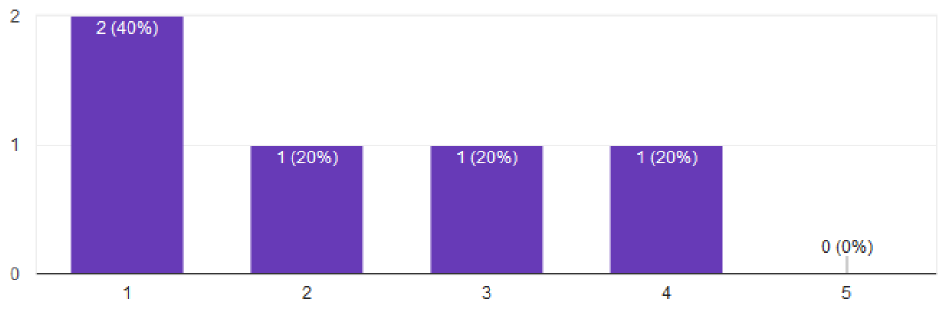
I would imagine that most people would learn to use this system very quickly

5 responses



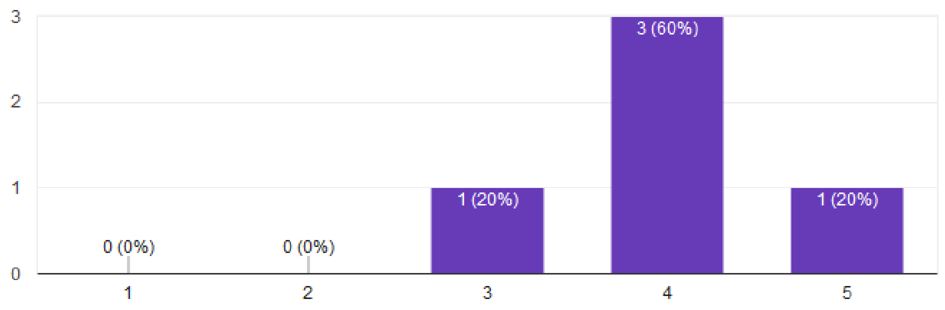
I found the system very cumbersome to use

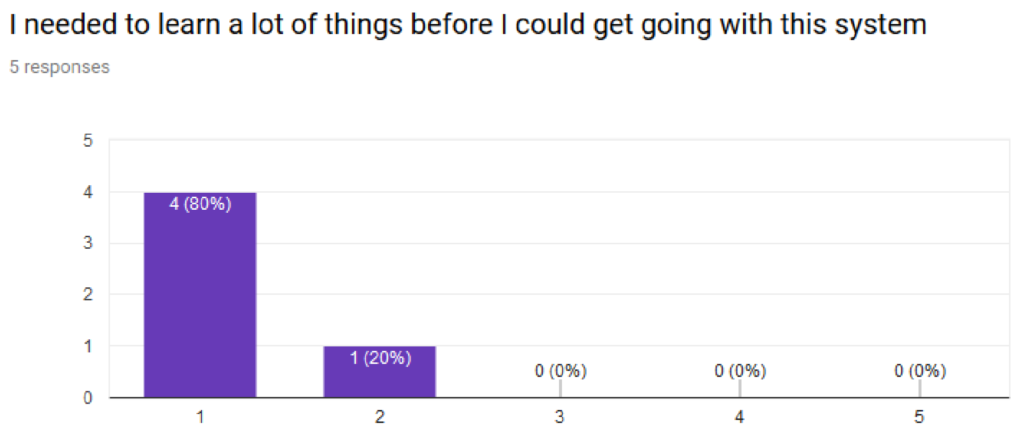
5 responses



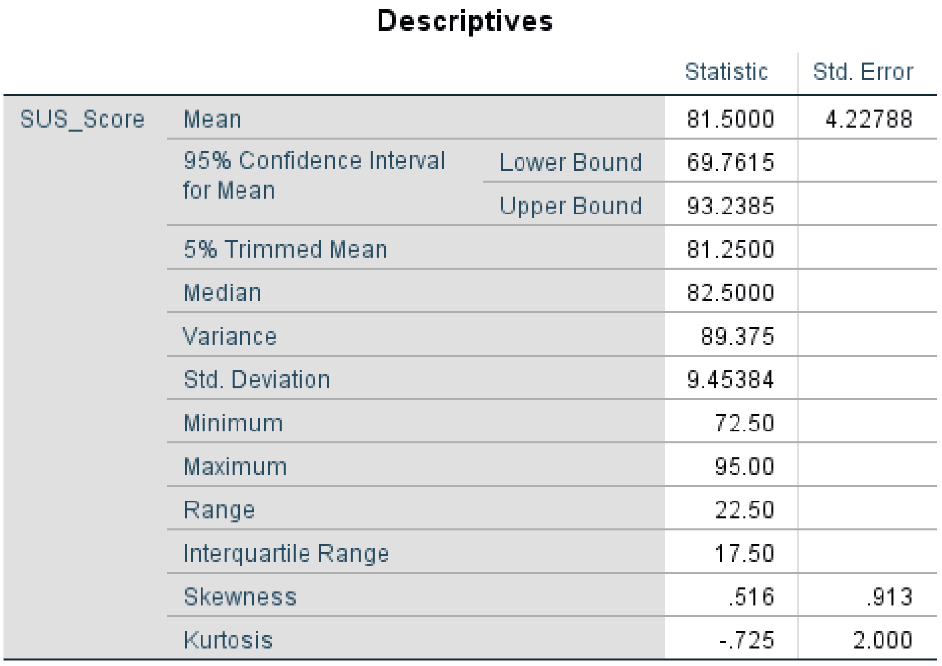
I felt very confident using the system

5 responses





Based on these answers the scores were calculated as mentioned before. The descriptives of the scores can be found in figure 4.7 and a corresponding box plot in figure 4.8. For the small dataset that there is, the data does not seem to suggest a lot of deviation in what users think about the usability of the application. No outliers can be seen in the box plot, and score values are quite close to each other.



Descriptives

			Statistic	Std. Error
SUS_Score	Mean		81.5000	4.22788
	95% Confidence Interval for Mean	Lower Bound	69.7615	
		Upper Bound	93.2385	
	5% Trimmed Mean		81.2500	
	Median		82.5000	
	Variance		89.375	
	Std. Deviation		9.45384	
	Minimum		72.50	
	Maximum		95.00	
	Range		22.50	
	Interquartile Range		17.50	
	Skewness		.516	.913
	Kurtosis		-.725	2.000

Figure 4.7: Descriptives for the SUS scores

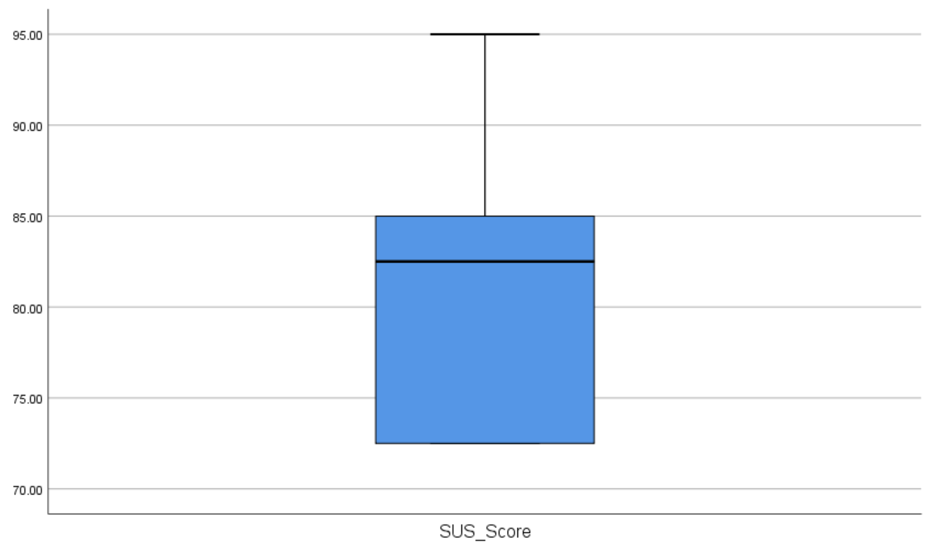
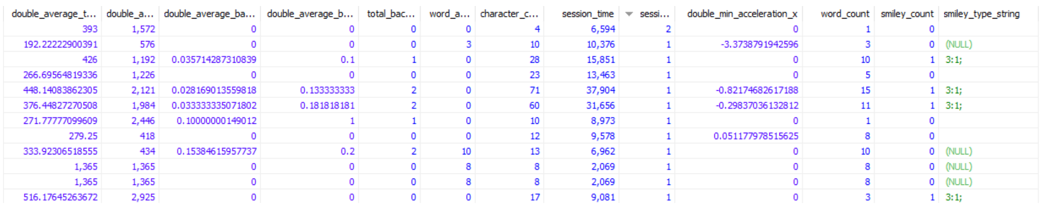


Figure 4.8: Box plot for the SUS scores

4.3.2 Data collection evaluation

When evaluating the plug-in we should also look at the data that is collected during the sessions. As the data in this study is merely first step into developing a machine learning model, we are more interested in what data is collected and if it looks consistent enough to do data analysis with. As discussed in chapter 3, data is collected in sessions. The session data that was collected during the testing moments is shown in figure 4.9. In the table we



double_average_t...	double_a...	double_average_ba...	double_average_b...	total_bac...	word_a...	character_c...	session_time	sessi...	double_min_acceleration_x	word_count	smiley_count	smiley_type_string
393	1,572	0	0	0	0	4	6,594	2	0	1	0	
192.2222900391	576	0	0	0	3	10	10,376	1	-3.3738791942596	3	0	(NULL)
426	1,192	0.035714287310839	0.1	1	0	28	15,851	1	0	10	1	3:1;
266.69564819336	1,226	0	0	0	0	23	13,463	1	0	5	0	
448.14083862305	2,121	0.028169013559818	0.133333333	2	0	71	37,904	1	-0.82174682617188	15	1	3:1;
376.44827270508	1,984	0.033333335071802	0.181818181	2	0	60	31,656	1	-0.29837036132812	11	1	3:1;
271.77777099609	2,446	0.10000000149012	1	1	0	10	8,973	1	0	1	0	
279.25	418	0	0	0	0	12	9,578	1	0.051177978515625	8	0	
333.92306518555	434	0.15384615957737	0.2	2	10	13	6,962	1	0	10	0	(NULL)
1,365	1,365	0	0	0	8	8	2,069	1	0	8	0	(NULL)
1,365	1,365	0	0	0	8	8	2,069	1	0	8	0	(NULL)
516.17645263672	2,925	0	0	0	0	17	9,081	1	0	3	1	3:1;

Figure 4.9: Example data

see that some values are still shown as being (null), this means these values are from earlier sessions that did not implement the data field yet. When the missing field was added to the plug-in, the fields that were not using the field yet would be given the value of (null). When the field was used, but just did not contain any data, it would be empty instead. In the 10th column, one of the accelerometer values is displayed. Some of the acceleration data fields are 0 while some are not, this is because the sensor data unfortunately is not as accurate all the time, or not tracked at all. This could mean that the sensor is not always active when the phone is in use. Other than that the data looks usable at first sight.

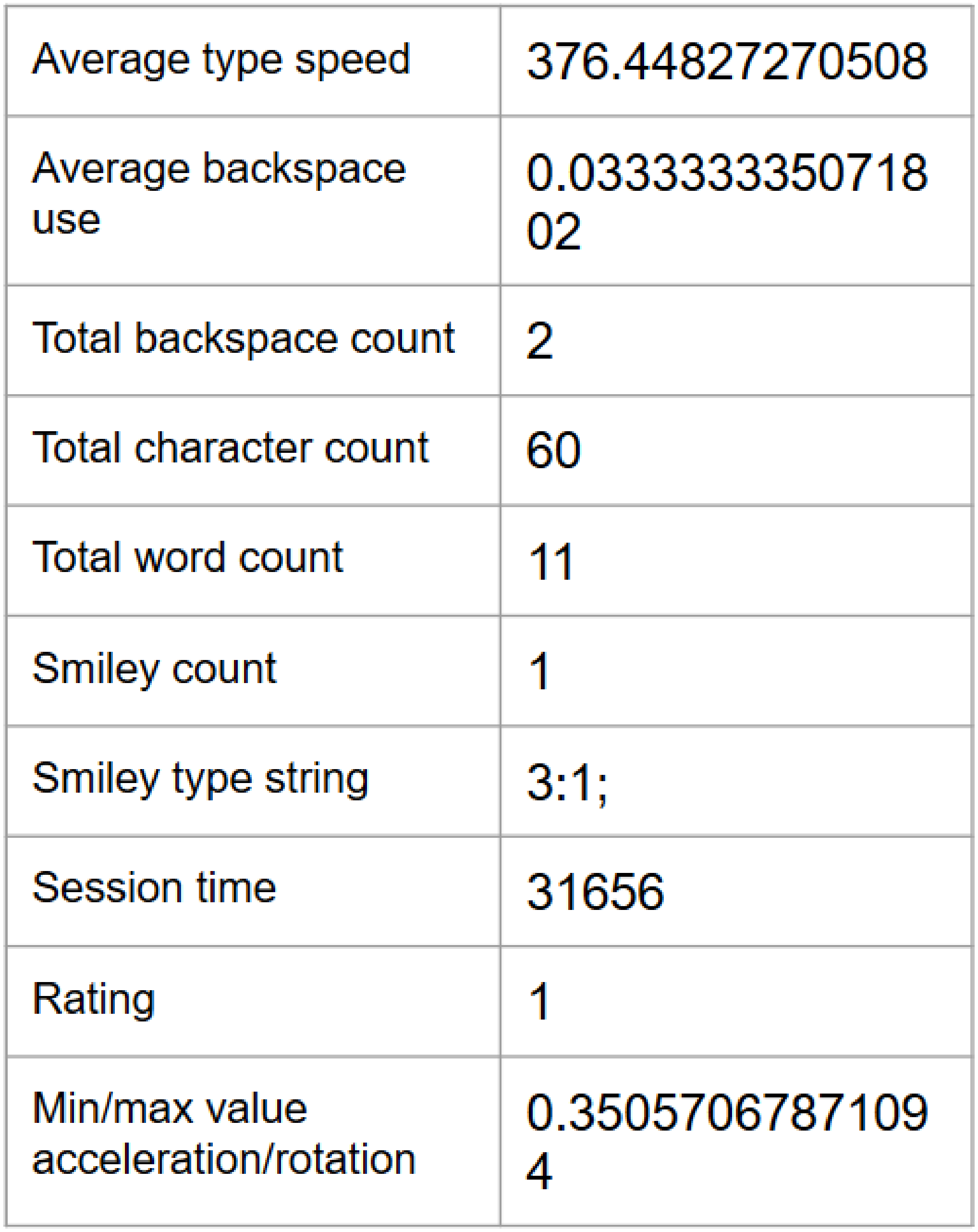
Example typing example: As an example of the data that is coming trough in the session data table, we now show what data was extracted when typing the sentence:

- Hey everyone, this is a message Thatt_t_ will be recorded :p

A smiley is included as well, to show that the smiley data is also collected. The _ characters indicate that a backspace was used while typing. The actual result of typing this character sequence would be:

- Hey everyone, this is a message That will be recorded :p

When the application analyzes this character sequence, the data shown in figure 4.10 can be found in the session data table. In the table only the important data is shown, not all data for each axis is shown because they are not all relevant for the purpose of the example. The most important fields to notice are the backspace count, total character count, the word count, and the smiley count, as these can easily be related to the typed character sequence. As we expect 2 backspaces are tracked, 60 characters including spaces and backspaces, 11 words including the smiley, and the smiley count of 1. A rating is also present, which for the purposed of the example was given the value of 1. As there is one smiley of type 3, the smiley string is defined as "3:1;". Other data was tracked as we wanted it to be. However without actually showing how fast was typed, or how the phone was held, we can only conclude that data for these measures are present, as just the character sequence in the example does not show the times the characters were typed. We can assume that the sentence was typed in application were we spent 31656 milliseconds, and that on average each character was typed 376 milliseconds after each other. Note that the average type speed ignores long pauses, and is thus has a lower value than the session time divided over the character count.



Average type speed	376.44827270508
Average backspace use	0.03333335071802
Total backspace count	2
Total character count	60
Total word count	11
Smiley count	1
Smiley type string	3:1;
Session time	31656
Rating	1
Min/max value acceleration/rotation	0.35057067871094

Figure 4.10: Example analyzed data

Chapter 5

Conclusions and future work

5.1 Summary

In this thesis the development of a tool is described, that collects data, that can lead to the indicators of alcohol intoxication. The project mainly focuses on smartphones, and how their users are typing on the keyboard. While the focus is on keyboard data, other sensors have also been tracked, to see if those could also be an indicator of alcohol intoxication. The main requirement was to have the data be collected in a way that would not be intrusive to the users, and that their privacy was respected.

The project was developed as a plug-in for use with the AWARE application, using the AWARE framework and its sensors. Installation of the plug-in on a smartphone is therefore simple and easy to do. No additional explanation steps are needed when the plug-in is installed and connected to a study. Researchers will have data on their server-side database when users are using their phones to collect data, and analysis can then be carried out on it.

The development of the application was loosely based on the Tapsense[6] paper, in which researchers were able to detect several emotions with typing data. The way this plug-in is developed should help in developing a machine learning model that can conclude similar things as the Tapsense project.

5.2 Discussion

A plug-in to be used with a smartphone was developed for use with the AWARE application, that allows for data collection that will allow researchers to identify indicators of alcohol intoxication. The user's privacy is respected and only meta-data will be collected. This complies with the must haves we defined in our MoSCoW analysis we did when designing the system. As for the should haves, it has been found hard to implement all the features that seemed to be useful. Partly this is because some of the limitations in the AWARE framework, and the complexity of some features. However the acceleration data that was mentioned in the should haves, is implemented and can be used for analysis.

An attempt has been made to also include the could haves. For the could haves, was mentioned that checking on word correctness would be good to have. This feature was partly implemented, but it has been found to be difficult to access Androids device specific dictionaries.

The application has been tested several times, and most of the users found that the latest version of the application is usable. Before the plug-in was in its current state, the ESMs would often pop up too many times, and the users felt interrupted. In the final version this is not the case which makes the tool much more usable. Users did not need any additional information to understand what was going on, as much of the process is done by the application in the background. The average SUS score that was measured was 81,5 which is above average when it comes to the SUS usability scale. Further changes would take a more in depth research of what could be better, instead of just looking at whether the application is usable or not.

As seen in the previous chapter, some people are worried about the privacy the application promises. People were concerned that their data was not hidden enough to be considered private. This is most likely since testing users do not get to see the data that is collected, unless researchers would share this data. Therefore it should be considered to give the user some clear feedback of what data is collected, and which data will go to the researchers.

Another aspect is that some users experienced some performance issues. Especially memory space of the smartphone was mentioned. This is because the application has to track a lot of data from the accelerometer sensors, in order to be able to do something with the data. When using this data, more should be done to prevent data from clogging up to much space on the phones. This would also benefit performance in the sense of speed, because less data will be synced to the server.

The collected data that was stored on the server side is easy to understand for researchers, as the columns of the data are labeled intuitively. The data seems to be consistent enough to conduct analysis on, and data that might be useless is easy to filter out.

Unfortunately for the intended goal of finding indicators of alcohol intoxication, a lot more data needs to be collected. A machine learning model would probably require much more data to conclude anything. And since gathering data about alcohol intoxication requires the users to drink alcohol, it can prove to be hard to do a data collection in a short amount of time. This is because most people would not drink a lot everyday, and people need to be willing to participate for a longer period of time because of it.

The research questions Concluding our research we try to answer the research questions. We discuss what we think the answers to these questions should be. Since we had two questions in support of our primary question, we will discuss those first.

Which tools and sensors on a mobile phone can be used to indicate alcohol use?

When answering this question, we must also question the relevance of the sensors. In this project we focused on keyboard usage, but we also tracked acceleration sensors. Since our data set is relatively limited, and the changes are relatively small, it is really hard to see if data was useful or not. To really conclude this question, more data, and machine learning models should be used to find the subtle differences. To add to this, most users felt that detecting intoxication through data collection like this application does, it should be possible to find indicators through typing. However, the acceleration data does not seem to add anything extra on top of this data yet, and the collection of this data could potentially even hurt user experience and usability. So to conclude we can state that keyboard analysis will most likely be the first sensor that should be looked at when finding indicators. Other sensors might still be useful, but we can not confirm or reject these yet.

The second question was:

Can we formulate a metric to indicate a certain level of intoxication?

At the time of formulating our project, we were confident that we could much more easily define a metric for alcohol intoxication. Unfortunately in the time dedicated to this project, we have not found a clear metric for it. Again machine learning should take over from this project to know more about the possibilities in finding this metric.

This brings us to the primary research question:

Can we identify in a non-intrusive way if a person is intoxicated by alcohol, with the use of smart phone technology?

Combining the supporting questions and the conducted researches into an answer to this question, we should break the question down into 3 parts.

The first part being can we identify if a person is intoxicated by alcohol. The second part being if we can use smartphones for it. And the third part, if we can do this in a non-intrusive way.

The developed plug-in is able to collect data, that might be an indicator of alcohol intoxication. Although we can not confirm the data to indicate alcohol intoxication yet, we can confirm that the smart phone can be used as a tool for data collection for this purpose. We also showed that most users found the application usable, and in the latest developed version was not intensely intrusive. So looking at this question as a whole, we have to answer it with maybe. If we look at the individual parts we can say that the only part we did not succeed in, was to identify alcohol intoxication.

To conclude this discussion we can say that this project lays a foundation for detecting alcohol intoxication using smartphones. Hopefully other researchers will be inspired by this approach as well and take it one step beyond.

5.3 Future work

As for future work we hope other people will pick up on this relatively new approach of detecting certain states of the mind and its emotions. When taking up a challenge in how data can be collected, this thesis should give some insights in the possibilities and the usability. When given a longer amount of time, more data can easily be collected. Future projects should include more incentive to use the application, and provide more insight in the use and collection of data. Feedback of the latest added data entries should be displayed to the user, so they can judge whether or not they feel good about the data that is collected. Also when data can eventually well if a user is intoxicated, it would be nice to have the user see if they are intoxicated or not.

As mentioned before, on a bigger dataset, a machine learning model should be implemented to have data interpreted by the system. When the system is able to do this, it can be used for multiple applications that can benefit from detection of alcohol intoxication. When we mention the feedback ESMs that were used in the development of this project, we can also state that with a slight change of the question that will be asked, it should also be possible to investigate other areas besides alcohol intoxication. When a user has to answer other questions with a scale, the data will most likely tell more about the specific thing mentioned in the feedback question.

When it has become more clear what data is relevant for determining certain states, optimizations should be made to prevent the smartphones having to hold too much data. The system can still improve in efficiency, so when developing further on this application, people should try to optimize data collection and its speed.

5.3.1 Practical applications

When it comes to the practical use of a system that can identify alcohol intoxication, we can see it used in several applications. In which some of the more impact full could be, rehabilitation reflection and DUI prevention. With the rehabilitation reflection we mean to aid the problem stated in the background section 1.1. There we mention that when people come from rehabilitation facilities for alcohol intoxication, the checkups preformed are usually not very effective. When the application can notify the facility, based on what it unobtrusively noticed about the patient, the facility can undertake actions more effectively to get their ex patient rehabilitated again.

For drunk driving a similar thing can be done. Some people are often convinced they are still capable of driving while in reality they still can not. When an application can notify people when he or she is about to drive while being intoxicated, actions can be taken to make sure the person is not going to drive. This could even mean that the car automatically will not start when the driver is drunk.

Besides these applications probably dozens of applications can be thought of when this technology is in existence. Of course when thinking of the applications for an application like that, we should always be weary of the ethical and moral aspects of it. Technology should in my opinion never be used in a way that, can hurt or damage other people in any way. With this I would like to conclude the thesis, and we would like to thank you for reading it.

Bibliography

[1] moderatedrinking-03 @ pubs.niaaa.nih.gov.

[2] Joseph M. Boden and David M. Fergusson. Alcohol and depression. *Addiction*, 106(5):906–914, 5 2011.

[3] a Buchoux and N L Clarke. Deployment of keystroke analysis on a Smartphone. *Proceedings of 6th Australian Information Security Management Conference*, (December 2006):29–39, 2008.

[4] Heather Cole-Lewis and Trace Kershaw. Text Messaging as a Tool for Behavior Change in Disease Prevention and Management. *Epidemiol Reviews*, 32(1):56–69, 2010.

[5] J Dai, J Teng, and X Bai. Mobile phone based drunk driving detection. *. . .), 2010 4th International . . .*, page 1–8, 2010.

[6] Surjya Ghosh, Niloy Ganguly, Bivas Mitra, and Pradipta De. TapSense: Combining Self-Report Patterns and Typing Characteristics for Smartphone based Emotion Detection. *Proceedings of the 19th International Conference on Human-Computer Interaction with Mobile Devices and Services - MobileHCI '17*, pages 1–12, 2017.

[7] National Institute on Alcohol Abuse. Alcohol Facts and Statistics Fact Sheet. 2017.

[8] A. Kolakowska. A review of emotion recognition methods based on keystroke dynamics and mouse movements. *2013 6th International Conference on Human System Interactions, HSI 2013*, pages 548–555, 2013.

[9] Hosub Lee, Young Sang Choi, Sunjae Lee, and I. P. Park. Towards unobtrusive emotion recognition for affective social communication. *2012 IEEE Consumer Communications and Networking Conference, CCNC'2012*, pages 260–264, 2012.

[10] Sachin Moonat, Subhash C Pandey, and D Ph. and Alcoholism. 34(4):495–505, 2009.

[11] The Lancet. Alcohol misuse needs a global response. *The Lancet*, 373(9662):433, 2009.

[12] Rachel Wittenauer, Lily Smith, and Kamal Aden. Priority Medicines for Europe and the World " A Public Health Approach to Innovation " Update on 2004 Background Paper Background Paper 6 . 12 Osteoarthritis. (June):1–31, 2013.