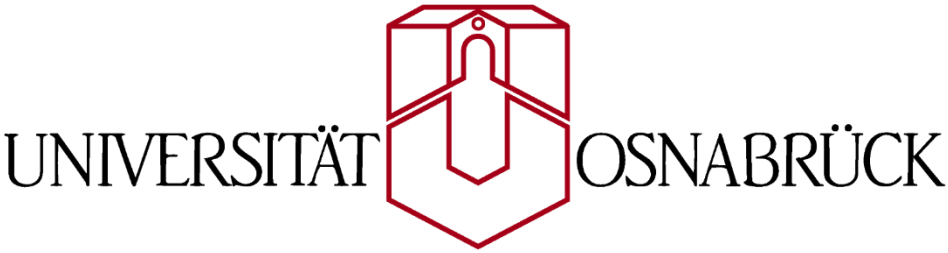


INSTITUTE OF COGNITIVE SCIENCE

Bachelor's Thesis

Developing and designing a mediator smartphone application for portable sleep EEG recording and biofeedback with focus on the end user experience

Author:
Martin Jäkel

First supervisor: Kristoffer Appel, M. Sc.

Second supervisor: Prof. Dr. Gordon Pipa

February 2017

Abstract

Developing and designing a mediator smartphone application for portable sleep EEG recording and biofeedback with focus on the end user experience

The Traumschreiber project at the University of Osnabrück by Kristoffer Appel, Johannes Leugering and Prof. Dr. Gordon Pipa has the goal to develop a sleep mask for research purposes to enable cheap, easy and large scale sleep experiments. The mask gathers electroencephalography (EEG) data and also is going to provide the means to influence human sleep and dream through visual and auditory stimuli.
One objective of this project is to conduct experiments at home without the need for research staff to be present. To allow layman this household use and control of the sleep mask and to manage and process the generated sleep data, the popularity of modern smartphones gave rise to the idea of an Android application to handle these matters.
In this thesis the concept and blueprint for this Android application is described and important key features are explained. On the basis of these ideas a prototype application was designed and implemented. The specific parts of this process are discussed and documented, including the user interface design with focus on the end user experience and the application implementation along with handling the connectivity via Bluetooth Low Energy (BLE).

Contents

List of Figures

Figures and Images that are original content are labeled with "(OC)".

List of Source Code Listings

Code listings are application excerpts unless stated otherwise.

List of Tables

List of Abbreviations

APP: Application

APK: Android application package

API: Application programming interface

BLE: Bluetooth low energy

ECG: Electrocardiography

EEG: Electroencephalography

EOG: Electrooculography

EMG: Electromyography

GATT: Generic Attribute Profile

GUI: graphical user interface

IoT: Internet of Things

IDE: Integrated development environment

OS: Operating system

SDK: Software Development Kit

SoC: System-on-a-Chip

UUID: Universally Unique Identifier

XML: Extensible Markup Language

Acknowledgments

First of all, I want to thank Kristoffer Appel for supervising, motivating and guiding this thesis.

Moreover, I want to thank every member of the ALP group: Kristoffer Appel, Johannes Leugering and Prof. Dr. Gordon Pipa for additional support and for creating the possibility to write this thesis.

Furthermore, a special thank you goes to my parents, Patrick Faion, Annika Schwitteck and Britta Grusdt.

1 Introduction

This chapter will briefly introduce sleep and dream research. In respect to this research, it will explain why the Traumschreiber project **[1]** was started and summarize the idea and desire behind it. Then the motivation for this thesis and the basic concept for the Android **[2]** application (app) is described. Furthermore, an overview about existing and similar products will be given and discussed. Lastly, a brief outline of this thesis is provided.

1.1 Sleep and Dream Research

The average human spends about one third of his life in a sleeping and thus at times dreaming state. Therefore, one could argue that sleeping and dreaming is a vital part of every human's life and effects our healthiness, e.g. through the negative effects that sleep deprivation can bear **[3]**.
Conducting and innovating basic research in this field is desirable and also is in my personal interest. But performing these sleep experiments to generate valuable data for example brain activity, recorded via electroencephalography or electrooculography (EOG) does usually require extensive procedures **[4]**. In most of all basic research outside the clinical and health care context, it can be difficult and time consuming to find study participants, since they usually have to attend in their spare time and are only paid a small compensation fee. This holds especially true for sleep and dream research. Unless there is personal interest, the threshold to participate can be even higher. Subjects are usually required to spend their night sleep away from home inside a sleep laboratory wired to different recording devices. These sleep laboratories are mostly expensive to set up and furthermore, research staff need to be present to monitor and oversee these neuropsychological studies.

1.2 The Traumschreiber Project

The Traumschreiber project was started with the intention to build a novel kind of sleeping mask for research purposes. The **ALP** group at the University of Osnabrück, consisting of Kristoffer **A**ppel, Johannes **L**eugering and Prof. Dr. Gordon **P**ipa, are the founders of the project. It is the project's goal to provide an open source hard- and software system that allows for a new, innovative and also easier way of conducting sleep experiments. As highest priority, the sleeping mask is supposed to be a fully mobile, cheap and also easy-to-use system. These features enable the conduction of sleep experiments outside the typical sleep laboratory environment and record data at home, leaving the participants inside more natural surroundings. The participants should be capable of operating the mask and setting up given experiments themselves with no research staff present. This "take home" system that minimizes resources in terms of necessary staff and hardware expenses, grants the possibility of conducting large-scale sleep studies that simultaneously gather data from several subjects. The sleep mask is supposed to record sleep data, such as brain activity, via EEG or EOG electrodes that connect to the mask. This data is wirelessly transferred to a peripheral (**Figure 1**) such as a smartphone or PC via Bluetooth Low energy and can then be transmitted to external servers for further analysis and research.

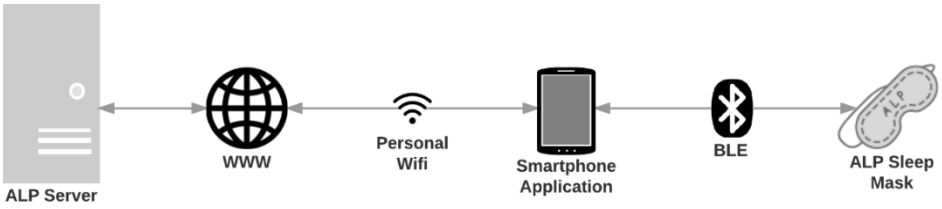


Figure 1: Traumschreiber connections (OC)

Additionally, the mask will integrate mechanisms for sensory stimulation and biofeedback. LED lights and speakers that connect to the sleep mask can pass signals to the sleeping individual and thus influence and manipulate their dreams and sleep. Among the possible applications are lucid dreaming[1] experiments. With the help of subtle sensory cues such as blinking LEDs or sounds that are emitted during a phase in which the participant is dreaming, they should get aware and conscious of their own dreaming state without

[1] For a more in depth explanation about Lucid Dreaming visit: *world-of-lucid-dreaming.com*

waking up. These additional features are included with the motivation to extend the range of possible experiments that can be conducted with this system to innovate and push research in this field.

1.3 Motivation and Goal of this Thesis

The Traumschreiber project is still being developed. One important component is the mentioned peripheral that wirelessly controls the sleep mask and receives the recorded data. The extensive availability and capabilities of modern smartphones make them the go to choice for this task. Smartphones can furthermore provide a graphical user interface that makes controlling the Traumschreiber easy and understandable to the home user. This gave the motivation to design and build a suitable application, which was part of this thesis. With a great experience for the end user in mind, this application should handle the Traumschreiber data, introduce the system to participants and help them use it.
The goal of this thesis is to describe the implemented Android application as well as document and discuss its development process and results. Additionally, by contributing with this to the Traumschreiber project, this thesis provided the opportunity to get a deeper insight into smartphone application development based on a real word project and actual applied research.

1.4 Alternatives

In preparation to designing and building the Traumschreiber app, a brief investigation of projects similar to the Traumschreiber sleep mask and mobile applications with related functionality was conducted. Online research about measuring sleep activity and tracking vital activity led to the so-called quantified-self movement[2], a community that is interested in tracking and recording data about their own body to learn and improve one's own health. Members use a variety of tools, gadgets and apps to achieve this goal. Very popular are sleep or fitness tracking wrist bands that connect to a smartphone application or other peripherals and include sensors to record heart rate activity and body movement. One prominent example is the

[2] The German quantified-self movement can be found at: *qsdeutschland.de*

Jawbone fitness tracker “UP3”, it specifically advertises its sleep tracking capabilities **[5]**. With respect to sleep tracking these devices can use the mentioned data to approximate and predict sleep stages and the level of wakefulness. Additionally, products like the sleep masks from *urbandroid*[3] or *remee*[4] are available. These sleep masks include eye facing LED’s that are supposed to “increases dream recall and vividness”[4] and support the “stage for lucid dreaming”[4].
None of these products however allow for deeper inspection of brain activity on a neurological level, and thus are not suited for the purpose of the Traumschreiber. The possibility to record brain activity via EEG and the combination of planned Traumschreiber features allow to conduct more precise and a way broader spectrum of scientific studies.

1.5 Thesis Outline

After this introductory chapter, chapter 2 will describe important technologies and key hardware components related to the application.
Chapter 3 discusses the application development process including the creation of first concepts, design and implementation details and also comments on used tools and methods. Application deployment and testing are referred to in chapter 4. An outlook and new ideas are given in chapter 5 and a final summary and conclusion is formed in chapter 6.

[3] *http://sleepmask.urbandroid.org*
[4] *https://sleepwithremee.com/index.html*

2 Technologies and Hardware

This chapter describes the involved hard- and software in more depth as well as explains some of the key technologies used in the app development process. First the Traumschreiber sleep mask as well as its circuit board is portrayed and the Bluetooth Low Energy (BLE) technology is outlined.
Then the Android operating system (OS) is introduced, including key concepts and main building blocks like *Layouts*, *Activities*, *Fragments* as well as *Services* and *Intents*.

2.1 The Traumschreiber Sleep Mask

The Traumschreiber project is still ongoing and a new iteration of the sleep mask is currently in development[5] but the important key features and components are likely to stay the same. Version 1.0, as shown in **Figure 3**, depicts the custom knitted fabric mask. On the one hand, this mask encloses built-in sensory stimulators, namely a set of speakers (in future versions) and eye facing LEDs which can be used for sleep manipulation and biofeedback mechanisms. On the other hand, the mask also includes the main control unit. This custom designed circuit board connects to these stimulators and to electrodes which record the subjects brain activity. Traumschreiber chip version 2.4 (**Figure 4**), has a power consumption of 3.7V, supports 8 Channels @ 250Hz, usable for EEG, EMG, ECG or EOG recording, includes amplification as well as analog high and low pass filters (0.04Hz, 72Hz) to process revived data already onboard. **Figure 2** shows some original example sleep data recorded with the Traumschreiber.

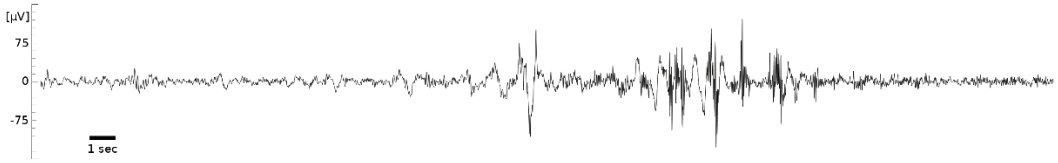


Figure 2: Traumschreiber example EEG data (traumschreiber.uni-osnabrueck.de, 2017)

Another main component of the control board is the Bluetooth Low Energy System on a Chip (SoC). A BLE chip that allows the sleep mask to

[5] Updates on newer versions will be available at *traumschreiber.uni-osnabrueck.de*

communicate and broadcast its recorded data as a so-called BLE peripheral wirelessly and with minimal power consumption to another BLE capable device such as a smartphone or desktop computer.

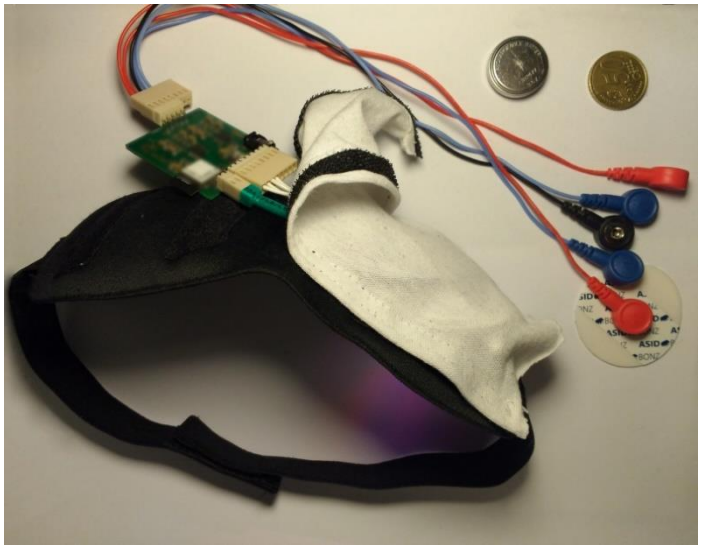


Figure 3: Traumschreiber sleep mask version 1.0 (traumschreiber.uni-osnabrueck.de, 2017)

Figure 4: Traumschreiber chip v.2.4, (media.ccc.de/v/MRMCD16-7762-traumschreiber, 2017)

2.1.1 Bluetooth Low Energy

Bluetooth is a standardized technology (IEEE 802.15)[6] much like WLAN or NFC, which allows devices to wirelessly transfer data and exchange meta information with other Bluetooth capable devices. In recent years this technology gained a lot of popularity because the market around the Internet of Things (IoT) started to emerge and Smartphones became even more powerful and popular. IoT in simple words describes the concept that everyday objects or devices in the physical world, for example a fridge or a light bulb get connected to the World Wide Web and thus can transfer information, which extends their capabilities[7]. The relatively new (2010) Bluetooth Low Energy (BLE) standard was especially developed for the IoT. In contrast to classical Bluetooth it requires significantly less power, which allows devices to be truly wireless and less maintenance intensive. This lower power consumption comes with the cost of an overall lower data throughput rate of 0.27 Mbit/s compared to 0.7 Mbit/s[8]. Nevertheless, with enough range, very low power consumption and 128-bit AES data encryption BLE has the best feature package to handle wireless data transfer for the Traumschreiber project.

When BLE devices interact with each other they can be separated into two different roles, see **Figure 5**.

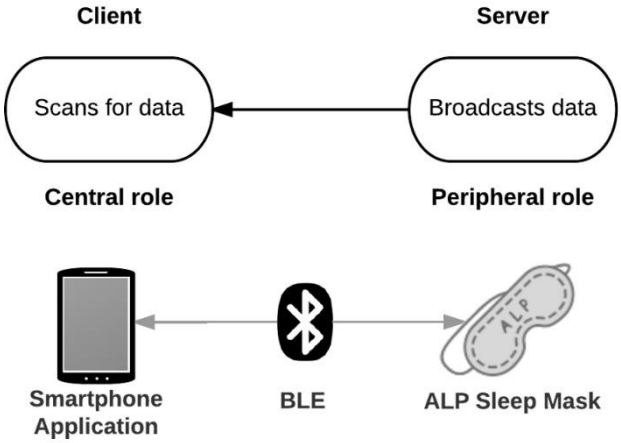


Figure 5: BLE Roles (OC)

On the one hand, they can take the peripheral role and act as a BLE server that broadcasts data. In relation to this thesis, the sleep mask, which sends the recorded brain activity over the air, takes this role. On the other hand, BLE devices can take the so-called central role. For example, an Android app that scans for available BLE services takes this role and acts as the BLE client.

The actual data transfer from device to device is defined by the so-called Generic Attribute Profile or short GATT (**Figure 6)**, which is built on top of the Attribute Protocol (ATT) **[6]**. It specifies how data is revived and send in a structural and hierarchical manner. Every profile consists of several attributes that are grouped and encapsulated further.

[6] More information about IEEE standards: *ieee.org/standards*
[7] For more detailed information about the IoT visit: *iot.ieee.org/definition*
[8] For more technical information, visit: *bluetooth.com*

Every attribute itself can be identified via a so-called Universally Unique Identifier (UUID), this 128-bit number is supposed to be globally unique. A BLE client in the central role can scan for GATT services. Every service bundles a set of characteristics, which in turn encapsulates further attributes and their descriptors. These attributes then hold a single value that describes the characteristics data, for example a sensor reading. Descriptors are simply human readable attributes that describe the value the characteristic holds e.g. "*light sensor*". Once a service or characteristic is discovered the client can either read a characteristic's values via their UUIDs either only once, or has, dependent on the attribute, the possibility to subscribe to these characteristics and get notified automatically if their values change.

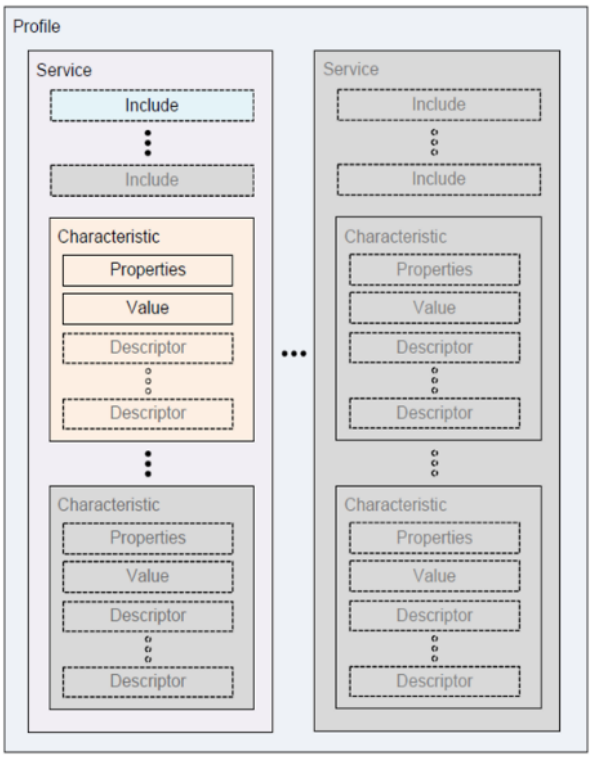


Figure 6: GATT profile hierarchy (Bluetooth.com, 2017)

2.2 The Android Platform

Android is a mobile operating system, mostly used on modern Smartphones and Tablets. It was developed by Google, with its first commercial appearance in 2008 **[7]**. In 2016, the most recent version 7.1.1 codename "Nougat" was released. The Android OS as well as Android applications are mainly developed in the Java programming language and additional XML. But among others C and C++ for performance dependent development are supported as well. The Android operating system is open source. With this philosophy, content creators get the chance to develop and innovate with minimal restrictions[9]. For this purpose, a Software Development Kit (SDK) (current version 25) but also several Application programming interfaces (APIs) are provided to help developers around the globe.

[9] To find out more about the Android open philosophy visit: *source.android.com/source*

Furthermore, the Android OS is the current and dominant market leader in terms of mobile operating systems used with 87.5% **[8]** and over 1.4 billion active users **[9]**. Because of its open source philosophy and its market positon it is the go to platform for the Traumschreiber project. As comparison, the next best option, the Apple iOS operating system holds 12.1% **[8]** marked share. It furthermore limits its content creators by forcing them to use an apple computer for development, whereas the Java programming language is platform independent and thus Android development can be done on all platforms. One disadvantage, the Android OS has in comparison to iOS, is a more complex development process, caused by the need to adjust and optimize the app separately for a larger variety of devices and OS versions, but this is most often outweighed by the described advantages.

2.2.1 Extensible Markup Language

The Extensible Markup Language (short form: XML) is a simple and also human readable markup language that structures information in a hierarchical and tree like manner[10]. In the Android development process, XML is mainly used to define graphical user interface (GUI) resources and their relations. For example, colors, menus and animations are defined in XML. In order to structure and design the way the app is presented on the smartphone screen, XML elements, so-called *views*, encapsulated in *viewgroups*, are defined. For an excerpt example XML layout see **Listing 1**.
Every GUI element, as for example a simple `BUTTON` or just plain, text is represented as a view. The relation between several of these views are often defined by parent viewgroups and their attributes. Common android viewgroups are the `LinearLayout` and `RelativeLayout`, these typically define how nested child views are arranged. Every view can furthermore hold a great amount of different attributes that describe its look and behavior. Views can hold an ID attribute "`android:id="@+id/button"`" that uniquely identifies this view throughout the application. This ID can be used to later interact and change the GUI appearance from within a Java class. If a view, as for example a button holds the `onClick` attribute, logic inside the Java code can be executed on user interaction. Although it is possible to define every GUI element in pure Java, it is often times handier and more practical to use XML. The appearance of one particular user screen is most of the time predefined inside one corresponding and specific resource file.

[10] More specific information about XML itself can be found at: *w3.org/XML/*

The first time this screen is shown to the user or is interacted with, the XML resource file is "*inflated*". This means its views and viewgroups are parsed into Java objects and the layout is rendered.

```
<RelativeLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    android:layout_margin="10dp">

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button"
        android:id="@+id/button"
        android:onClick="buttonLogic"/>

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="18dp"
        android:textColor="@color/colorAccent"
        android:text="Hello World"
        android:layout_below="@+id/button"/>

</RelativeLayout>
```

Listing 1: XML layout example

2.2.1.2 The Manifest File

Besides defining GUI resources, XML is also used to define the manifest file, which takes a central role in every android app. This file is used to declare permissions, such as BLE, internet or location services the application wants to access. The user has to grant these permissions on first use or on installation, depending on the OS version. It also describes the applications hard- and software requirements, as for example the minimum Android version to restrict installation to a given device range. Most importantly though, all main application components, such as activities and services, including some of their attributes, are defined here[11].

[11] For a more detailed look at what the Manifest File does see: *developer.android.com/guide/topics/manifest/manifest-intro*

2.2.2 Activities and Fragments

One of the main building blocks of every Android application is the `Activity`, a Java class that usually corresponds to one screen. This Java class handles the functionalities of every GUI element it holds, manages user input and is also responsible for navigation between screens. On application start up, one particular activity is chosen as entry point or launcher activity (declared in the Manifest). From this point, further application and activity behavior is described in the activity lifecycle seen in **Figure 7**. While an activity is loaded for the first time, the `onCreate()` method **Listing 2** gets executed. Here the XML layout resource file is inflated via `setContentView(R.id.layoutfile)` and for example `clickListeners` are bound to their GUI elements.

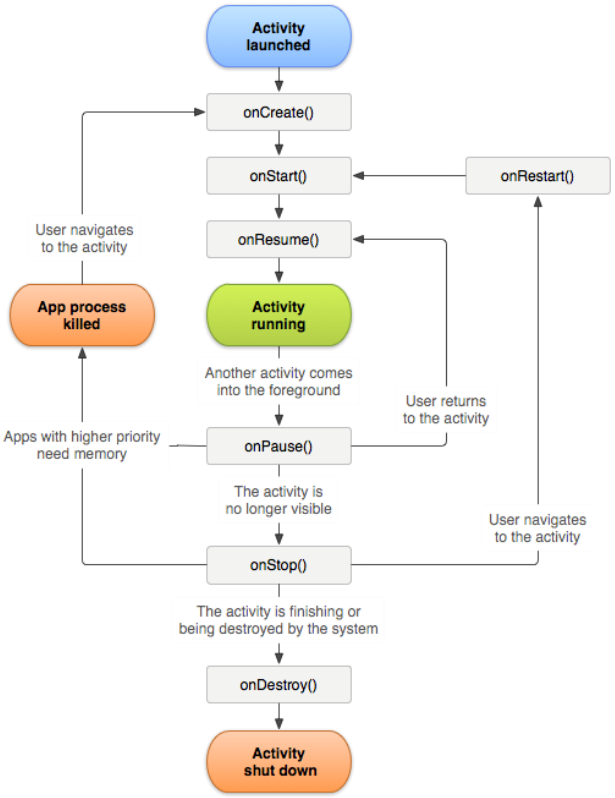


Figure 7: The Activity lifecycle (developer.android.com, 2017)

For every state the application and thus the activity can be in, or moves through, certain behavior can be implemented inside its corresponding method. If the user navigates to a different activity and the current activity is no longer visible, `onPause()` and on return `onResume()` is triggered.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setContentView(R.layout.activity_demo);
    ListView listView = (ListView) findViewById(R.id.deviceList);
    listView.setAdapter();
    listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
        @Override
        public void onItemClick(AdapterView<?> parent, View view, int
position, long id) {
        }
    });
}
```

Listing 2: onCreate() inside an Activity

One exception to the "one activity on screen" rule is the `Fragment` class. A fragment can be thought of as a view inside an activity that is dynamically interchangeable and can take up the complete screen. In contrast to a simple view the fragment can handle its own user inputs and callbacks and possesses its own lifecycle, which is aligned to its containing parent activity, depicted in **Figure 8**. During runtime, a Fragment gets attached to an activity, here its XML resources are nested into the activity's layout structure and the `onAttatch()` and `onCreate()` methods are called accordingly. Fragments can be very handy if layout designs and logic need to be reused or the screen needs to change dynamically for better user experience. A common example is to create user-friendly navigation between application screens by swiping fragments but internally staying inside the same activity.

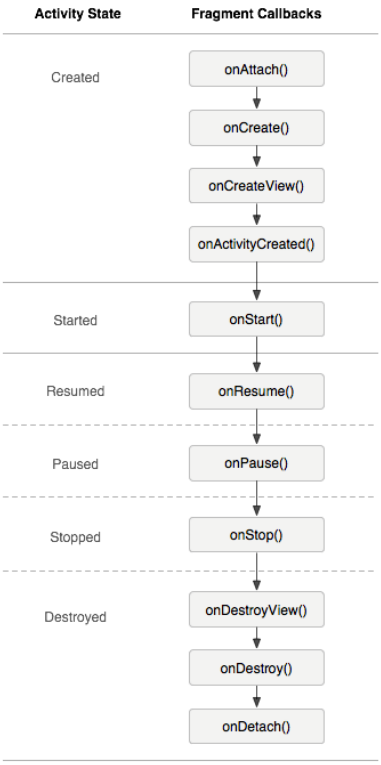


Figure 8: Fragment lifecycle (developer.android.com, 2017)

2.2.3 Intents

In simple terms, `Intents` are the vehicles inside the Android OS and its applications. Independent from object oriented programming paradigms they send information from one app component to another. Typically, if a user wants to navigate between screens and thus activities, for example by button press or the menu use, the developer has to make use of intents. Furthermore, intents are capable of a lot more: they can wait for a result from another activity or transfer data between app components. Most of the time explicit intents are used, these are interactions between internal components from within the same application. Here the intent explicitly calls another component, e.g. a service or an activity by its name. The Android platform also allows for implicit intents. That is, to target functionality outside of the current application without knowing what kind of application will be used. See **Listing 3** for an explicit and implicit intent example. This could be sending an email or opening a website.
Applications can register their purpose inside the android system by declaring intent functionally inside the manifest file, this way the user can

choose between available applications that support the requested feature, if an implicit intent is received.

```
public void showAboutActivity(MenuItem mi) {
    Intent intent = new Intent(this, AboutActivity.class);
    startActivity(intent);
}

public void showSurvey(MenuItem menuItem) {
    Intent i = new Intent(Intent.ACTION_VIEW, Uri.parse(surveyUri));
    startActivity(i);
}
```

Listing 3: User navigation via Intents

2.2.4 Services

One last important building block of Android applications, which is relevant for this thesis are `Services`. Services, unlike activities, do not provide a graphical user interface.
In contrast to activities, which pause or destroy themselves if the application is minimized or closed, services are able to run in the background of the Android OS as long as needed. This is typically necessary if the application listens for incoming information such as a messaging app or bigger amounts of data are transmitted or synchronized. Services again bring their own lifecycle depicted in **Figure 9**. Services can be started using implicit or explicit intents and run until unbound or destroyed.

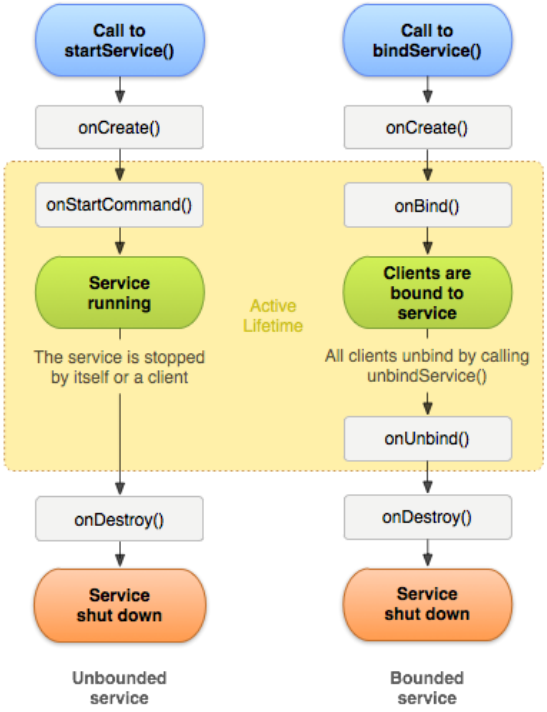


Figure 9: The Service lifecycle (developer.android.com, 2017)

3 Application Development

In this chapter the application development will be inspected in more detail. First it will be described what kind of conceptual considerations were made and which features where planned prior to technical realization.
Tools and methods that aided the work on the application are reviewed.
Moreover, the design and implementation process of specific application elements are examined in depth. Furthermore, challenges that occurred during development are described.

3.1 Application Concept & Idea

When this thesis was planned and agreed upon, one functional version of the Traumschreiber system including all components as seen in **Figure 10** was already usable.

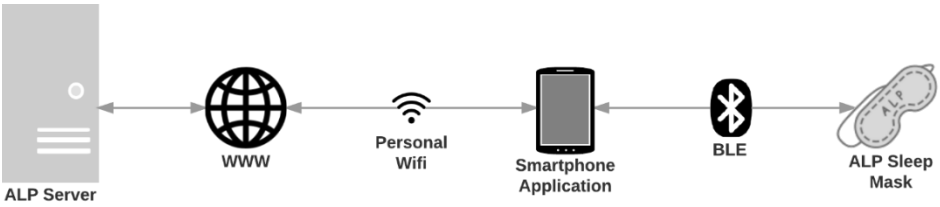


Figure 10: The Traumschreiber system (OC)

The general concept for the mobile application, which acts as hub or mediator between sleep mask and end user, was already explained in the introduction to this thesis. This first iteration included the smartphone application named “Traumdeuter” developed by Johannes Leugering for the Android system. It consisted of a basic user interface and was able to connect to the sleeping mask via BLE and forward the recorded data to a University server that was running a python script for further data visualization and analysis.
A new Android app with a broader feature set and improved design and user experience was desired. Searching for a novel application name that was catchy and also language neutral, first led to “Somnium EEG” and ended up to be just “Somnium” (Latin for "The Dream"). Brainstorming and discussions in meetings then led to a list of features for the new Android app.

3.1.1 Desired Application Features

The following list of features was conceived and a similar one was written down in the thesis exposé in a relatively early stage of the project. It states desirable features, that a ready-to-release Traumschreiber application could have at a later stage and that should be worked towards.

In terms of connectivity features, the app should be capable of:

- Receiving EEG/EOG data from the Traumschreiber and connecting to it via Bluetooth low energy (BLE) in an effective manner
- Triggering the built-in sensory stimulants (LEDs)
- Connecting to an external server and transmitting Traumschreiber and user data between smartphone and server in a secure way, possibly by using secure websockets
- Providing a data buffer for potential connection problems and time outs prevention

In terms of user interface and user interaction, desired features are to:

- Provide the end user with a clearly structured and easy-to-use application that can control the sleep mask
- Grant the user sufficient means to configure his application, set his user data and test his Traumschreiber mask
- Include a short and comprehensible starter guide that explains the app and how to use the sleep mask properly
- Display live or processed data to the user, which was recorded by the Traumschreiber. This could include live EEG or statistics about sleep phases
- Make it possible to export or save recorded sleep data
- Collect additional user data with questionnaires after sleep experiments
- Provide a built-in alarm clock that syncs up with planned sleep experiments

During the course of this thesis, it became clear that implementing and achieving all of those features might be improbable. This was due to the sheer amount and complexity of features desired but also due to constant hardware shortage and changes concerning the sleep mask circuit board. An outlook regarding these missing features will be given and discussed in chapter 5.

3.2 Tools and Methods

Apart from searching the Internet, the official android documentation**[10]** and *stackoverflow.com*, more resources have been used and helped during the development process. This section will describe the software used for development, list the libraries included in the app and comment on the Bluetooth Low Energy device used as sleep mask substitution.

3.2.1 Android Studio

Somnium was developed with Google's official integrated development environment (IDE) for Android development. The Android Studio[12] IDE currently in version 2.2.3, based on the IntelliJ[13] IDE, is freely available for all platforms including Mac OS, PC and Linux. In **Figure 11** the usual project structure inside Android Studio is depicted. Applications are separated into Java files such as activities or services and resource files, which are XML layouts, as well as images or text files, which have all been described in chapter two. Android Studio has many features, for example it allows to preview XML layouts on different screen sizes, it integrates version control management such as git[14] and also provides the possibility to fully emulate Android devices[15] to test and debug apps in a fast manner. Overall Android Studio is a powerful tool that was really helpful during the development process of the Traumschreiber app.

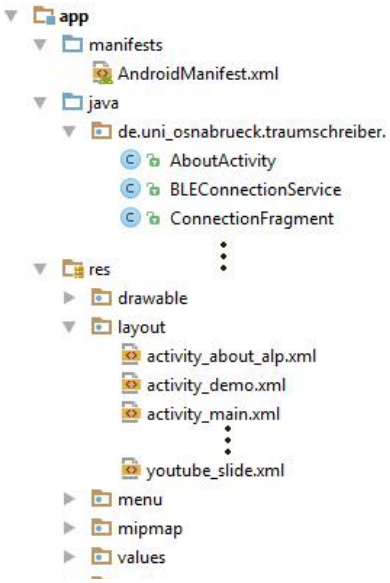


Figure 11: Android Studio project structure (OC)

[12] *developer.android.com/studio*
[13] *jetbrains.com/idea*
[14] *git-scm.com*
[15] *developer.android.com/studio/run/emulator*

3.2.2 Libraries and APIs

Android Studio, by default, uses the Gradle[16] build system, it compiles the source code and packages it, including the necessary resources.
It is common to include external source code in form of code libraries or application programming interfaces. These external `dependencies` are declared in the `build.gradle` file. Libraries are often added to an Android application if a certain functionality that is desired, is not unique to this application and was already developed and published by other Android engineers to be reused. On good example is the YouTube Android Player API, it provides code functionality to integrate a video player inside your Android application that is able to play online videos from the Youtube video platform. Instead of implementing complex video playing features again, this API provides the means to do it in a faster and easy manner.
Excluding native Android libraries, the Somnium app has the following dependencies:

- YouTube Android Player API[17]

  This API enables to integrate Youtube videos as already explained.
  It requires the mobile YouTube app to be installed on the given device to function properly.

- MP Android Chart[18]

  A library that provides chart views and Java classes to integrate plots and data visualization into application layouts, all in a simple and visually appealing way.

- Paolorotolo AppIntro[19]

  The AppIntro library supports you to create a visually appealing introduction to your Android application that complies with Google's design guidelines. Users can swipe through fragments that contain useful information about your app and its usage.

---

[16] To learn more about the Gradle build system visit:
docs.gradle.org/current/userguide/userguide
[17] *developers.google.com/youtube/android/player*
[18] *github.com/PhilJay/MPAndroidChart*
[19] *github.com/apl-devs/AppIntro*

3.2.3 BLE Hardware Development Kit

Developing and implementing the Bluetooth Low Energy connectivity for Android devices requires a BLE peripheral to test data transfer and connectivity.
The *Wicked Sense 2*[20] development kit (**Figure 12**) was used to simulate the Traumschreiber mask as a BLE peripheral. It includes a BLE SoC, as well as several sensors and was directly developed for the IoT market[21]. Using the Wicked Smart IDE[22] the development kit was configured to emulate a BLE Hear rate monitor, that broadcasts mock data, allows for direct connections and notifies connected devices about characteristic changes that they have subscribed to.

Figure 12: Wicked Sense 2 BLE development kit (cypress.com, 2017)

[20] *cypress.com/documentation/development-kitsboards/bcm9wicedsense2-evaluation-and-development-kit*
[21] *http://www.cypress.com/internet-things-iot*
[22] *http://www.cypress.com/products/wiced-software*

3.3 Application Design and Implementation

During the process of setting up a new Android application project, it is required to specify the minimum API level, i.e. the minimum Android Version the application is supposed to run on. With every new API the Android operating system is being improved and new features are added. For example, support for Bluetooth Low Energy was first added with API level 18 **[11]**. With that in mind, it is important to carefully balance the chosen API level between: backwards compatibility, feature requirements, being future proof and most importantly the distribution of Android OS versions on the market (**Figure 13,** state May 2,2016). With these considerations in mind, API level 22 codename "Lollipop" was chosen as minimum API, when the implementation process for the Somnium app started. Although in mid-2016 only 26,9% of all circulating Android devices used this API level, the distribution changed rapidly. Today (state January 9, 2017), already 53,6% are using API level 22 **[12]**. Furthermore, Android codename Lollipop brought improvements and updates to the Bluetooth Low Energy API **[13]**, thus making it the most future proof and reasonable choice.

Version	Codename	API	Distribution
2.2	Froyo	8	0.1%
2.3.3 - 2.3.7	Gingerbread	10	2.2%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	2.0%
4.1.x	Jelly Bean	16	7.2%
4.2.x		17	10.0%
4.3		18	2.9%
4.4	KitKat	19	32.5%
5.0	Lollipop	21	16.2%
5.1		22	19.4%
6.0	Marshmallow	23	7.5%

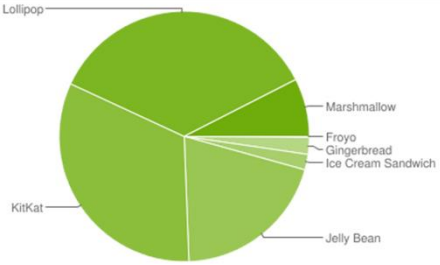


Figure 13: Android Version distribution developer.android.com (2016)

This section of the thesis will describe application design and implementation in more depth.
Starting with the user interface design and material design considerations, several application components such as notifications and user navigation but also specific Android activities and classes are explored.

3.3.1 User Interface & Material Design Guidelines

At a relatively early stage of this thesis, as soon as the general application concept and idea was set in place, early user interface sketches with pen and paper were drafted. A lot of these sketches have been discarded while others got refined after external assessment and discussions and feedback from the ALP group. Especially the official Material Design specifications for Android[23] have been taken into account while creating new versions of the user interface design. The Material Design specifications are guidelines created by Google with the goal to "*Create a visual language that synthesizes classic principles of good design with the innovation and possibility of technology and science.*" **[14]**. Components of these guidelines such as user navigation patterns or toolbar usage helped to improve early designs of the Traumschreiber app to provide a simpler and more consistent user experience. **Figure 14** shows two example sketches. The left drawing depicts one of the very first designs, while the right one shows one screen of the nearly finished design, incorporating more material design concepts.
The current user interface and finished screen designs will be depicted and described during component discussion, later in this chapter.

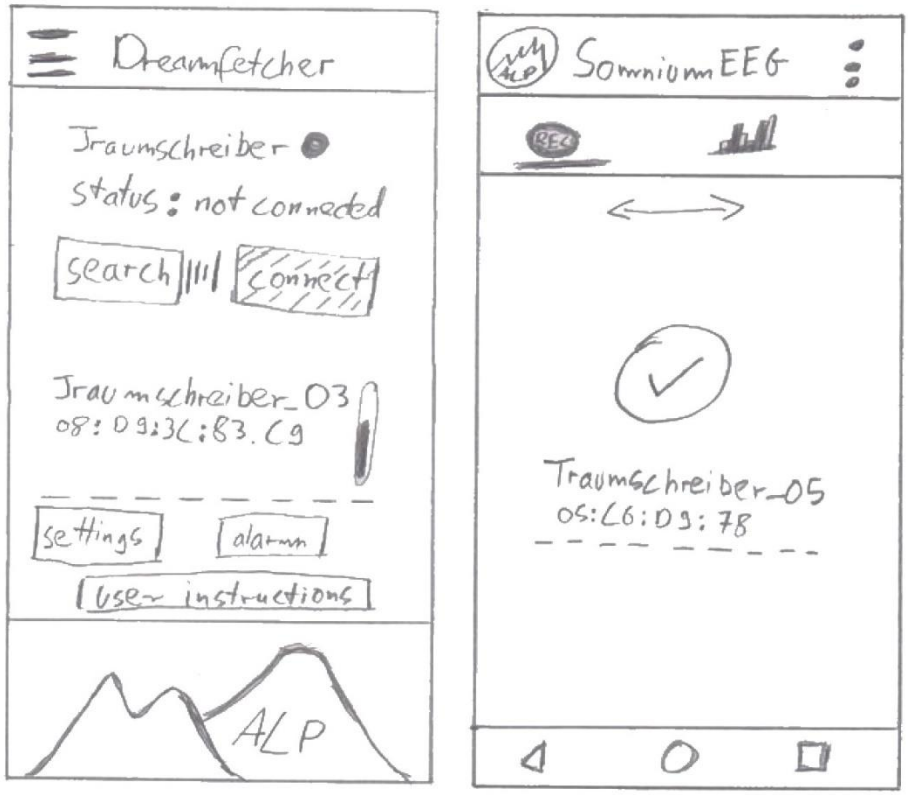


Figure 14: Traumschreiber application UI sketch examples (OC)

[23] *https://material.io/guidelines/*

3.3.1.1 User Navigation

A main component of every user interface is navigation, especially for mobile smartphone applications. How and when a user is able to reach different parts and screens of a given application can highly change end-user experience. The material design guidelines provide several different navigation patterns for different purposes.
Two frequently used patterns are, on the one hand the *Navigation Drawer* and on the other hand navigation using *Tabs* **[15]**. The navigation drawer is similar to a large menu that can be swiped in front of the application from the left side. It contains links to all other destinations inside the app[24]. For the Somnium app however, navigation using tabs was chosen to enable movement between the two main screens. Additionally, a menu[25] embedded inside the application toolbar[26] allows to reach less important screens. Tabs have been chosen over a navigation drawer pattern because it is only necessary to navigate between a small number of screens frequently. The official documentation states "*Tabs allow users to quickly move between a small number of equally important views*." **[15]**, which supported this decision. The applications `MainActivity` holds two different `fragments`, which can be swiped between. These two `fragments` are bound to the `MainActivity` with the `SwipeFragmentPagerAdapter` class. **Figure 15** at position (**1**) depicts how the user is able to swipe between the `ConnectionFragment` and the `DataFragment`. Furthermore, at position (**2**) and expanded in **Figure 16** the app menu is displayed, which helps navigate through the remaining application screens. Similar to an activity layout, these menus are defined in XML (**Listing 4**). Every menu `item` holds an `android:onClick` attribute which calls a method defined in the `MainActivity`, that in turn fires an `Intent` to navigate to the desired screen. A full graph of all screen relations and their navigation pathways it mapped out later in section 3.3.2.

[24]To get more information about the navigation drawer visit:
https://material.io/guidelines/patterns/navigation-drawer.html
[25] More in depth information about Android menus:
https://material.io/guidelines/components/menus.html#
[26] Details about the Android toolbars:
https://material.io/guidelines/components/toolbars.html#

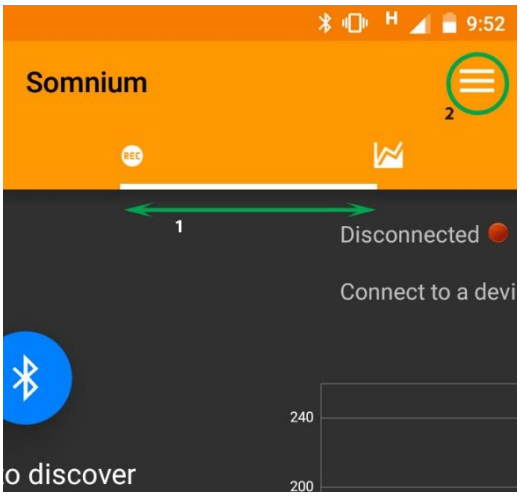


Figure 15: Somnium Navigation (OC)

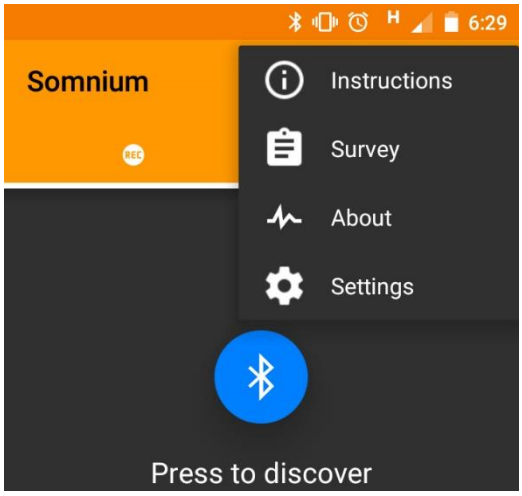


Figure 16: Somnium menu (OC)

```
<item
    android:icon="@drawable/menu"
    android:title="@string/settings"
    app:showAsAction="ifRoom">
    <menu>
        <item
            android:icon="@drawable/ic_info_outline_white_36dp"
            android:onClick="replayIntroActivity"
            android:title="@string/instructions">

        </item>
        <item
            android:icon="@drawable/assignment_white_48x48"
            android:onClick="showSurvey"
        .
        .
        .
```

Listing 4: Application menu XML example

3.3.1.2 Notifications & Toasts

Another important part, required for well-designed usability, is to provide the user with sufficient feedback about the application status, behavior and about ongoing internal processes. Two user interface components have been implemented to achieve this goal. When using the Somnium app, the application is often executing internal processes that are important but usually invisible to the user. Related to the Traumschreiber app, these are for example: the automatic disconnection from a BLE device if no signal is received for a specific time or the detection of a disabled BLE adapter while a device search is requested. To provide the user with feedback on those processes so-called Toasts[27] have been implemented. Toasts are short messages that can be prompted to the user at the bottom of the application screen for a certain timeframe (**Figure 17**). Integrated into the `activity` or `fragment` logic, toasts are fairly simple to implement (**Listing 5**).

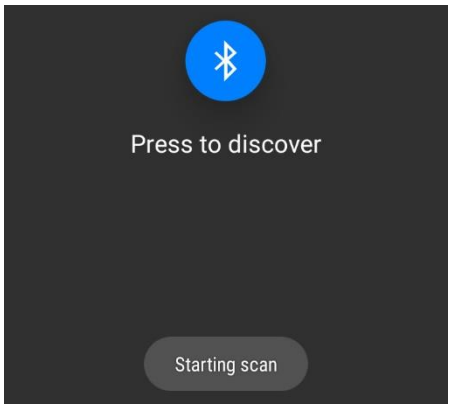


Figure 17: Toast UI example (OC)

```
if (!mBluetoothAdapter.isEnabled()) {
    Toast.makeText(getContext(), "Enable bluetooth and try again",
Toast.LENGTH_LONG).show();
    return;
}
```

Listing 5: Displaying a toast on screen

The second component integrated into the Somnium app is the so-called Notification[28]. A notification is a dynamic user interface element that is shown in the Android notification and status bar outside of your application.

[27] Detailed information about Toasts:
https://developer.android.com/guide/topics/ui/notifiers/toasts.html
[28] Android documentation about Notifications:
https://developer.android.com/guide/topics/ui/notifiers/notifications.html

Typically, a notification becomes useful if information from background services should be shown to the user while the application is not in the foreground. The Somnium app displays a notification (**Figure 18)**, if a Bluetooth Low Energy device is connected and data is received. This notification indicates the ongoing connection to the user, displays the full duration of data transmission and furthermore allows the user to quickly return to the application by tapping on the notification. On successful BLE connection state change, the `showNotification();` or `killNotification();` method inside the `BLEConnectionService` is triggered.

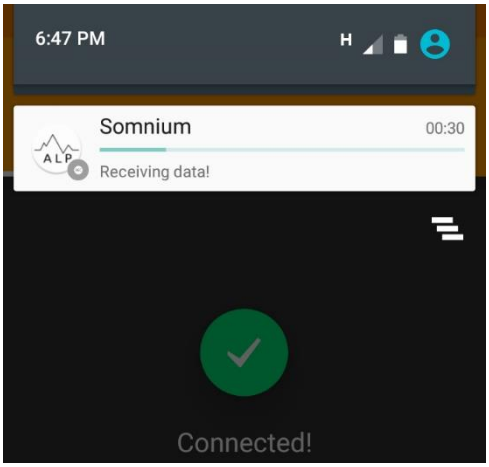


Figure 18: Somnium notification (OC)

3.3.2 Components and Content

The implemented Traumschreiber application currently contains 16 Java classes, such as activities and fragments, 12 corresponding XML design layout files and several additional resources, such as animations defined in XML or the manifest file. Overall, the present application, "Somnium" consists of approximately 3500 lines of custom code. This section will provide an overview first and later describe selected components, related functionality and the included content in more detail.

All external sources for images and icons included in the application, some of which have been customized with Photoshop, are referenced in appendix **A** Application Image and Icon Sources.

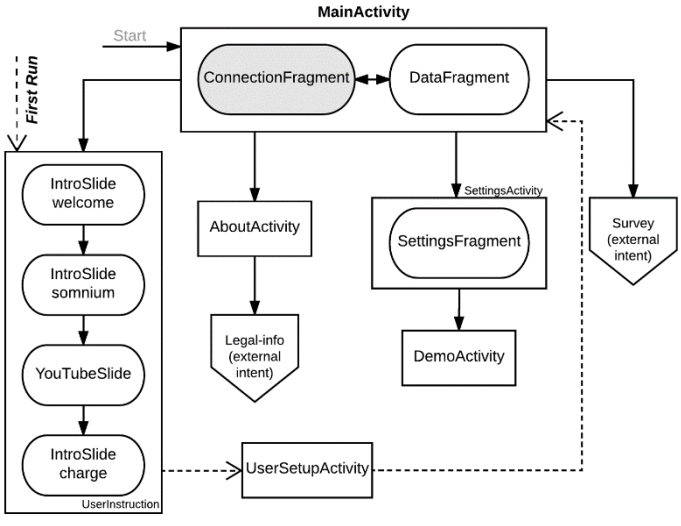


Figure 19: Application screen relationship flowchart

In **Figure 19** a screen relationship flowchart depicts the application's user interface structure. It shows, in what ways the user is able to navigate between screens and components. As soon as Somnium is started, it will always execute the `MainActivitys onCreate()` method and from there on, operate dependent on its lifecycle and user input. If Somnium is started for

the first time however, logic inside the `MainActivitys onCreate()` method will trigger an intent to display the user instructions slides followed by the `UserSetupActivity`. The user is introduced to the application and a short video tutorial is provided. Moreover, basic user data is collected. From here on, if the application is started, the `MainActivity` will embed and display the `ConnectionFragment` as well as the `DataFragment`. The first fragment allows the user to search and connect to the BLE device and the latter displays the received data.

The previously described menu, furthermore enables the user to reach the `AboutActivity,` which holds contact information and links to external legal information as well as the `SettingsActivity` and an external Survey.

The `SettingsActivity,` allows the user to access and change his information, application settings as well as to reach the yet empty `DemoActivity`.

Conducting a user/experiment survey is currently realized with Google forms[29], started from the menu through implicit intents, the Android OS opens the pre-defined survey link inside an available online browser (**Figure 20**) . The biggest advantage this method offers, is the possibility to change and replace the survey's content easily. Moreover, collected user data can be dynamically integrated into the survey's HTTP link and thus be automatically filled into the survey.

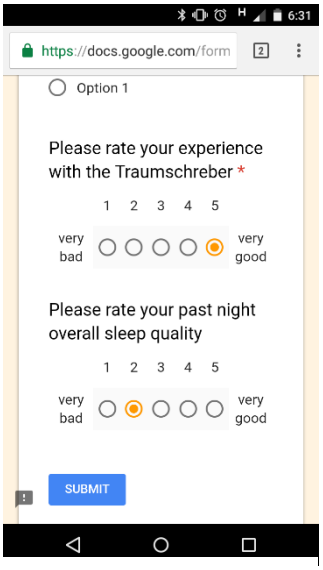


Figure 20: User surveys (OC)

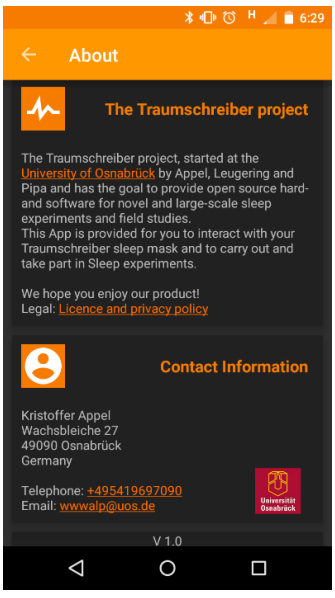


Figure 21: About activity (OC)

The About activity (**Figure 21**), also reachable through the toolbar menu provides a short and central point of information about the Traumschreiber project as well as contact information. The XML layout consist of so-called CardView[30] elements, that maintain a consistent and structured look. The description text links to the imprint of the Traumschreiber homepage[31].

To make the user familiar with his sleep mask and its application, the introduction slides have been implemented (**Figure 22**). As mentioned in section

[29] Detailed information about Google forms: *https://www.google.com/forms/about/*
[30] *https://developer.android.com/training/material/lists-cards.html#CardView*
[31] *https://www.traumschreiber.uni-osnabrueck.de/impressum.html*

3.2.2, to support this feature, the AppIntro library was used. The `IntroSlide` fragment class acts as a blueprint for every slide except the `YouTubeSlide` which needs additional logic. The content of every different slide is simply represented within their corresponding XML layouts, such as "`welcome_slide.xml`". Linking the blueprint fragment classes to the designed XML layouts is implemented inside the `UserInstruction` class (**Listing 6**). It becomes very easy to add new slides, changes the content or rearrange them with this approach. As soon as the intent form the MainActivity is triggered the introduction slides are created and their layouts are inflated.

```
public void init(Bundle savedInstanceState) {
        .
        .
        addSlide(IntroSlide.newInstance(R.layout.welcome_slide));
        addSlide(IntroSlide.newInstance(R.layout.somnium_slide));
        addSlide(YouTubeSlide.newInstance(R.layout.youtube_slide));
        addSlide(IntroSlide.newInstance(R.layout.charge_slide));
        .
        .
}
```

Listing 6: UserInstruction class excerpt

The `YouTubeSlide` fragment class provides additional logic to display a tutorial video. Currently this video is a short demo that can be found on the official Traumschreiber homepage. The video, hosted on Youtube, is integrated into the fragment slide on runtime, within a `YouTubePlayerSupportFragment` that is provided by the Youtube API mentioned in section 3.2.2.

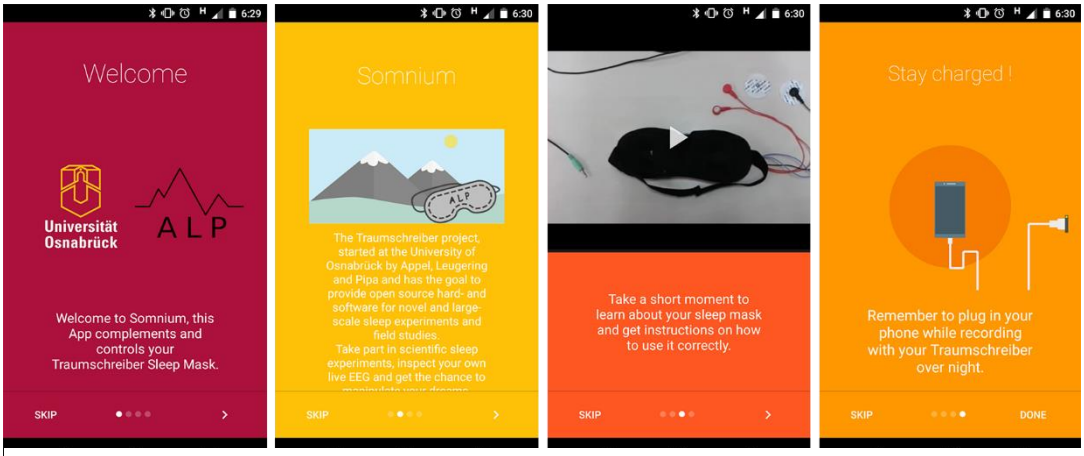


Figure 22: Introduction slides (OC)

Moreover, if Somnium launches for the first time, as soon as the user finished to read the introduction slides, he is prompted with the `UserSetupActivity` (**Figure 24**). A `RadioButton` as well as a `NumberPicker` is displayed, that allow the user to input his year of birth as well as gender. Later on, it is possible to access and change the entered data inside the `SettingsActivity` (**Figure 23**). This activity hosts the `SettingsFragment`, which displays the user data, links to the `DemoActivity` as well as other options. All selected options as well as the entered data is permanently stored on the device, even if the application process is terminated. Every option is saved within a `SharedPreference`[32], a key-value pair set that allows to access and change the stored data throughout the application. Initial key and value pairs are defined inside the `prefernces.xml` file and added to the fragment within its `onCreate()` method.

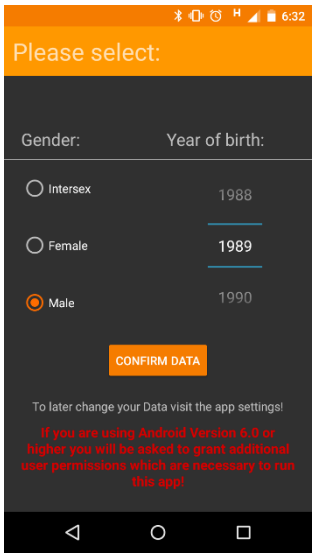


Figure 24: User setup screen (OC)

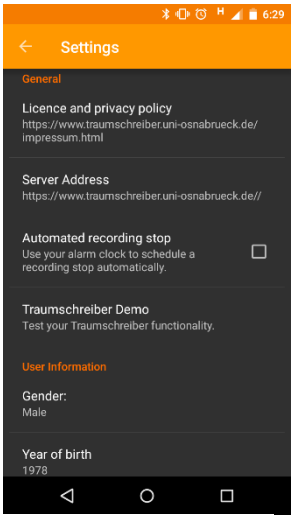


Figure 23: Settings screen (OC)

Somniums main screen, which is opened when starting the application, is the connection screen (**Figure 25**). Implemented within the `ConnectionFragment` class, is the logic to enable the user to search the surrounding for available BLE devices. All found devices, including their context information is then listed. As soon as the user tabs on the Bluetooth button, its `onClickListener` method checks if the BLE adapter is enabled and then further triggers the `scanLeDevice() method` to start the search. Every found device is added to the fragments `ListView` and displayed on screen. As soon as the user selects a device from the list, its `OnItemClickListener` method triggers the `connect()` method that is implemented inside the `ConnectionFragments` host, the `MainActivity`.

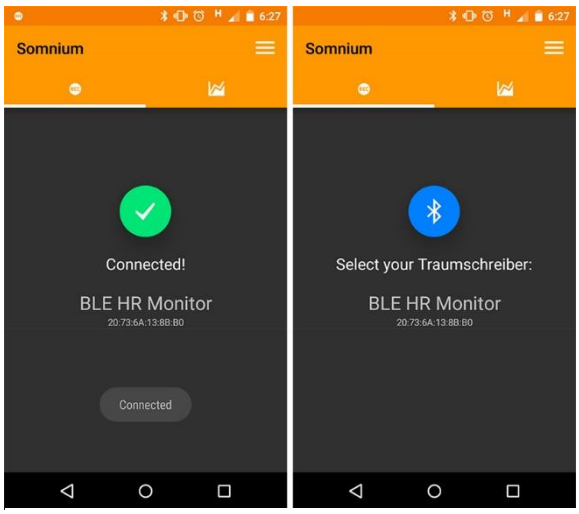


Figure 25: Connection screen (OC)

[32]Details about shared preferences: *https://developer.android.com/training/basics/data-storage/shared-preferences.html*

When the connection state changes, a variety of different methods handle the graphical user interface and for example change the Bluetooth button or display feedback toasts to the user. The main task of the `connect()` method is to start the `BLEConnectionService`. The important service, build on top of Googles official BluetoothLeService[33] sample, handles the ongoing connection to a BLE device as a background service. Even if the application is minimized, it will stay connected. If the service is able to connect to the desired device, the `showNotification()` method triggers the information bar outside of the application's user interface, described in section 3.3.1.2.

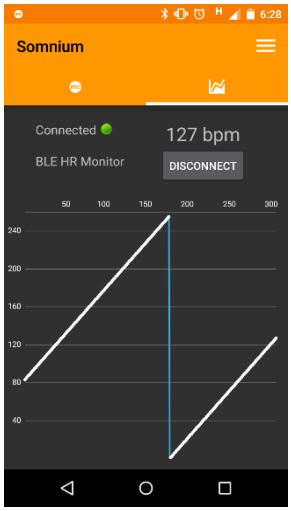


Figure 26: Data visualization screen (OC)

More importantly though, the service will subscribe to a BLE characteristic, determined by its UUID, within the `setCharacteristicNotification()` method. Received data from this BLE characteristic is then indirectly transmitted to the `MainActivity` via the `broadcastUpdate()` method. Immediately, as the `BroadcastReceiver` inside the `MainActivity` receives an update about a successful connection, it will call the `plottingSetup()` method. The MP Android Chart library described in section 3.31.2 will set up a line chart plot inside the `DataFragment` layout. Finally, the `BroadcastReceiver` is going to pass over the received BLE characteristic values to the `LineDataSet` which are plotted inside the `DataFragment layout` chart (**Figure 26**). If the BLE device is later disconnected manually or because of an interruption, the `clearUI()` method inside the MainActivity resets the user interface to its original form.

3.4 Challenges

The first big, general challenge concerning the application development was to get a proper overview. The task of conceptualizing, designing and implementing such a project was bigger than expected, considering that usually, these tasks are distributed between a team of designers and developers. Often times when researching a new feature, the rapid development of the Android operating system and smartphones became a problem. A lot of tutorials and information available online, even sometimes the official android developers page[34] contained outdated, incomplete and

[33] Android BluetoothLeGatt: *https://github.com/googlesamples/android-BluetoothLeGatt*
[34] *https://developer.android.com/index.html*

insufficient data. Finding out what the best practices for current Android versions on certain topics are, proved to be hard, without having a senior Android app developer at hand, sometimes only posting on stackoverflow.com would provide a solution. Bluetooth Low energy in conjunction with Android is a good example, although the technology itself is still very young, even official android tutorials already contain deprecated methods and incomplete information. Another example was, to find out what exactly the Android Operating System is doing internally in some situations, for example what specifically happens when a user force closes the application, with respect to its lifecycle.

Furthermore, another challenge specific to android development was to support and account for several different soft- and hardware combinations. A good example is, how the inquiry of user permissions is handled differently between Android versions. As of version 6.0 additional code needs to be implemented to request permissions such as the device location on runtime, in contrast to requesting it on application installation **[16]**.

Lastly, graphical user interface programming often provided the challenge to account for different and sometimes unexpected application stats. The introduction video provided in one of the slide fragments requires an active internet connection to work, therefore, to not provide the user with a black screen, a separate fragment layout needed to be implemented that replaces the video in case the smartphone is in an offline state.

4 Application Deployment and Testing

Debugging and testing a new Android application on several different devices and operating system versions is usually very easy. This is because Android Studio provides the possibility to emulate different devices and OS versions the app can be run and tested on easily[35]. Bluetooth capabilities however are not supported on emulated devices. For that reason, the Somnium app was tested only on physical devices and thus on a smaller range of hard- and software combinations. During ongoing development, typically one device was connected to Android Studio, which allowed to test small app changes with the Instant Run feature[36]. On bigger implementation milestones however, the application was tested on a wider range of devices to ensure compatibility, these devices are listed in section 4.1.
Once an Android application is ready for release and deployment, a so-called Android application package (APK) **[17]** is generated. Android Studio packages all project resources and source code into one APK file. This allows easy transfer to- and installation on all supported devices. For final deployment, ProGuard[37], a code optimization tool, can be used to shrink the resulting APK size. ProGuard is set and configured inside the `build.gradle` file. Among other things, it is able to remove unused resources and left over code from libraries. After ProGuard was used, the deploy ready Somnium.apk was reduced to 1,47 megabytes of data. It is included on the attached CD-ROM in appendix **B** CD-ROM.

[35] Run Apps on the Android Emulator: *https://developer.android.com/studio/run/emulator.html*
[36] *https://developer.android.com/studio/run/index.html#instant-run*
[37] Information about ProGuard: *https://developer.android.com/studio/build/shrink-code.html*

4.1 Test Devices & Android Versions

The Traumschreiber app "Somnium" was tested on the following list of devices and Android operating system versions:

Device	Android Version
Motorola moto G (1st generation) **Doogee X5**	Version **5.1** "Lollipop"
Google Nexus 7 (2013)	Version **6.0** "Marshmallow"
Google Pixel	Version **7.1** "Nougat"

Table 1: Test Devices and Android Versions

4.2 Application Bugs

The current application release runs on all devices mentioned in **Table 1** without having any major issues. All problems and bugs that occurred during development have been debugged or bypassed.

One bug, which is worth mentioning although it has been bypassed in the current version, concerns the YouTube API. While the user slides trough the different introduction fragments, every upcoming fragment is preloaded to ensure smooth transition animations. But as soon as the fragment, containing the youtube view, is preloaded, the screens starts to flicker and the transition animation is not displayed correctly. During online research, the problem was found to be officially reported to Google, with current status: "WontFix" (Jul 15, 2016) **[18]**. Luckily a method to bypass this problem is provided within the report. After adding a `SurfaceView` with specific attributes to the layout containing the Youtube video, the fragment transition animation is displayed correct again although the bug itself is not fixed.

5 Outlook & Perspectives

The Traumschreiber sleep mask is currently still in development.
A new version, including an improved circuit board, is planned to be available in higher quantities soon. As already mentioned in chapter 3, some of the features conceptualized for the Somnium app have not been realized. This is, because among other things, delayed and defective hardware led to a lack of testing and development possibilities concerning the sleep mask's BLE data transmission. Although the current Traumschreiber app is not yet ready to use with the upcoming sleep mask, its development and user interface design created perspectives, knowledge and ideas for upcoming versions. As soon as this new sleep mask version is available, and the server that receives the Traumschreiber data is set up, missing features and also new ideas described in section 5.1 can be implemented, tested and improved. Furthermore, as soon as the hole system is able to work with a larger quantity of devices and is able to conduct experiments and record sleep mask data in parallel, usability studies about the sleep mask and application itself can be conducted to further improve the end-user experience and the Traumschreiber product overall.

5.1 Shortcomings

The current Traumschreiber mobile app is missing out on some considered features listed in the beginning of this thesis. First and foremost, data transmission between the smartphone application and a given webserver needs to be implemented. The Android app is supposed to connect to a remote server using websockets[38] in a secure way. For that reason, the chosen protocol needs to support secure websockets, implemented in Java. During online research the "*nv-websocket-client*" was found, it is secure and compatible with Android **[19]** and should therefore be considered for implementation in future app versions.
Additionally, as soon as the application is able to connect to the new Traumschreiber mask, the way BLE characteristics are read and how this received data is then plotted needs to be adjusted. As soon as BLE characteristics are read in correctly, a way to trigger the sensory stimulants can be implemented inside the yet empty `DemoActivity`.

[38] Information about the websocket protocol: *https://tools.ietf.org/html/rfc6455*

When recording sleep data, it was considered to provide some sort of data cache and long term storage. This should, on the one hand, prevent data loss if the internet connection it interrupted during data transmission. And on the other hand allow sleep data to be saved on and exported from the device. A good way to implement these features might be to use a SQLite database. The Android OS natively supports SQLite **[20]**, which provides a clean and standardized way of saving long- and short term data on the device.
Furthermore, allowing the Somnium app to set up and synchronize an alarm clock to a given Traumschreiber experiment is not yet possible. Although technically easy to implement, the feature was postponed because it is not yet clear how specifically these experiments will be integrated into the application flow. To implement this feature, implicit intents can be used to control and set Android's internal alarm manager **[21]**.

5.2 Additional Concepts

Before implementing and designing new application versions, additional features and ideas for improvement that came up after development should be considered.
While connected to a Traumschreiber device, additional user interface elements that indicate the connection strength to the sleep mask as well as its current battery level and status might prove useful. Additionally, the custom Bluetooth search button which also indicates the connection status might benefit from an upgrade to a so-called Floating Action Button[39] which brings further functionality recommended by the material design guidelines. These functionalities could include the possibility to save currently connected devices to a "previous devices" list and to automatically connect to the last known device, if in its range. Also, to further improve security, some sort of user and device authentication process should be considered, in which the user has to enter a device ID into the smartphone before a connection can be established. In terms of data visualization, additionally to showing a live EEG plot, tracking, analyzing and plotting sleep stages and cycles directly on the device could be even more interesting to a potential end user.
A very experimental feature for versions in the distant future, when the Traumschreiber might be capable of correctly and accurately detecting eye movement from the recorded EEG data, might be to use eye movement gestures to control the Android application and for example start a recording or play certain sounds.

[39] *https://material.io/guidelines/components/buttons-floating-action-button.html#*

6 Summary & Conclusion

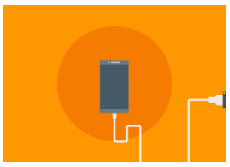
This thesis was started with the ambition to design and develop a mobile smartphone application to complement the sleeping mask that is part of the Traumschreiber project. At the beginning of the development process, a list of features, concepts and ideas was conceived and formulated. These features should enable the application to support the Traumschreiber project and its idea to provide a novel hard- and software system for new kinds of sleep experiments. After the initial planning phase, a user interface was designed and an Android application called “Somnium” was implemented that is meant to introduce private users to the sleep mask and help them use it properly. The application collects basic user data and provides the possibility to integrate user surveys. In its proximity it is able to search for available Bluetooth Low Energy devices and connect to a yet emulated sleep mask represented by a BLE development kit. On connection, the application subscribes to pre-defined BLE characteristics and received data is then plotted inside the user interface. The resulting application was tested on several device and software versions and all integrated features function as intended.

It has to be concluded though that not all goals have been achieved and some major and minor features that had been conceived still need implementation. However, during the process of this study a lot of knowledge was acquired with respect to Android development in general and further insight into implementation and design processes.

All being well, the application and concepts developed in this thesis can contribute and push the Traumschreiber project a bit further to achieve its goal and provide a new way of conducting sleep and dream research.

Appendices

A Application Image and Icon Sources

Image	Source
	*https://pixabay.com/en/icon-landscape-mountains-snow-1294942/* Public Domain Free for commercial use
	*https://pixabay.com/en/phone-battery-charge-energy-1503703/* Public Domain, Free for commercial use
	*https://www.shareicon.net/sleep-eye-mask-sleeping-mask-mask-travel-journey-113850* Creative Commons (Attribution 3.0)
	*Traumschreiber project: Johannes Leugering* (used with permission)
	*https://www.uni-osnabrueck.de* (used with permission)
	*https://www.materialui.co/icons* Public Domain, Free for commercial use

(All Icons used in the Application)

B CD-ROM

Every physical copy of this holds a CD-ROM which includes:

- This thesis as digital PDF document
- The entire Android Studio application project
- The deploy ready Somnium app APK

Bibliography

[1] The Traumschreiber Project [Online] *https://www.traumschreiber.uni-osnabrueck.de* [accessed January 2017].

[2] Android Developers [Online] *https://developer.android.com/index.html* [accessed January 2017]

[3] Alhola P, Polo-Kantola P. (2007). Sleep deprivation: impact on cognitive performance. Neuropsychiatr Dis Treat. 3(5):553–567.

[4] K. Appel, G. Pipa, M. Dresler. (2017). Lucid dreaming – an interdisciplinary overview. *In submission review*, Page 3.

[5] Jawbone fitness tracker "UP3" [Online] *https://jawbone.com/fitness-tracker/up3* [accessed January 2017]

[6] Generic Attributes (GATT) and the Generic Attribute Profile [Online] *https://www.bluetooth.com/specifications/generic-attributes-overview* [accessed January 2017]

[7] A brief history of Android phones [Online] *https://www.cnet.com/news/a-brief-history-of-android-phones/* [accessed January 2017]

[8] Google Android hits market share record [Online] *http://www.cnbc.com/2016/11/03/google-android-hits-market-share-record-with-nearly-9-in-every-10-smartphones-using-it.html* [accessed January 2017]

[9] Google says there are now 1.4 billion active Android devices [Online] *http://www.androidcentral.com/google-says-there-are-now-14-billion-active-android-devices-worldwide* [accessed January 2017]

[10] Android developer documentation [Online] *https://developer.android.com/develop/index.html* [accessed January 2017]

[11] Bluetooth Low Energy API [Online] *https://developer.android.com/guide/topics/connectivity/bluetooth-le.html* [accessed January 2017]

[12] Android Platform Versions [Online]
https://developer.android.com/about/dashboards/index.html#Platform
[accessed January 2017]

[13] Android 5.0 APIs [Online]
https://developer.android.com/about/versions/android-5.0.html
[accessed January 2017]

[14] Material Design Goals [Online]
https://material.io/guidelines/#introduction-goals
[accessed January 2017]

[15] Material Design Navigation Patterns [Online]
https://material.io/guidelines/patterns/navigation.html#navigation-patterns [accessed January 2017]

[16] Requesting Permissions at Run Time [Online]
https://developer.android.com/training/permissions/requesting.html
[accessed January 2017]

[17] Building Your Application for Release [Online]
https://developer.android.com/studio/publish/preparing.html#publishing-build [accessed January 2017]

[18] Issue 4722: YouTubePlayerSupportFragment flickers [Online]
https://code.google.com/p/gdata-issues/issues/detail?id=4722
[accessed January 2017]

[19] High-quality WebSocket client implementation in Java [Online]
https://github.com/TakahikoKawasaki/nv-websocket-client
[accessed January 2017]

[20] Saving Data in SQL Databases [Online]
https://developer.android.com/training/basics/data-storage/databases.html [accessed January 2017]

[21] Scheduling Repeating Alarms [Online]
https://developer.android.com/training/scheduling/alarms.html
[accessed January 2017]

DECLARATION OF AUTHORSHIP

I hereby certify that the work presented here is, to the best of my knowledge and belief, original and the result of my own investigations, except as acknowledged, and has not been submitted, either in part or whole, for a degree at this or any other university.

Martin Jäkel

Osnabrück, February 8, 2017