CZECH TECHNICAL UNIVERCITY IN PRAGUE

Faculty of Electrical Engineering

Department of Telecommunications Engineering

Mobile Application for Controlling the Embedded System

B.c. Programme: Communication, Multimedia & Electronics

Specialization: Communications and Electronics

January 2018

Author: Lazaros Stamatiadis

Supervisor: Ing. Tomáš Zitta

BACHELOR'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Stamatiadis Lazaros** Personal ID number: **437030**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Telecommunications Engineering**

Study program: **Communications, Multimedia, Electronics**

II. Bachelor's thesis details

Bachelor's thesis title in English:

Mobile Application for Controlling the Embedded System

Bachelor's thesis title in Czech:

Mobilní aplikace pro ovládání embedded platformy

Guidelines:

Design and implement a mobile application for OS Android with capability to control selected IoT device - embedded system. Perform a security analysis of preferred communication technologies Bluetooth, NFC and Wi-Fi. Compare these communication technologies in terms of data rate and latency. Statistically evaluate and document performed tests.

Bibliography / sources:

[1] MacLean, D.; Komatineni, S.; Grant, A.: Pro Android 5. Berkeley: Apress, 2015. ISBN: 978-1-4302-4680-0
[2] Dodd, A. Z.: The essential guide to telecommunications, 5th edition. Upper Saddle River, NJ: Prentice Hall, 2012. ISBN: 978-0-1370-5891-4

Name and workplace of bachelor's thesis supervisor:

Ing. Tomáš Zitta, Department of Telecommunications En, FEL

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **18.10.2017** Deadline for bachelor thesis submission: **09.01.2018**

Assignment valid until: **28.02.2019**

Ing. Tomáš Zitta		prof. Ing. Pavel Ripka, CSc.
Supervisor's signature	Head of department's signature	Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

I hereby declare that this bachelor thesis is completely my own work and that I used only the cited sources in accordance with the Methodical instruction about observance of ethical principles of preparation of university final projects.

Prague, January 9, 2018

. . . . . .

Signature

Acknowledgements

I would like to sincerely thank my supervision, Ing. Tomáš Zitta, for his support and guidance throughout the thesis. My special thanks also belong to my family, for their encouragement and love.

Abstract

Moving towards the era of IoT, where many different types of devices can interact with each other remotely, smartphones are required to further expand their capabilities in areas that a few years back were unheard of. This advance in the communications and devices' hardware, increases the need for software development that can take advantage of what IoT has to offer.

In this thesis, the author attempts to create a mobile application based on Android Operating System, which will access and control a remote embedded system device, over a Secure Shell connection via Wi-Fi and Bluetooth. In addition this thesis includes an evaluation and comparison of each communication technology based on achieved throughput and latency.

Key words: Android, IoT, Wi-Fi, Bluetooth, Raspberry Pi, Wireless Communications, Java, NFC, SSH

Abstrakt

Přesouváme-li se k éře IoT, kde mnoha různých typů zařízení mohou na sebe působitvzájemně vzdáleně, je nutné aby u chytrých telefónů došlo k dalšímu rozšíření svých schopností v oblastech, o kterých se před několika lety ještě neslýchalo. Tento pokrok v oblasti komunikací a hardwaru zařízení, zvyšuje potřebu vývoje softwaru, který může využít výhod, které loT nabízí.

V této práci se autor pokouší vytvořit mobilní aplikaci založenou na operačním systému Android, která bude mít přístup a bude moci ovládat na dálku vestavěné systémové zařízení přes připojení Secure Shell přes Wi-Fi a Bluetooth. Dále tato práce obsahuje hodnocení a porovnání každé komunikační technologie založené na dosažené propustnosti a latenci.

Klíčová slova: Android, IoT, Wi-Fi, Bluetooth, Rasberry Pi, Bezdrátové komunikace, Java, NFC, SSH

Content

List of figures

List of Tables

List of Acronyms

IoT – Internet of Things

OS – Operating System

IEEE – Institute of Electrical and Electronics Engineering

OHA – Open Handset Alliance

Wi-Fi – Wireless Fidelity

MPEGG4 – Motion Picture Experts Group Layer-4

CDMA/EVDO – Code-division multiple access/Evolution Data Optimized

VPV – Virtual Private Network

HMLT5 – Hypertext Markup Language 5

UI – User Interface

PPI – Pixels per inch

NFC – Near Field Communications

CPU – Central processing unit

BSD – Berkeley Software Distribution

JPG – Join Photographic Group

API – Application Programming Interface

HTTPS – Hypertext Transfer Protocol (Secure)

SHA – Secure Hash Algorithms

RFID – Radio Frequency Identification

GSMA – Global System Mobile Association

TCP/IP – Transmission Control Protocol

XML – Extensible Markup Language

WLAN – Wireless Local Area Network

PAN – Personal Area Network

UDP – User Datagram Protocol

SSH – Secure Shell

RFCOMM – Radio Frequency Communications

Mbits – Mega bits

kbits – kilo bits

DOS – Disc Operating Systems

ICMP – Internet Control Message Protocol

DVM – Dalvik Virtual Machine

JVM – Java Virtual Machine

L2CAP – Logical Link Control and Adaptation Protocol

LMP – Link Management Protocol

SDP – Service Discovery Protocol

Introduction

1.1. Motivation

In the later years smartphones have become the central control station of our everyday lives used in a wide spectrum of areas, from communications to entertainment and online shopping to healthcare and security, the use of smartphones is an essential to our lives. Moving towards the era of IoT, where many different types of devices can interact with each other remotely, smartphones are required to further expand their capabilities in areas that a few years back were unheard of. This advance in the communications and devices' hardware, increases the need for software development that can take advantage of what IoT has to offer.

One area that has seen significant contribution from developers across the world is the home automation, where nowadays a person can control their home appliances from their smartphones. From controlling home appliances to regulating thermostats and setting up home surveillance systems, user obtains control to almost every electronic device inside their homes and everything is controlled from within their smartphones.

Therefore, the idea of contributing to this rapidly developing world of mobile application development and gain more experience on the current technological advancement and trends while working on this thesis, was the main source of motivation.

1.2. Outline of Thesis

The thesis is constructed in the following way: Firstly, in ***Chapter 2. Android Operating System***, the author introduces the reader to the basics of Android Operating System, with a brief explanation of the history of Android OS versions and an introduction to Android OS Architecture. Concluding Chapter 2., the author refers to some limitations and security issues of the Operating system.

Following are ***Chapters 3. Communication between the Mobile Application and the embedded system***, and ***Chapter 4. Implementation of the Android Application***, where the author briefly describes the communication scenario between the mobile application and the selected embedded system, and the actual implementation and functionality of the mobile application respectively.

The final chapter of the thesis is ***Chapter 5. Tests & Results***, documents the procedure followed for the evaluation of each of the selected communication technologies and the outcome of the results considering the achieved throughput and the latency. Moreover, Chapter 5 included security comparison of each of the selected communication technologies.

The ***Conclusion*** of the thesis includes the outcome of the evaluation and few suggestions on feature developments of the application.

Android Operating System

According to the latest statistics on smartphone's Operating Systems Market Share, Android holds a well-established strong position with over 64% of the market share[1]. Google, not only holds the leading position on market share for operating systems, but also in similar charts about number of applications on online store and active developers[2]. These statistics show that Android not only is the preferred OS among end users but also among the developers.

Given the fact that Android is an open-source operating system and uses Java as the main programming language, it makes it easier for developers and manufacturers to access all the features available and manipulate the operating system to much their needs.

2.1. History of Android OS

Since the public release of Android's Beta version on November 5, 2007 Google have continued with regular updates on their operating system. The first commercial version of the Android OS, was released on September 2008 under the codename Android 1.0 but since then Google and OHA have decided to give a confectionery-themed codenames to every new version of Android that is released.

- **Android 1.0 (API 1)**

Initially released on September 23, 2008 and added some more features to the already existing Android Beta. Some of these features were support of Bluetooth and Wi-Fi, Google Talk instant messages and Google Sync, allowing management over-the-air synchronization of Gmail, Calendar and other Google apps. Also with this version of Android some more applications where introduced such as YouTube video player, Android Market and Google maps with Street View for mobiles. [3]

- **Android 1.1 (API 2)**

Initially released on February 9, 2009 this version was the first one to release with a codename and was known as "Petit Four", though this name was not used officially. It added some extra features to the ones existing in version 1.0, such as ability to save attachment in messages and support for marquee in system layouts. [3]

- **Android 1.5 (API 3)**

On April 27, 2009 the Android 1.5 was released under the codename Cupcake, a theme which follow all version releases henceforth. In this version a lot of new features where introduced, such as support for third-party keyboards, Widgets, support of MPEG4 video recording and playback, ability to upload videos on YouTube, Auto-pairing and stereo support for Bluetooth and many more. [4]

- **Android 1.6 (API 4)**

This version was released on September 15, 2009 under the codename Donut and it was based on Linux 2.6.29. In this update among other new features, updated technology support for CDMA/EVDO, 802.1x, VPNs and text-to-speech engine were included. In addition to that support for WVGA screen resolution was also introduced with this version. [5]

- **Android 2.0 (API 5) – 2.1 (API 7)**

On October 26.2009 the Android 2.0 SDK was released, known as the Éclair. The Google account synchronization was further expanded and users now can synchronize multiple accounts on the same device. Bluetooth 2.1 support, support of HTML5 for web-browser, optimization on hardware and revamped UI there a few of the new features brought to users with this version update. Also there numerous new camera features, further developing camera performance and support of extra camera related hardware. Android 2.1 version was just an update for API and some bug fixes. [6],[7],[8]

- **Android 2.2 – 2.2.3 (API 8)**

On May 20, 2010 the new update for Android OS was released under the codename Froyo, this time based on Linux kernel 2.6.32. This version included Android Cloud to Device Messaging (C2DM) service, USB tethering and Wi-Fi hotspot functionality, support for high-PPI displays, Adobe Flash support, etc. The 2.2.3 update further included two security updates and minor bug fixes. [9]

- **Android 2.3 (API 9) – 2.3.7 (API 10)**

Android 2.3 Gingerbread was released on December 6, 2010. This update increased the simplicity and speed of the UI and further added some new features to the user's experience. Some of which were native support of SIP VoIP internet telephony, faster and more intuitive keyboard, support for Near Field Communication (NFC), Download Manager, as well as native support for more sensors such as gyroscope and barometers. For the update 2.3.3 a new API was implemented (API 10). [10], [11]

- **Android 3.0 (API 11) – 3.2.6 (API 13)**

Android 3.0 version was released on February 22, 2011 with the codename Honeycomb and it was the first tablet-only Android update. This update optimized the tablet support and added some new features to tablet user's experience. [12],[13],[14]

- **Android 4.0 (API 14) – 4.0.4 (API 15)**

On October 18, 2011 the new Android Ice-Cream Sandwich was released this time based on a Linux kernel 3.0.1. The Ice-Cream Sandwich was the last version of Android to support Adobe Systems' Flash player. However, as all previous updates this update also brought new features to Android OS. The 4.0.3 version of Android also included a new API update (API 15). Some of the new features of this update were Android VPN Framework (AVF) and Face Unlock which allowed users to unlock their devices using facial recognition software. Another new and important feature was the Android Beam, a near-field communication feature that allowed the rapid short-range exchange of data. [15], [16]

- **Android 4.1 (API 16) – 4.3.1 (API 18)**

Android 4.1, also known as Jelly Bean, was released on July 9, 2012. It was mainly aimed to improve the functionality and performance of the user interface. This version was based on Linux kernel 3.0.31 and among many other new features it brought Bluetooth data transfer for Android Beam and enhanced accessibility. Releases of Android 4.2 and 4.3 followed, with the first version (4.2) adding new stacks for Bluetooth and NFC technologies and SELinux and the second version bringing Bluetooth low energy support, Bluetooth Audio/Video Remote Control Profile (AVRCP), OpenGL ES 3.0 support, 4K resolution support and numerous security updates and performance enhancements. [17],[18],[19]

- **Android 4.4 – 4.4.4 (API 19)**

Google released Android 4.4 KitKat on October 31, 2013 with a refreshed interface and many new features enabling further usability and customization on the user interface. Later updates of the 4.4 KitKat version extended the improvement in UI, both in its appearance as well as in its accessibility. The update 4.4.4 included a security fix, eliminating an OpenSSL man-in-the-middle vulnerability.
As part of the KitKat version, Google released 4.4W KitKat on June 24, 2014 which was an exclusive version of the Android OS for wearables. [20]

- **Android 5.0 (API 21) – 5.1.1 (API 22)**

Android 5.0 Lollipop was released on November 12, 2014 as an over-the-air (OTA) update for selected devices. Lollipop featured a redesigned UI built around a responsive design language referred to as material design. It brought numerous improvements in the notifications process and it replaced Dalvik with Android Runtime (ART), which improved application performance. Also this update included support for 64-bit CPUs and a new technology named by Google as Project Volta for battery life improvements.
On March 9, 2015 Android 5.0 further updated to Android 5.1 remain under the same codename (Lollipop), which among other features and improvements it included support for multiple SIM cards, Device protection which was a security feature allowing the device to remain lock until the user sings into their Google account, in case the device was stolen or lost. – [21],[22]

- **Android 6.0 – 6.0.1 (API 23)**

Google released Android 6.0 Marshmallow on October 5, 2015 and with it, it introduced some new features to the users. Some of the most important ones were the introduction of Doze mode, which reduces CPU speed while display is off to save battery life. Also the Marshmallow update included native fingerprint reader support, USB Type-C support, and an experimental version of the multi-window feature. – [23]

- **Android 7.0 (API 24) – 7.1.2 (API 25)**

Android 7.0 Nougat was commercially released on August 22, 2016. This version of Android OS added more UI characteristics allowing the user to further expand the capabilities of their devices, as well as few new technologies and improvements on previously established technologies. For instance, Android Nougat introduced Daydream virtual reality platform, improvement on Doze functionality aimed to prolong battery life, finalized multi-window support etc. Also with Nougat, Google introduced a new Just in Time (JIT) compiler with code profiling to ART, which makes for 75 percent faster app installation and 50 percent reduction in compiled code size, as well as improving the performance of Android apps as they run.
On October 19, 2016 Google released Android 7.1 which mainly improved user's Android experience and brought some more improvements to the UI. It also included minor bug fixes and improvements for Google's Pixel smartphone. – [24],[25]

- **Android 8.0 (API 26)**

The most recent version of the Android OS is the Android 8.0 Oreo which was released by Google on August 21, 2017. The biggest change that this update brought to the Android world

was the introduction of Project Treble, a modular architecture which allowed hardware manufactures to deliver Android updates on their devices, easier and faster. A few of the rest features included to this update are Sony LDAC codec support, 2 times faster boot time, Google Play Project, Wi-Fi Assistant and Android Go which is a optional more lightweight version of Android for "low-end" devices with less than 1GB RAM. – [26]

2.2. Android OS Architecture

The Android OS is an open-source operating system, with the resource code released publicly by Google on every new update of the OS, under the Apache 2.0 license.
The software stack of Android is divided into four layers: Kernel layer, Libraries and runtime layer, application framework layer and application layer.

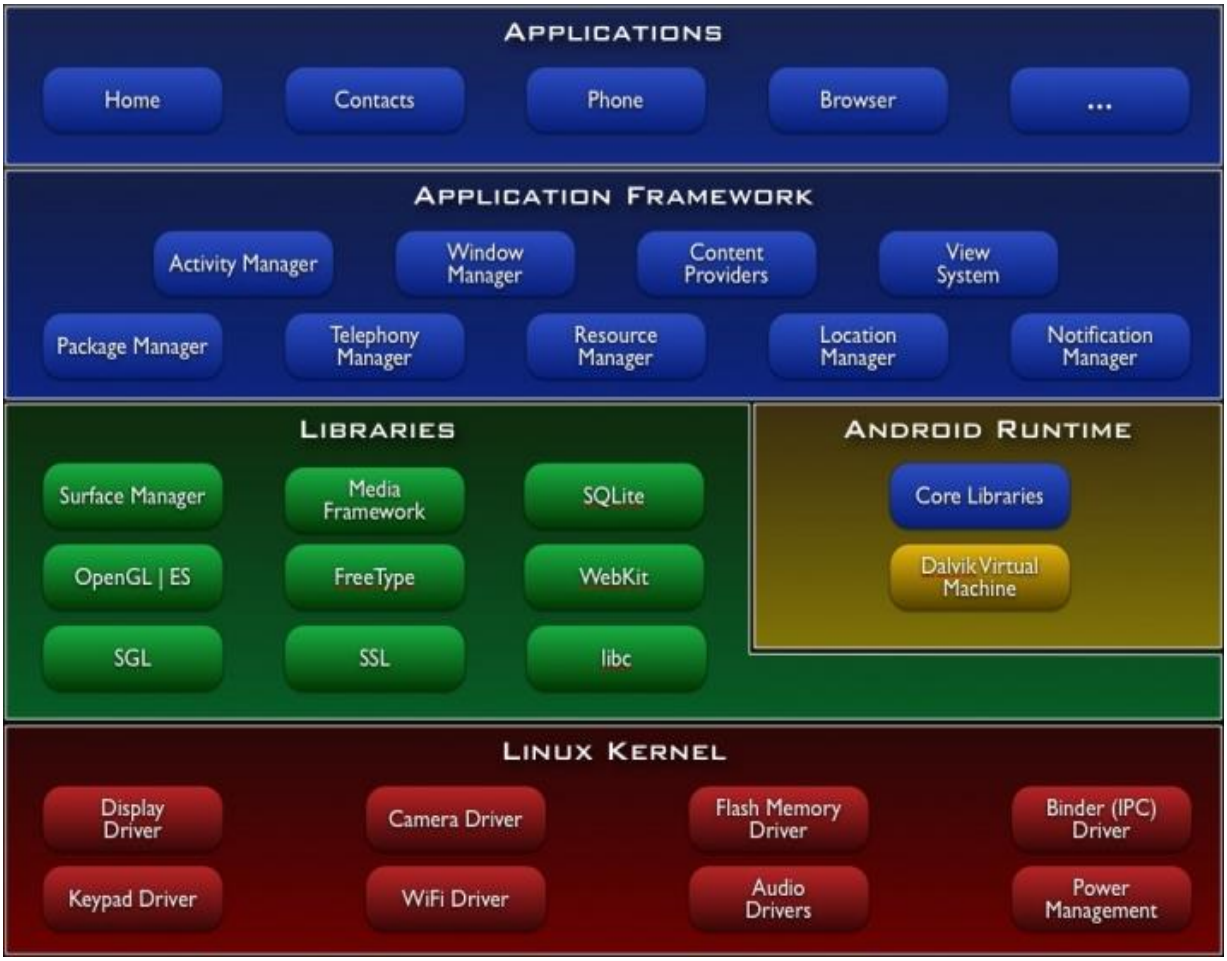

Figure 1. Architecture of Android Operating System **[27]**

2.2.1. Linux Kernel

Android kernel is based on Linux 2.6 kernel, with some modification added by Google and it is regularly updated, whenever a new version of Android OS is released. This layer is the core of android architecture, as it provides the main services for the operating system. It acts as the abstraction level between the hardware of the device and the upper levels of the software stack. The kernel includes drivers for different hardware components of the device, and it provides system services such as the power management and memory management. In addition it provides network stack and some security services. [27],[28]

2.2.2. Libraries and runtime layer

On the top of the Linux kernel layer is the native libraries of Android. This layer enables the device to handle different types of data related to the hardware.
These libraries are written in C or C++ programming language and they are accessed through Java interface. Google's specifically designed C/C++ library for Android, so called Bionic, is a derivation of standard BSD's and system C library (libc). The Java libraries used by Android are based on OpenJDK.[27] Some of the native libraries included are:

- Surface Manager: used for managing the display of the device and the composition of windows on screen.
- SQLite: SQLite is the database used from Android, for data storage.
- WebKit: it is the browser engine used to display HTML content.
- Media framework: It is used to provide playbacks and recording of various audio, video and image formats, such as MP3, AMR, JPG, MPEG4 etc.
- OpenGL|ES: it is used to render 2D and 3D graphics content on the screen.

The Android runtime consists of the Dalvik Virtual machine (DVM) and core Java libraries. The Dalvik Virtual machine was specifically designed for Android and it was mainly developed for mobile devices. It is similar with the Java Virtual machine (JVM), although there are differences and DVM is optimized for Android.

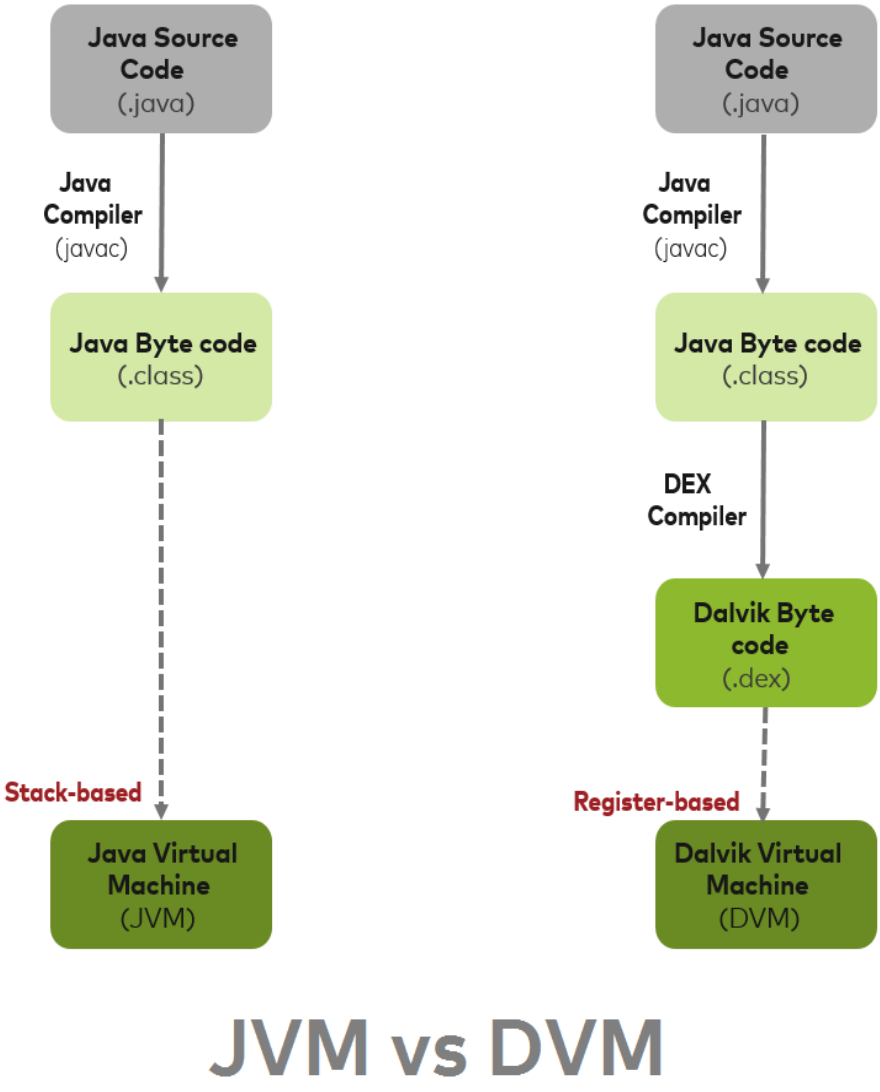

Figure 2. Dalvik Virtual machine and comparison with Java Virtual machine **[58]**

In Dalvik Virtual machine, every application runs in its own process. Dalvik uses its own 16 bit instruction set in comparison with Java's 8 bit stack instruction, thus reducing instruction count and increasing its interpreted speed. In Figure 2, we can see the execution of the code by the DVM. The source code is written at first in Java and with the help of a java compiler (javac), is compiled to Java byte code (.class files). Then, DEX compiler compiles the .class files to Dalvik Byte code, or .dex files, which is the format that can be finally executed by Dalvik virtual machine.

The Dalvik Virtual machine is optimized by Google, for low processing power and low memory environments and allows multiple instances of VM running simultaneously, providing security, isolation and memory management. [27],[28]

2.2.3. Application Framework

The Application framework is the layer that provides many higher-layer services and major APIs to the mobile application in the form of Java classes. Developers are allowed to make use of these libraries and include some of their provided services into their applications. Some of the most important blocks included in the Application framework layer are:

- Activity Manager: It is responsible to manage the application's life-cycle.
- Content Provider: Content Provider is used to manage the data sharing between applications and it manages how this data is accessed from other applications.
- Telephony Framework: It manages all voice call related functionalities.
- Location Manager: Allows access to location related services using GPS or cell tower.
- Resource Manager: It manages various resources used by the application and provides access to non-code embedded resources such as strings, color settings and user interface layouts.
- Notifications Manager: Allows the application to display alerts and notifications to the user.

The blocks included in Application framework are the blocks with which the application directly interacts and can be manipulated by developers and manufacturers to meet the needs of any application. [27],[28]

2.2.4. Application layer

The Application layer is the top layer of the Android stack. It is the layer with which the end user will interact mostly. This layer includes pre-installed applications by Google, as email client, calendar, browser, maps and contacts which are all written in Java programming language. However, a lot of manufacturers manipulate this layer and its pre-installed application to match their signature features or better serve and match their user interface esthetics. [27]

2.3. Limitations and Security

As mentioned earlier, Android OS runs on a Linux Kernel and so Android's security model takes advantage of the security features that the Linux Kernel provides. Due to Linux being a multi-user operating system, the kernel can isolate user resources in similar fashion as it isolates processes. [29],[30]

Android OS follows similar approaches for the internal security, however there are a few differences compared to those of a traditional Linux OS. In Android OS, a user-ID is assigned automatically to each application at installation and is later used to run that application in a dedicated process. In addition to that, each application is given a dedicated data directory to which only the application itself has permission to read or write. Thus, by running each application in a dedicated process and by assigning to each application a dedicated directory, the applications are isolated. This isolation is also known as sandboxing. [31]

Moreover, Android OS uses a user-based permission model. By default, an application can access only a limited range of system resources. This limitation can prevent application to maliciously use the system's internal API's. If an application needs to access internal resources, it has to request permission from the user and if the user accepts the request, the application can then access the internal API's. For this reason it is more than wise to check permission requests of an application, before we install it, and to download trusted applications only from Google official PlayStore. [31]

In addition to Application Sandboxing and user-based permissions, Google's Android OS implements cryptographic API's to allow secure information transfers between application and OS. These API's include basic cryptographic standards such as RSA, DSA and SHA keys as well as higher level protocols such as SSL and HTTPS. [30]

Communication between the Mobile Application and the embedded system

In this Chapter the author introduces the reader to the consideration that had to be taken for the selection of the communication technologies and briefly introduces the concept of each of the selected communication technologies. In addition, the Chapter includes a short introduction to required security parameters of the communication channels and further explanation of the Secure Shell Protocol, used for the secure communication of the devices.

3.1. Communication Technologies

Before we started designing the application and the ways via which it will communicate with the embedded system, we had to make a few considerations.

Firstly, the goal of the thesis was to be able to establish a communication remotely and so we had to discard technologies such as Ethernet or USB connectivity. The system simulates a home automation scenario where the user can access the embedded device from anywhere within the home residence, however the use of cabled communication technologies would limit the mobility of the user and the need of extra cables would increase the value of the end product. Therefore we focused strictly on wireless communication technologies.

In addition to increasing user's mobility and minimizing installation costs, we also had to consider wireless communication technologies that are broadly used and are supported by most of the Android devices and embedded systems. As a result we came up with the four mostly used wireless communication technologies, which are ***Wi-Fi***, ***Bluetooth***, ***NFC*** and ***Cellular Mobile Data***.

3.1.1. Wi-Fi

Wireless Fidelity, or as it is widely known *Wi-Fi*, is a technology used for wireless local area networking (WLAN). It is the most common and widely used method for internet networks and it can be found nowadays anywhere, from homes and offices to airports and restaurants, even to some public transportation. A simple diagram of such a network is shown in the Figure 3.

Within such a wireless LAN, a user can connect and exchange information with any other device within the network. Such connections can be achieved in two different fashions, the one being through an access point or router and the other one as a peer-to-peer connection.

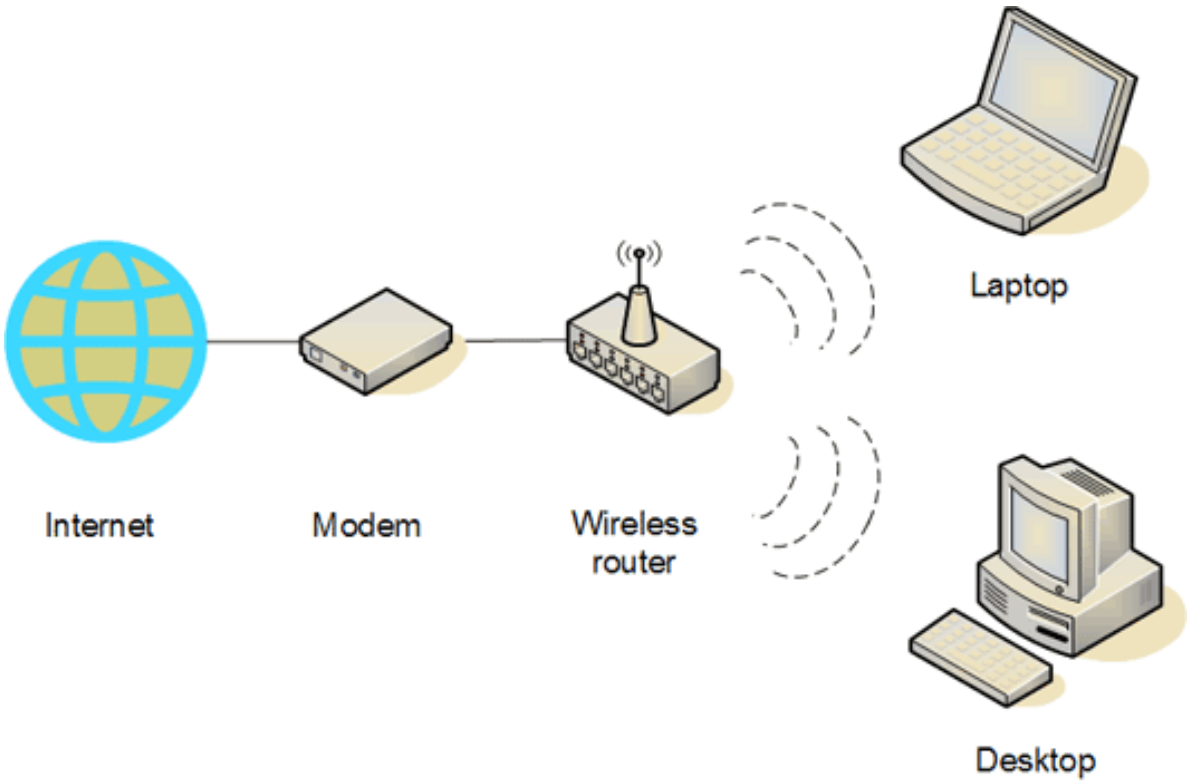

Figure 3. WLAN diagram **[57]**

Wi-Fi technology covers the IEEE 802.11 standards for radio communications, which is a broader set of standards of wireless communication themselves, as well as security aspects, quality of service and the like. The standards are categorized as 802.11 with a letter suffix. Few of the most widely known standards are:

- 802.11a – Wireless network bearer operating in the 5 GHz ISM band with data rates up to 54Mbps.
- 802.11b – Wireless network bearer operating in the 2.4GHz ISM band with data rates up to 11Mbps.
- 802.11g – Wireless network bearer operating in the 2.4GHz ISM band with data rates up to 54Mbps.
- 802.11n – Wireless network bearer operating in the 2.4and 5GHz ISM bands with data rate up to 600Mbps.
- 802.11ac – Wireless network bearer operating below 6GHz to provide data rates of at least 1Gbps for multi-station operation and 500Mbps on a single link.

- 802.11ad – Wireless network bearer providing very high throughput at frequencies up to 60GHz.

Security-wise however, Wi-Fi networks perform poorly. Security standards used for the protection of Wi-Fi networks are, WEP, WPA/WPA2 and 820.1X or also known as RADIUS. WEP, is the first security standard used by Wi-Fi technology and it is widely used even nowadays, however its limited security protection makes a choice only for home networks or older devices that don't support newer security standards. WPA, was designed to replaces the WEP standard and thus is mostly used for home networking and small businesses. WPA2 is a more secure version of WPA and is the security standard used from every newer wireless device. The 802.01X standard on the other hand, is the most secure standard of all the Wi-Fi standards, however it requires additional expertise to set up and maintain. Thus, 802.1X is intended to be used by larger businesses. All these security standards are based on utilization of wireless encryption key mechanisms, where the user who requests access to the network is authenticated with the use of a long sequence of hexadecimal numbers. [32],[33]

3.1.2. Bluetooth

Bluetooth is another wireless technology standard, used for the exchange of data between devices over short distances. Like, Wi-Fi technology, Bluetooth is used to form networks to, however in much shorter ranges than Wi-Fi. These networks are known as WPANs, which stands for Wireless Personal Area Networks.

The IEEE standardized Bluetooth as IEEE 802.15.1, but no longer maintains the standard. The Bluetooth standard is maintained by Bluetooth SIG which oversees development of the specification and manages the qualification program. [35]

Bluetooth operates in same radio frequencies with Wi-Fi and other radio communications; however they differ in the communication schemes. Bluetooth form a network, also referred to as piconet, and establishes connection between devices in a master-slave relation. There is a master device, which at an instance can be connected with up to seven slave devices simultaneously. The slave devices on the other hand, cannot be connected with more than one master device at a time. The master device chooses which slave device to address typically in a round-robin fashion. Device can change roles upon agreement and a slave device can become master device and vice versa.

Bluetooth technology differs from other wireless technology also in the layer protocol architecture. The Bluetooth's protocol stack consists of core protocols, cable replacement protocols, telephony control protocols and adopted protocols. The mandatory protocols, for all Bluetooth stacks are LMP, L2CAP and SDP. The Link Management Protocol (LMP) is the logical layer of the Bluetooth protocol stack and it is used for set-up and control of the radio link between two devices. [34]

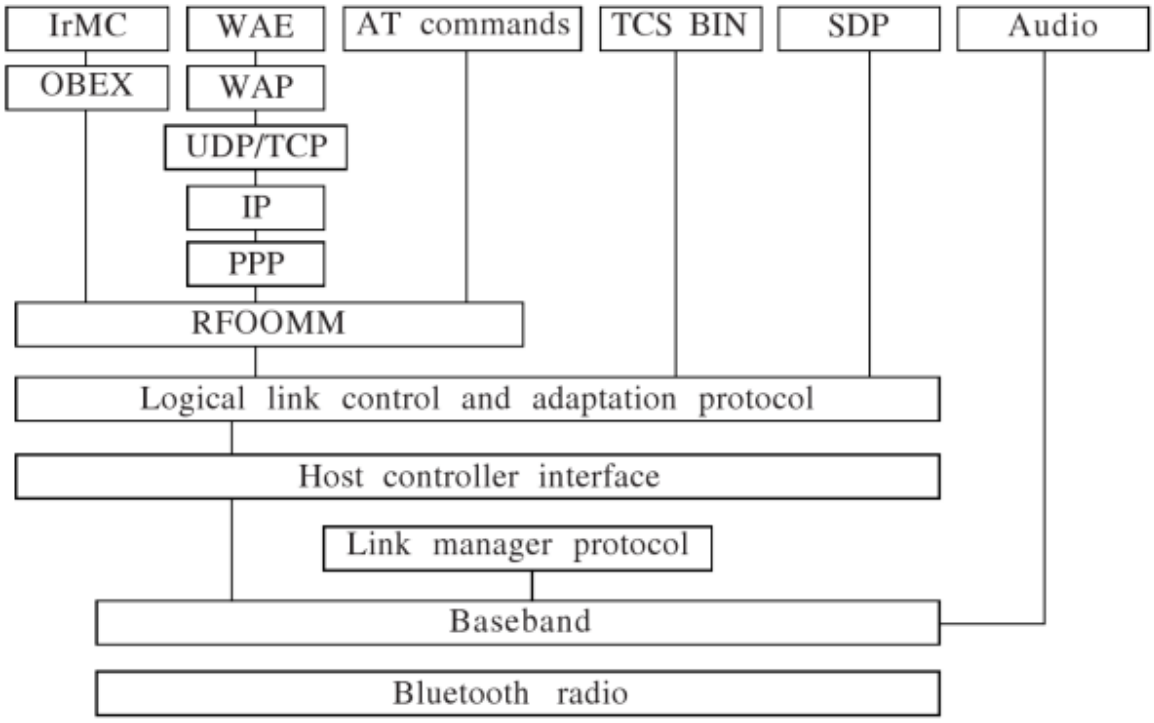

Figure 4. Bluetooth protocol stack **[25]**

The L2CAP is used to multiplex multiple logical connections between two devices, using different upper level protocols. The SDP allows a device to discover services provided by other devices, and their associated parameters. SDP then determines which Bluetooth profile the device can use and the protocol multiplexer settings needed.

Another mandatory protocol is Bluetooth Network Encapsulation Protocol. BNEP is used for transferring another protocol stack's data via an L2CAP channel. Its main Purpose is to transmit IP packets in the PAN profile. BNEP performs a similar function to SNAP in WLAN.

For the secure communication of devices, Bluetooth technology uses a PIN authentication and key derivation with custom algorithms. Bluetooth key generation is based on a PIN, which must be entered to the devices upon pairing of devices. [34].[35].[36]

3.1.3. NFC

The NFC, or Near-Field Communication, is a set of communication protocols that allow devices to communicate, by bridging them within a very short distance. NFC employs electromagnetic induction between two loop antennas when NFC-enabled devices exchange information. It operates within the unlicensed radio frequency ISM band of 13.56 MHz on ISO/IEC 18000-3 air interface at rates ranging from 106 to 424 kbit/s.

The NFC technology has many applications, such contactless payment systems, or transfer or media files between devices, however NFC offers a low-speed connection and thus is not optimal choice for transfer of big files.

NFC standards cover communication protocols and data exchange formats and are based on existing RFID standards, including ISO/IEC 14443 and FeliCa. Many different organizations try to standardize the NFC technology with NFC Forum being the major one. Other organizations such as GSMA, StoLPaN and ISO/IEC standardized NFC technology, however their standards are not universally accepted. [37],[38]

3.2. Security Parameters

So far, we have dealt with the security of the operating system itself and the security mechanisms that each of the communication technologies under our scope provide, for the secure transfer of data within a network. However, since the goal of the thesis is the remote control of the embedded system and execution of some security related commands with the use of the mobile application, we were required to use an additional security protocol. For that matter, we have chosen the ***Secure Shell*** protocol, or else known as ***SSH***.

3.2.1. Secure Shell

The SSH protocol is a method for secure remote login from one computer to another, and it is a secure alternative to the non-protected login protocol Telnet.[41] The SSH protocol finds use in many application due to its capabilities to provide secure access for users in automated processes, capabilities for remote issuing of commands and secure file transfers.

The Secure Shell protocol is based on a client-server communication model and it consists of three major components:

- The Transport Layer Protocol
- The User Authentication Protocol
- The Connection Protocol

The Transport Layer Protocol is typically being run over a TCP/IP connection, although it might also be used on top of other reliable data streams. In this layer the server-side authentication takes place. For this procedure the server generates a pair of public-private keys which will be used by the client to authenticate the server. The public key is shared with the client, and is referred to as the host key. The client must have knowledge of the host key in advance, for the authentication to be possible. According to [44], there are two alternative trust models that can be used from the client to acknowledge the host key. Either the client should maintain a local database that associates each host name with the corresponding public

host key or a Certification Authority (CA) technique has to be used for the host name-to-key association. In the latter, the client knows only the CA root key and can verify the validity of all host keys certified by the accepted CAs [40]. [41],[42]

After the successful exchange of public key and the acknowledgement of the host key from the client, the public key is used to match the private key maintained in the server. If the match is successful, the server authentication process is completed successfully.

Following the Transport Layer Protocol is the User Authentication Protocol. As its name reveals, this protocol is responsible for the authentication of the client-side from the server. In [46], the author suggests that there are three methods for the user authentication process, those are ***Public Key Authentication***, ***Password Authentication*** and ***Host-based Authentication***. From all three methods, the Public Key Authentication is the safest one and thus is strongly suggested for any client authentication process. This method works by having the client send a signature created with its private key. The server then checks if the key is valid for the client and the validity of the signature. If both hold the authentication request is accepted. The generation of the signature associated with the client's private key is achieved by the user supplying a passphrase.

Despite Public Key Authentication method being the most secure option, not all SSH connections require this method for authentication. A simpler alternative is the Password Authentication method, which requires the user to supply a password on the client-side, which is then used to the server-side, for authentication.[42], [43]

The least used method from the above mentioned methods is the host-based. The authentication is based on the hostname of the client and the username of the server. For this authentication method, the client send a signature associated with its private key, which then the server checks with that client's public key. Once the client's identity is established, authorization is performed based on the username stored on the server and the one supplied by the client's side and the hostname of the client. [43]

The last component of SSH's protocol stack, is the Connection protocol. This protocol runs on top of the Transport Layer Protocol and the User Authentication Protocol and it provides interactive login sessions, remote execution of commands as well as TCP/IP and X11 forwarding [47]. The Connection Protocol uses the prior secure authentication connection established from the previous two protocols, to multiplex a number of logical channels. The lifecycle of these channels is divided in three stages, ***Opening a channel***, ***Data transfer*** and C***losing a channel***. The side that initiates a channels request sends a message containing the channel type, sender channel, initial window size and maximum packet size. The channel type identifies the application that the channel is going to be used for, whereas the sender channel corresponds to the local channel number. At this point it is useful to mention that each side of the connection may run multiple channels simultaneously. Thus each side, associates a unique

channel number to each one of the channel to distinguish one channel from the others. The rest of the parameters specify the number of bytes of channel data that can be sent to the sender without adjusting the channel window, and the maximum size of an individual data packet that can be sent to the sender, respectively. If the remote side is able to open the channel, the data transfer is performed. Otherwise the remote side sends a message back to the sender including a reason code indicating the reason of failure. [44]

During the data transfer phase of the channel, data is transmitted from client to server and vice-versa. If each of the two side wishes to close the channel, it has to send a "close" message to the other side, informing that the channel will close.

In [47], there are four channel types recognized. Namely they are, ***Session***, ***X11 forwarding***, ***Forwarded-TCP/IP*** and ***Direct-TCP/IP***. The session channel type is used for the remote execution of a program. The program may be a shell or an application or a system command. The X11 refers to the X Window System, which is a computer software system and network protocol, which allows application to run on a network server but be displayed on a desktop machine [40]. The rest two of the four channel types are used for global and local port forwarding respectively.

Following is Figure 5, which depicts a visual representation of an SSH connection between a client device and a server device, as well as the information the two sides exchange to set-up a connection authenticate each other and transfer data.

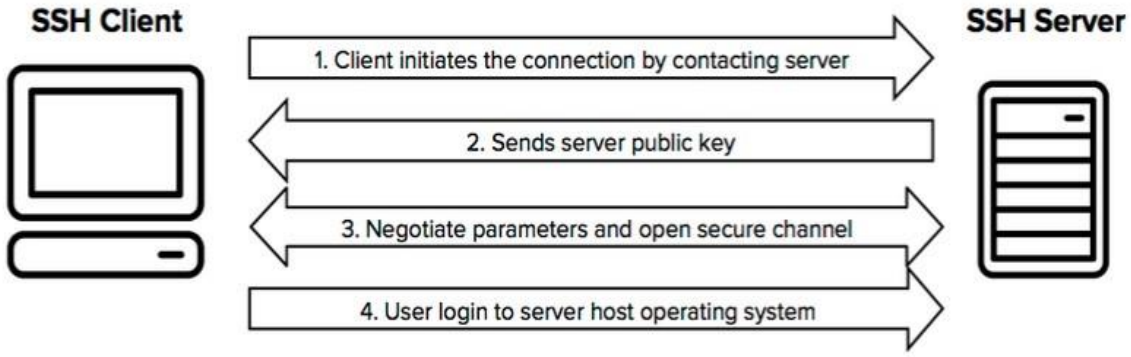

Figure 5. The Secure Shell connection model. **[41]**

Since one of the goals of this thesis is to remotely access an embedded system and execute some commands, the implementation of SSH protocol was essential. A deeper dive into the implementation of the SSH protocol is provided in the following chapter under the subsection ***4.3 Communication Scenario***, where we also included programmatic implementation of some of the procedures mentioned above.

Implementation of the Android Application

The initial goal of the thesis is to create a simple, user friendly application to test device's available communication technology and to remotely access and control and embedded system device. In order to implement our ideas and create the actual application we had to follow the standard procedure that every developer follows: we choose a development environment, our devices needed to achieve our goal and we began implementing.

4.1. Environment and devices

For the development of the application and the followed communication of the device itself with the selected embedded system, the required tools and devices we had to use are the developer environment for the implementation of the application, a smartphone device to run and test the application and the embedded system.

4.1.1. Android Studio IDE

The environment used for the implementation of the application, was Google's Android Studio IDE. We used the latest version of the software which is version 3.0.1. This IDE is based on JetBrains' IntelliJ IDEA software and it is designed specifically for Android development. Google provides a lot of features within this platform with some of the most important being [49]:

- Gradle-based built support
- Built-in support for Google Cloud Platform
- Android Virtual Device (Emulator)
- A rich layout editor
- Support for building Android Wear applications
- Code templates and GitHub integration
- Extensive testing tools and frameworks
- Lint tools to catch performance, usability, version compatibility and other problems

When building an Android project using Android studio, the developer will come across with a screen like the one depicted in Figure 6. By default, on the left side of the screen there is a drop-down list which contains all the source files of the project. The list is divided into two main categories. The first of the two is the Application category, which contains all the source files of the project, such as the ***AndroidManifest.xml*** file, the ***Java*** classes used in the project and the resource files which include the layout of the project, string values, menus and others.

The second category of the list is the Gradle Scripts which contains all the Gradle related files, such as the build.gradle, settings.gradle and some property files. Android studio uses Gradle as the foundation of the build system. This build system runs as an integrated tool from the Android studio menu. It is used to customize, configure and extend the build process, to create multiple APKs for the application and reuse code and resources across source-sets [49].

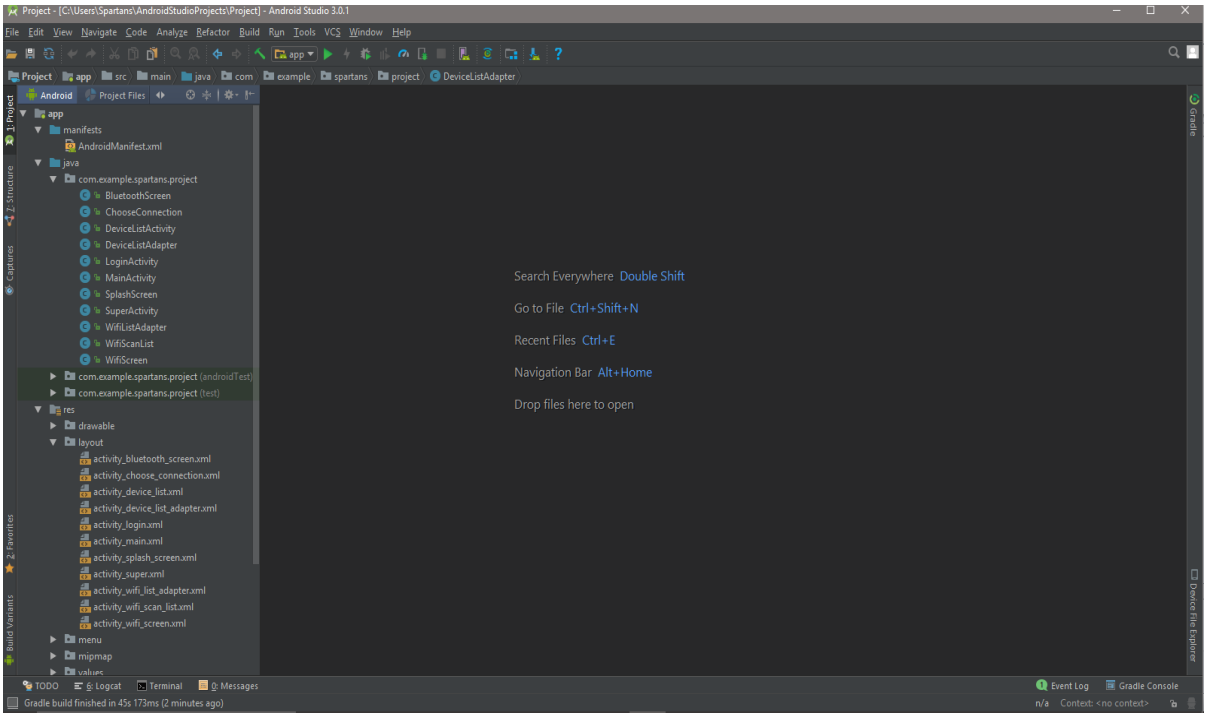

Figure 6. Android Studio Workplace

As we have mentioned above, the Application category contains all the source files of the project. The first source file that falls under this category is the AndroidManifest.xml file. This file is responsible to provide the Android system with the essential information of the application, which they have to be known before the system can run the application's code. In addition to providing the system with the needed essential information, AndroidManifest.xml file is responsible for other processes also:

- It names the Java package for the application.
- It describes all the components of the application, such as the activities, services, broadcast receivers and content providers. These declarations inform the Android system of the components and the conditions in which activities can be launched.
- It declares permissions which the application must have in order to access protected parts of the API and interact with other applications or access device's features.
- It lists the libraries that the application must be linked against.

- It lists the Instrumentation classes that provide profiling and other information as the application runs.
- It declares the minimum level of the Android API that the application requires.

As its indexing reveals, AndroidManifest file is written in the XML programing language.[50] The second group of source files under the Application category is the Java classes. The Java classes define the functionality of the application as a whole. Generally, the Java classes include the functionality of each of the activities of the application and define the corresponding layout file which will be attached to any of the activities. The Java classes is where the developer will implement the functionality of the application and he will include all the necessary functions and features of the application.
The final group of the Application category is the Resource files. In this group of files the resources of the application are stored. Resources such as string values, menus and layouts are the main component of the Resource files group. The layout files are responsible for the design of the UI of the application. They are written in XML language and they are attached to some Java class for their functionality. The values resources contain different values set by the developer, for visual features of the application. Also under the resource files falls the drawable files, where they contain built-in icon or custom ones where they can be used for aesthetical purposes in the UI of the application.[53],[54],[55]

4.1.2. HTC Desire C

The smartphone device selected to run and test the functionality of the application, was HTC's Desire C. The device was selected because it fulfilled the basic requirement given by the functionality of the application.

HTC's Desire C runs on Android Ice-Cream 4.0.3 MR1, which is based on Linux's modified Kernel 3.0.16. The device fulfilled also all of the required communication capabilities, as it is Bluetooth and NFC enabled and its hardware included both cellular and Wi-Fi antennas.

Figure 7. HTC Desire C **[51]**

However, due to the fact that HTC Desire C runs on a relatively old version of Android's Operating System, we had to make several compromises during the development of the application, since many required functionalities are not supported on older OS verisons.

4.1.3. Raspberry Pi 3B

For our embedded system we have selected the Raspberry Pi 3B, which is the latest version of the well-known Raspberry Pi system. Since our goal is to test and evaluate wireless communication means, we needed a device with the respective capabilities. This version of the Raspberry Pi includes an on board wireless LAN adapter and a Bluetooth Low Energy (BLE) module, and there was no further need of extra adapter or dongles in order to match our requirements.

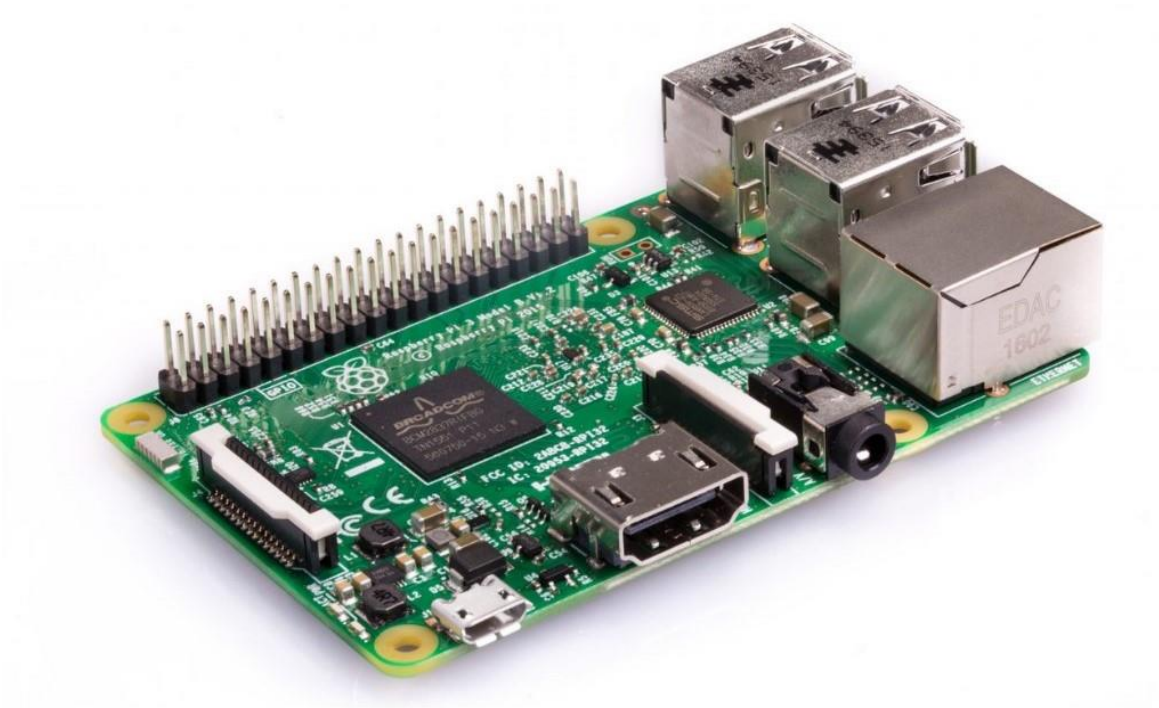

Figure 8. Raspberry Pi 3B **[52]**

This was our embedded system of choice due to the fact that from all other available embedded system devices used for similar project, Raspberry Pi 3B was the one that required the less set-up and configurations out of the box, to match our needs.

4.2. User Interface and Functionality of the Application

The application is an implementation of an SSH client and a connectivity manager infused together. It included three main activity classes, where the function and the logic of the application take place, and five layouts files responsible to create the UI of the application.

Following in Figure 9, is a representation of the Lifecycle of the activity and the transition from one activity to another within the application.

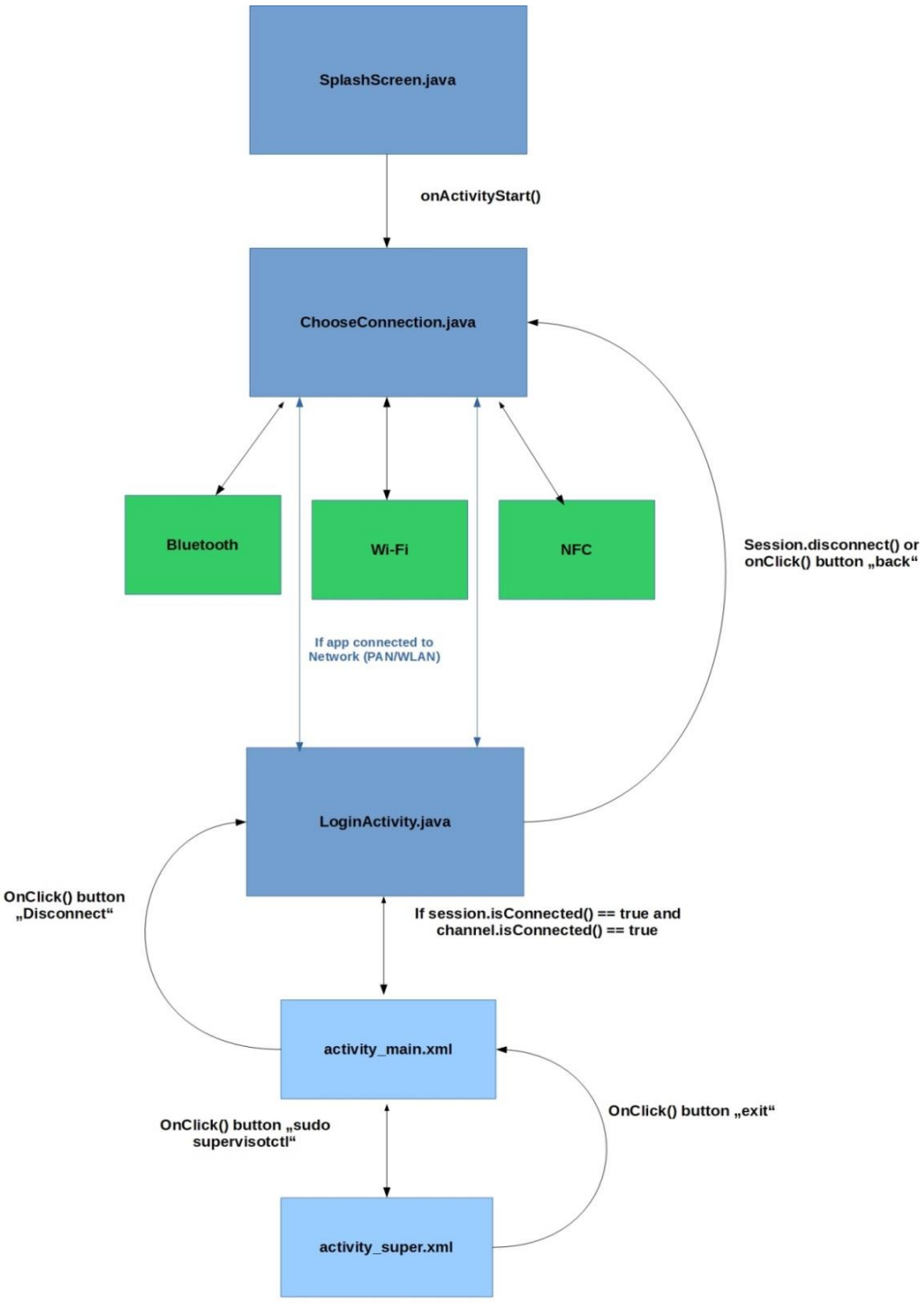

Figure 9. Lifecycle of the Application

When the user launches the application for the first time, the onStart() process takes place and the systems starts the application. While the OS loads the components of the application, the

user is introduced to a splash screen, which serves as a time window for the set-up of the application. This is the SplashScreen.java class and is depicted in Figure 10.

Figure 10. SplashScreen.java - Launcher Display of the Application

After the loading of the application is finished, ChooseConnection.java activity starts, with its respective layout being displayed on the screen for the user to interact with. It contains four buttons, and one TextView field, as shown in Figure 16. However, as depicted in Figure 11, initially they are only three of the buttons displayed on the screen. The fourth button, which is labeled “NEXT”, will be displayed only after the user is connected to a network either vie Bluetooth connectivity or via Wi-Fi.

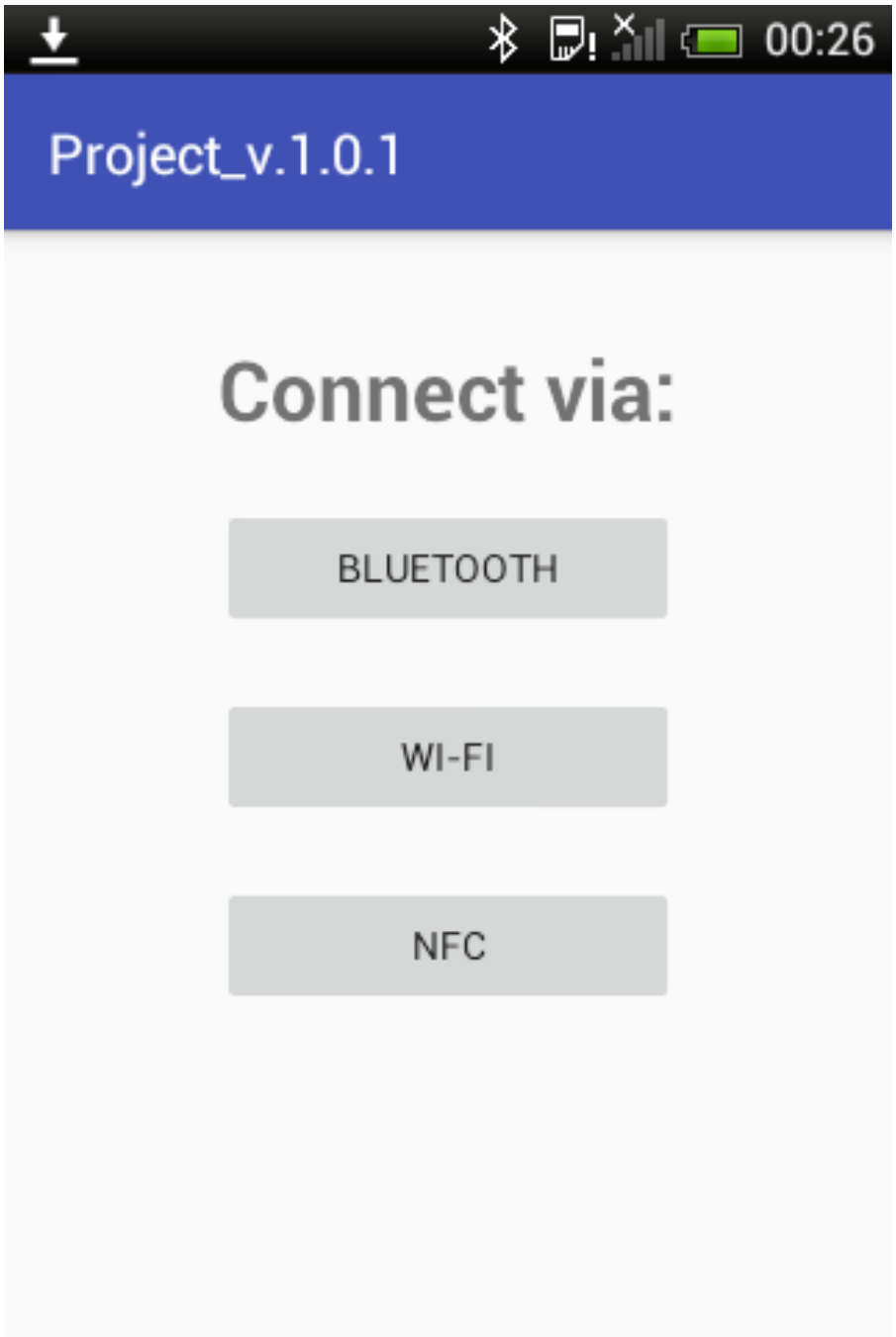

Figure 11. ChooseConnection.java activity's UI

Each button is labeled with its corresponding functionality and the TextView field suggests the user to click one of the three grey buttons, in order to choose connectivity type. When the user clicks on the “Bluetooth” button, the device’s Wireless setting screen is displayed as shown in Figure 12. While on the Wireless settings screen, the user can turn on or off the

Bluetooth connectivity of the device and also enable Bluetooth Tethering capabilities on the device.

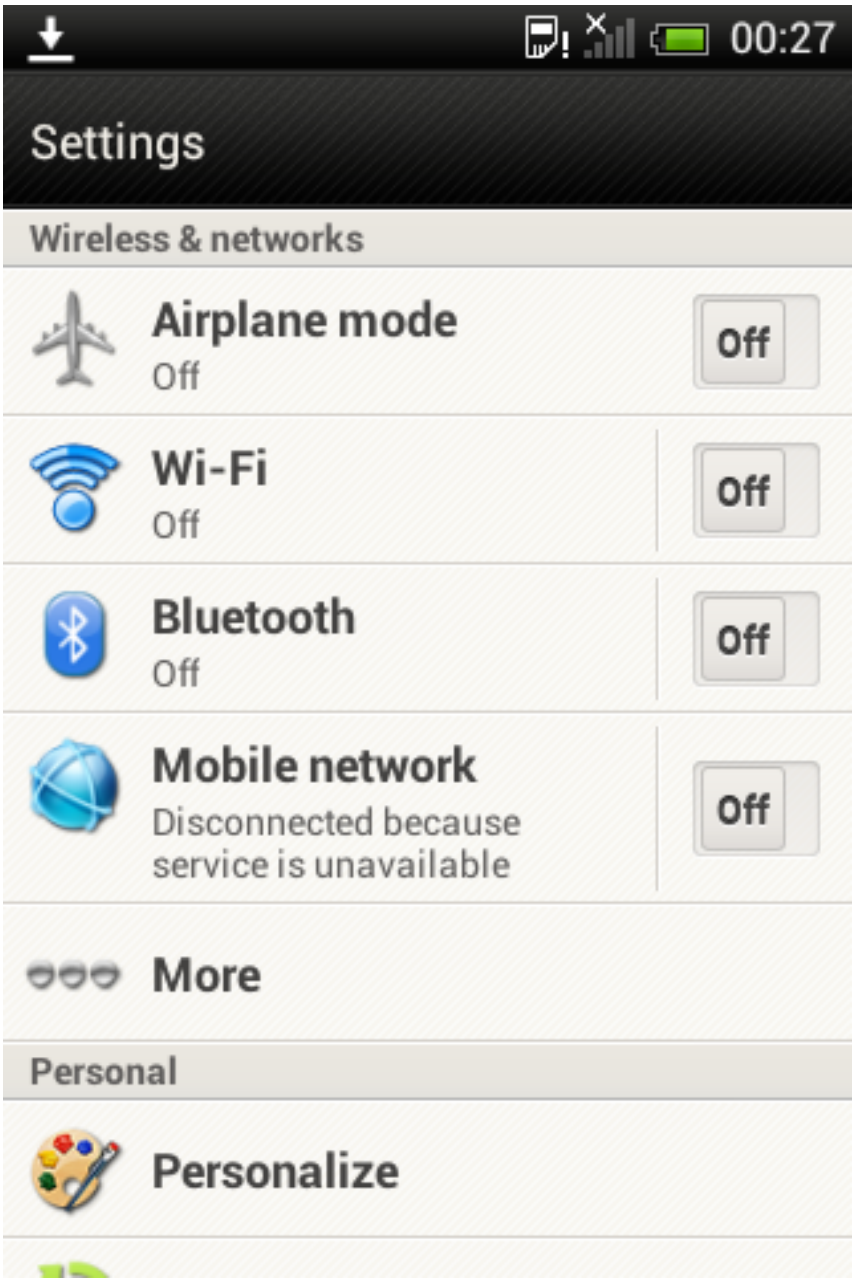

Figure 12. Wireless setting screen

If Bluetooth connectivity is enabled, the user is capable to see the paired devices that has previously paired the phone with and can choose to scan for new devices available in the area. By clicking on one of the devices listed in this view, the user initiates a connection with that device, over Bluetooth. Figure 13 illustrates the device's Bluetooth functionality.

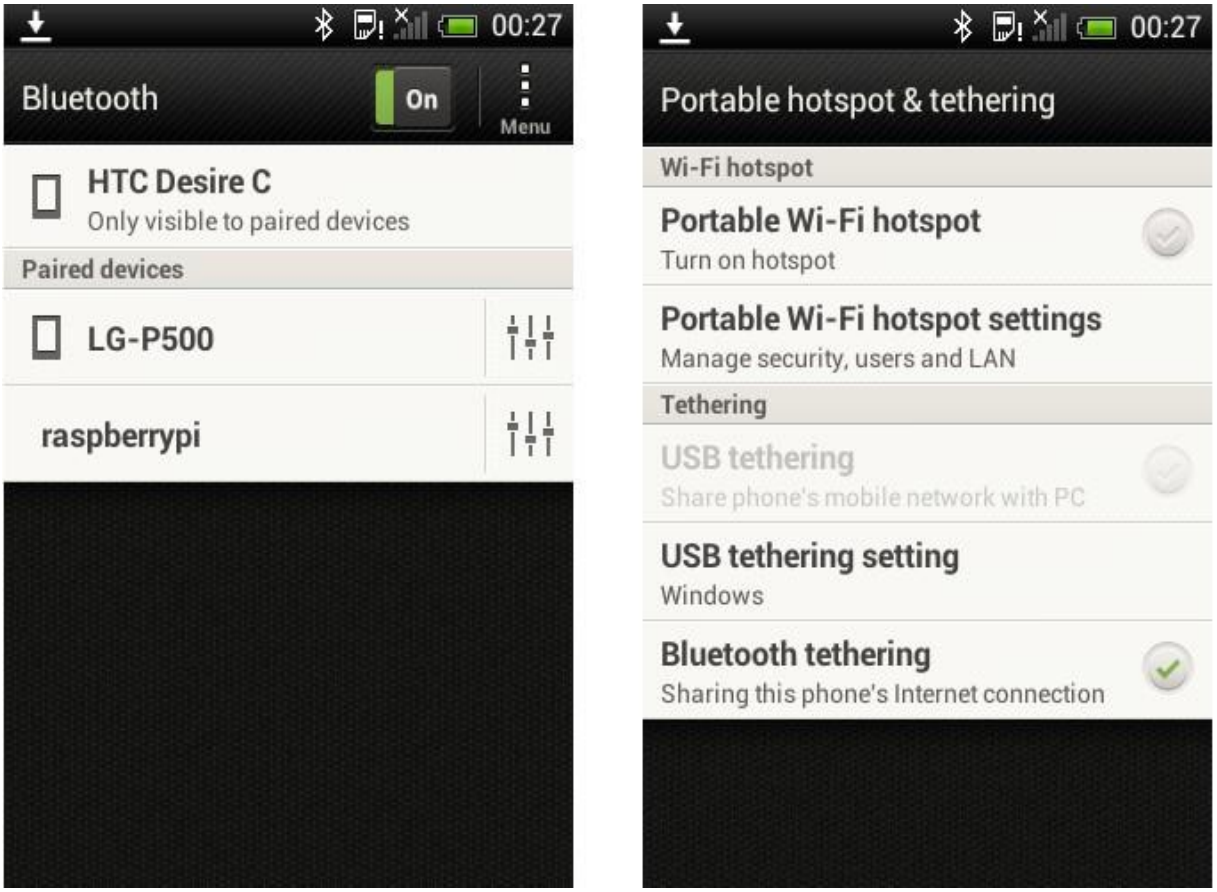

Figure 13. Bluetooth Settings screen - Paired Devices (left) Bluetooth Tethering (right)

In similar fashion, the user can access the Wi-Fi connectivity settings, by clicking on the "Wi-Fi" button. Once, the "Wi-Fi" button is clicked, the device's Wi-Fi settings are displayed on the screen. A list of available networks is displayed and the user can choose to which network he wishes to connect to. Once he clicks on a network item from the list, if the network is not secured, the device will connect to that network without the need of any credentials. However, if the user chooses to connect to a network, which is secured by a password a dialog window will prompt the user to type a password for authentication. After the authentication is finished the device will be connected to the corresponding network. Figure 15 displays the User Interface of the device's Wi-Fi setting.

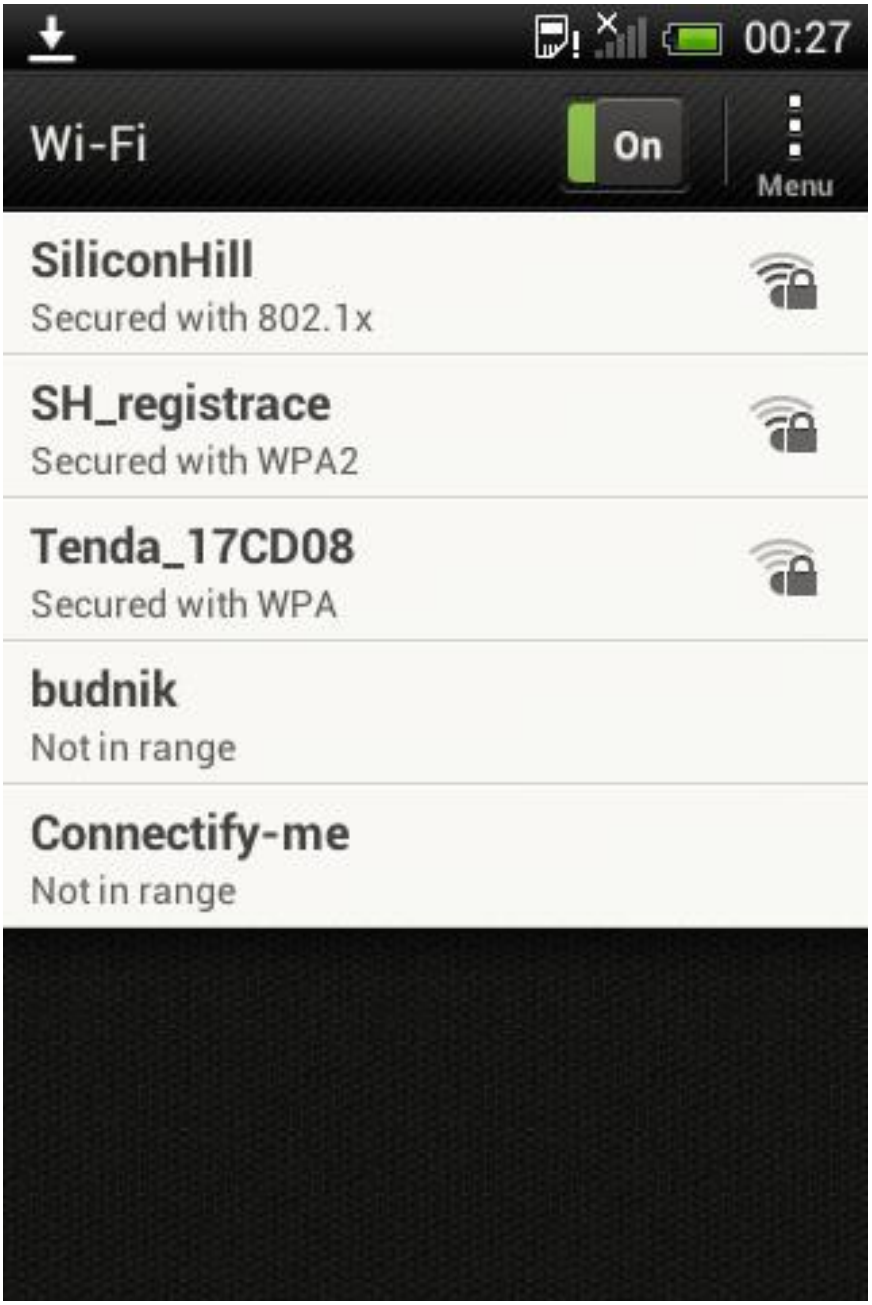

Figure 15. Wi-Fi Settings screen

Finally, once the device is connected to some network, the user is returned to the ChooseConnection.java activity, where now the "NEXT" button is available for the user to click and proceed to the next activity of the application. Figure 16 shows the ChooseConnection.java activity, after the device is connected to a network.

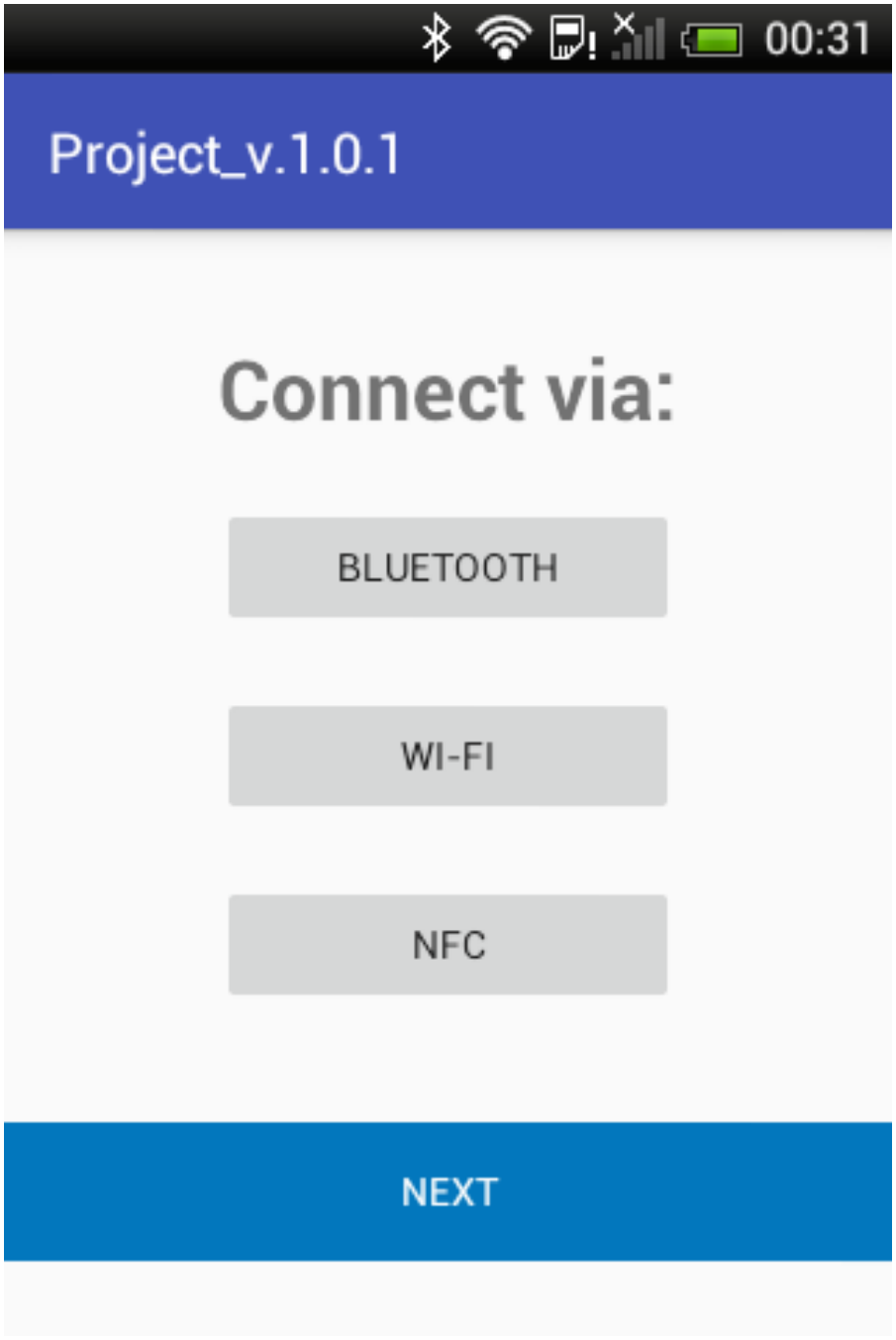

Figure 16. ChooseConnection.java - Display of connected device.

In this instance of the ChooseConnection.java activity, the user is allowed to click the "NEXT" button and trigger the onCreate() and onStart() methods of the LoginActivity.java activity. Moreover, the UI of the application changes with a new layout being applied on the screen and the corresponding LoginActivity.java activity is instantiated. In this activity, a connection to the internet is already established by the ChooseConnection.java activity and now the user is able to communicate with the remote embedded system, over an SSH connection.

As mentioned earlier in the subsection 3.2 of Chapter 3, the SSH connection follow a client-server model. In our case our application is the client and thus the side which needs to provide some credentials for the authentication process from the server. For this reason, the UI of the LoginActivity.java includes four EditText Views, where they are going to be used for the user to input the required credentials, as well as a button which will be clicked to initiate the connection.

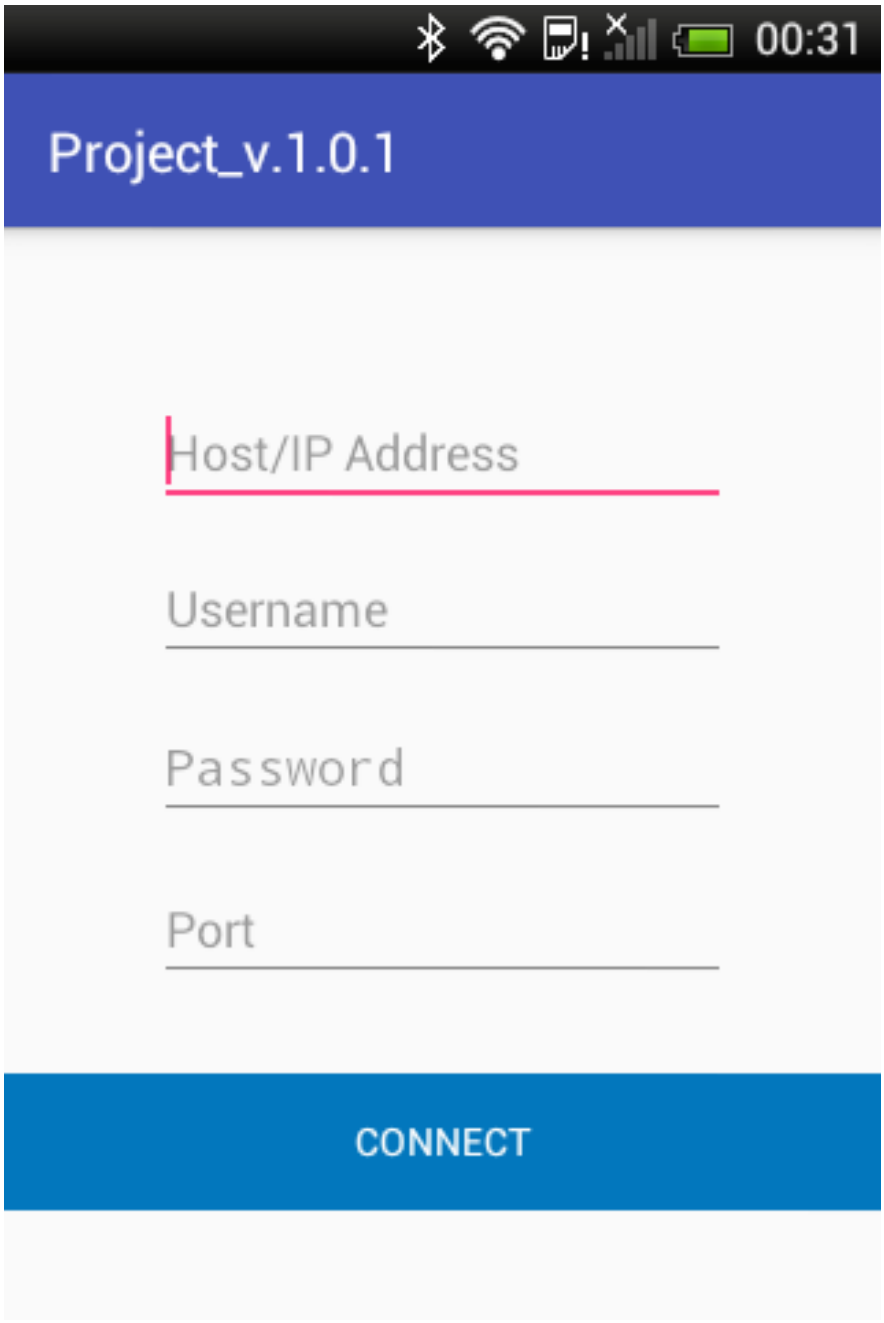

Figure 127. LoginActivity.java - UI screen

In Figure 17 we can see the UI of the LoginActivity.java screen. It consists of four EditTexts which are labeled according to the value that they pass in the java code, and a button labled “CONNECT” which the user clicks to start a connection. The first EditText, labeled as ”Host/IP Address” requires the user to input the hostname of the server he wants to access, or the IP Address of that server. The “Username” and “Password” EditTexts require the username of the client and the password with which he is trying to access the server. These two values, will be used on the server side for authentication and if they are correct they will grant access to the client. Finally, the “Port” EditText passes an integer value in the java source code of the application and is the number value of the server’s port which is listening for connection requests.

After all credentials are filled in, the user can click on the “CONNECT” button and trigger the connection process. However, if one or more of the EditText fields were left empty by the user or the input was incorrect, the connection will be declined by the server and the application will print out a Toast Message to inform the user about the error that occurred. Table 1, in subsection 4.3.1 shows a snippet of the source code which is responsible to handle this error.

Therefore if the credentials are incorrect or empty, the authentication procedure on the server side will fail and thus the connection will return false. That is, the application will not be able to proceed further and the user will be informed that an error occurred while trying to connect to the server. However, in case the credentials are correct and the authentication procedure is successful, a connection will be established between the application and the remote embedded system, this will grand the user with the ability to proceed on controlling the embedded system and run commands from within the application.

To do so, the application has to check if a connection is currently open and listening for data streams. If so, the application will chance its UI to another layout which will populate the screen with a few new buttons in a ScrollView. In Figure 18, we can see what this new layout will look like and what are the buttons that it includes.

Figure 138. Activity_main.xml - This is the layout which includes the command buttons

In the activity_main.xml layout we have included numerous buttons, which are responsive to the user's click. Each button is responsible to send a command to be executed on the server, when the user clicks on it. We have labeled each button with the corresponding command that is being send to the server in order to smoothen the user experience. Moreover, we have preferred to execute commands with button clicks instead of user's input type approach since it adds more simplicity and usability to our application.

The UI of the activity_main.xml layout is encapsulated inside a ScrollView, which allows the user to scroll on the screen and choose the command he wants to execute. It is easily noticed from Figure 18, that there are two slightly different buttons included inside the layout. The one is the "DISCONNECT" button and the other one is the "SUDO SUPERVISORCTL" button. In the case of the "DISCONNECT" button, when the user clicks on it, the application disconnects the channel which runs the commands as well as the session which is currently connected. That is, the application will terminate the SSH communication between the device and the embedded system and it will return the user to the activity_login.xml, which is the layout responsible for credential inputs. For the "SUDO SUPERVISORCTL" button, if the user clicks on this button a new process will be triggered. This button is responsible to start the supervisor set of commands, which is a different set of commands that the user has access to right now. This will result in a new layout being instantiated and displayed on the screen.

This new layout is of similar fashion with the activity_main.xml layout. That is, is a layout encapsulated inside a ScrollView, which contains buttons responsible to execute supervisor commands on the server end. The design of the new layout is depicted in Figure 19.

Figure 149. Activity_super.xml - Supervisor commands layout

This layout contains only the buttons that run supervisor commands, and there is not functionality implemented that affects the application directly. If the user wishes to exit the supervisor commands and go back to the main command thread, he has to click on the button "EXIT", which will exit the supervisor command set and bring back on screen the activity_main.xml layout.

The whole application is terminated once the user totally exits the application. This can happen as a response to the user's click on on-screen home button or the user exiting and then killing the app. Nevertheless, if the user remains inactive during a session for longer time interval than 83 minutes, the session automatically will disconnect which will result to the channel termination and the application returning on the credential input screen activity_login.xml layout. If the user wishes to reconnect to the server again, he should instantiate a new connection following the same procedure, with passing credentials and authenticating, from the beginning.

4.3. Communication Scenario

In earlier chapters, we have mentioned that our main goal of our application is to be able to access and control an embedded system device remotely. We have talk so far about the application user interface, the functionality of the application as well as the application lifecycle. The following chapters will describe the programmatic approach to achieve a secure communication, over the selected communication technologies and an insight explanation of how our application handles these processes.

4.3.1. Communication of the Application and Raspberry Pi

As we have mentioned before, for the remote control of the Raspberry Pi we need to implement an SSH connection, between the Android Application and the Raspberry Pi. Since the SSH communication is a client-server based communication, we had to establish the roles of each device for our communication scenario. Therefore in our communication scenario, the android device is the client requesting connection and the Raspberry Pi serves as the server responsible to grand access to the client.

In order to add SSH client capabilities to our application, we used the open-source Java Library JSch, implemented by JCraft development company, and we implemented a clients to match our requirements for the SSH connection. This library includes all the required elements for the client-end of our communication, such as SSH protocol, User Authentication, SFTP channels and remote control capabilities.

Since we need to remotely control the Raspberry Pi system, we had to instantiate the following parameters in our application:

- **JSch** – Is a java Object which acts as the main repository to extract all required features for our application's functionality
- **Session** – Is a java Object derived from the JSch Object which is responsible to handle a session for the connection of our app with the Raspberry Pi
- **ChannelExec** – Is one of numerous types of channels supported by JSch SSH implementation and is mandatory for the execution of commands on the remote server
- **OutputStream** – Is the output data stream of our device, which will contain the commands to be executed on the server side in the form of bits
- **InputStream** – Is the input data stream of out device, which will contain the response of the server after we execute a command

As we have mentioned earlier, the user needs to input the credentials required from the server side in order to set-up a connection. This process takes place in the LoginActivity.java activity. The user fills out the EditTexts presented on the UI, with the requested information for each EditText. Table 1 shows the programmatic implementation of this process.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_login);

    Host_ip = findViewById(R.id.Host__ip); // IPAddress or Hostname of the
server
    Password = findViewById(R.id.Password); // Password used for the
authentication of client
    Username = findViewById(R.id.Username); // Username of client to
associate with password for future connection
    Port = findViewById(R.id.Port); //Port number to which communication
socket will be created (default value: 22)
    Connect_btn = findViewById(R.id.Connect_btn);

    Host_ip.setOnClickListener(new View.OnClickListener(){
        @Override
        public void onClick(View v) {
            strHost_ip = Host_ip.getText().toString();
            if(TextUtils.isEmpty(strHost_ip)){
                Host_ip.setError("Please enter a valid ID Address.");
            }
        }
    });
    Username.setOnClickListener(new View.OnClickListener(){
        @Override
        public void onClick(View v) {
            strUsername = Username.getText().toString();
            if(TextUtils.isEmpty(strUsername)){
                Username.setError("Please enter a valid Username.");
            }
        }
    });
    Password.setOnClickListener(new View.OnClickListener(){
        @Override
        public void onClick(View v) {
            strPassword = Password.getText().toString();
            if(TextUtils.isEmpty(strPassword)){
                Password.setError("Please enter a valid Password.");
            }
        }
    });
    Port.setOnClickListener(new View.OnClickListener(){
        @Override
        public void onClick(View v) {
            strPort = Port.getText().toString();
            if(TextUtils.isEmpty(strPort)){
                Port.setError("Please enter a valid Port.");
            }
        }
    });
```

Table 1. Java Code Snippet - Input of SSH Session credentials

After all EditTexts are filled out the user presses the “CONNECT” button to initiate a connection. If all credentials are filled in the corresponding EditText and are correct, the

application will attempt to connect with the server. Instances of the JSch object and all its derived object are created and the attempt for connection is triggered, as shown in the snippet of Table 2.

```
Connect_btn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
       //Create new Thread to avoid Application Crashing on execution
                new Thread(new Runnable() {
                    @Override
                    public void run() {
      //Initiation of SSH connection using JSch SSH client library
      //Authentication procedure and establishment of session connection
                        strPort = Port.getText().toString();
                        int port = Integer.parseInt(strPort);
                        strHost_ip = Host_ip.getText().toString();
                        strUsername = Username.getText().toString();
                        strPassword = Password.getText().toString();

                        try {
                            jsch = new JSch();

                            session = jsch.getSession(strUsername,
strHost_ip, port);
                            session.setPassword(strPassword);

                            // Avoid asking for key confirmation
                            Properties prop = new Properties();
                            prop.put("StrictHostKeyChecking", "no");
                            prop.put("PreferredAuthentications",
"publickey,keyboard-interctive,password");
                            session.setConfig(prop);
                            session.connect(100000);

                        }catch (JSchException e){
                            e.printStackTrace();
                            Toast.makeText(LoginActivity.this, "There was
something wrong with the connection.", Toast.LENGTH_LONG).show();
                        }
                    }
                }).start();
```

Table 2. Java Code Snippet - Initiation of connection with the SSH server

At this point, the application has tries to initiate a connection passing all the credentials as parameters of the session object. If the connection is established successfully the application will continue with instantiating a Channel object, which is required for the remote execution of commands. On the other hand, if the connection to the server was not established, the application will print out a Toast Message to inform the user about the error occurred. Following in the Table 3, is a snippet of the source code for the above stated procedure.

```
if(session.isConnected()){
    setContentView(R.layout.activity_main); //Change the layout of the
Activity to enable command execution

    //Declare the buttons present on this activity layout
    button1 = findViewById(R.id.button1);
//This part of code includes additional buttons
    DisConnect_btn = findViewById(R.id.DisConnect_btn);

    //Initiate a channel for passing the commands to the server
    try {
        channel.getSession();
        channel = (ChannelExec) session.openChannel("exec");
        channel.connect();
        channel.setInputStream(bais);
        channel.setOutputStream(baos);
    } catch (JSchException e) {
        e.printStackTrace();
    }
```

Table 3. Java Code Snippet - Instantiation of channel object

Once the channel is created, as mentioned in subsection 4.2, the user is able to pass commands to the remote server for their execution. This is achieved by on-button clicks which in return will pass the corresponding command to the server. The onClick() events of the button are managed as described in the snippet of Table 4.

```
button1.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if (channel.isConnected()){
            channel.setInputStream(bais);
            channel.setOutputStream(baos);
            channel.setCommand("sudo supervisorctl");
            channel.disconnect();

            //Change the layout of the Activity to enable Supervisor
commands execution
            setContentView(R.layout.activity_super);

            //Declare buttons present on this activity layout
            button13 = findViewById(R.id.button13);
//This part of code includes additional buttons
            button22 = findViewById(R.id.button22);
```

Table 4. Java Code Snippet - onClick() event handler

In our code, “button1” represent the supervisor command, thus, when clicked it calls for the initiation of the activity_super.xlm which hold the UI and logic of the supervisor command set. This can be seen on Table 4, where the onClick() method initiates a change to the ContentView() of the application.

Following this call, the UI of the application is changed and the user can now execute commands from the supervisor command set. If the user wishes to exit the supervisor command set, then he should click the "EXIT" labeled button and the application will initiate a return to the previous layout, for the execution of system commands. This procedure is programmatically described on Table 5.

```
button20.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if (channel.isConnected()){
            channel.setInputStream(bais);
            channel.setOutputStream(baos);
            channel.setCommand("exit");
            channel.disconnect();

            setContentView(R.layout.activity_main);
        }else {
            try {
                channel.connect();
                channel.setInputStream(bais);
                channel.setOutputStream(baos);

                channel.setCommand("exit");
                channel.disconnect();

                setContentView(R.layout.activity_main);

            } catch (JSchException e) {
                e.printStackTrace();
            }
        }
    }
});
```

Table 5. Java Code Snippet - User exiting supervisor command set

Generally all onClick() events of the application, for the execution of commands are handled in the same manner. Every button checks is a channel is connected, if so it passes the command to the server, else the application will try to create again a new channel, and follow the same procedure to execute the command.

Finally, if the user wishes to exit the command execution completely and to close the session too, this can be achieved by clicking the "DISCONNECT" button on the activity_main.xml. The application will disconnect any active channel and will proceed to disconnect from the session. Once the session is disconnected the application will return to activity_login.xml and the user will have to re-enter its credentials to the corresponding fields is he wishes to reconnect to the server. Table 6 illustrates the process of disconnection from the server.

```
DisConnect_btn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if (channel.isConnected()) {
            channel.disconnect();
            session.disconnect();

            setContentView(R.layout.activity_login);
        }else {
            session.disconnect();
            setContentView(R.layout.activity_login);
}
    }
});
```

Table 6. Java Code Snippet - Disconnection from SSH server

4.3.2. Secure Shell over Wi-Fi

Before analyzing the results of the measurements of this thesis, it is worth mentioning the procedure that each communication technology required in order to allow our application to connect to the remote embedded system over SSH.

The Wi-Fi communication technology required a rather simple implementation. The server's port is set to listen for TCP/IP connection requests. Thus trying to connect to the embedded system from within a WLAN was pretty straight forward since the Wi-Fi protocol contains a TCP/UDP Transport Layer.

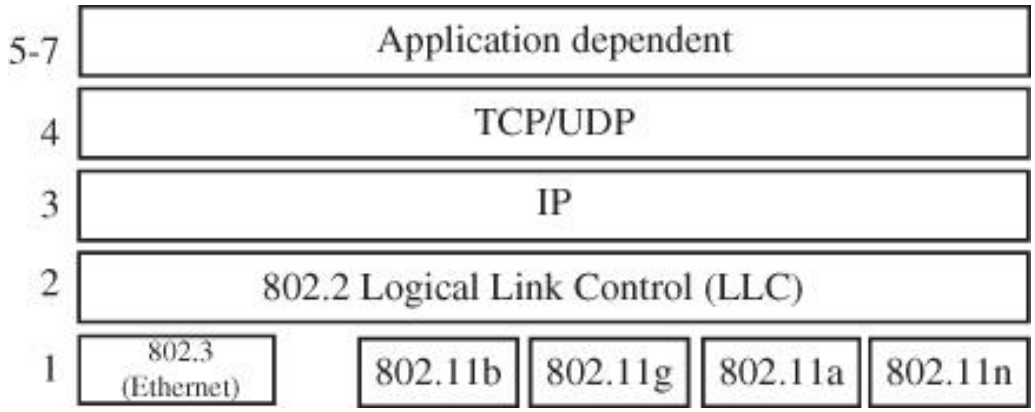

***Figure 215. WLAN Protocol Stack* []**

As we can see in Figure 15, the Transport Layer Protocol runs on-top of the Wi-Fi Physical layer protocol and so the initiation of a TCP/UDP connection is the standardized way of transmitting data over a WLAN network.

4.3.3. Secure Shell over Bluetooth

On previous subsection we saw how the WLAN protocol stack is constructed and that the Transport Layer of it is the TCP/UPD protocol. However, in the case of the Bluetooth technology the protocol architecture is a bit different. While Bluetooth protocol stack doesn't provide exactly the same transport layer as the WLAN protocol stack, it provides alternative protocols that can offer similar functionality with a TCP/UDP protocol. The equivalent protocol of Bluetooth stack for TCP is the RFCOMM protocol. Although it was designed to emulate an RS-232 serial port, RFCOMM can be used in similar manner as the TCP protocol. However, RFCOMM cannot connect to a TCP/IP port and thus it requires the set-up of another non-TCP/IP port or the port forwarding method. Moreover, there are several available software which allow TCP/IP ports to listen for RFCOMM connections.

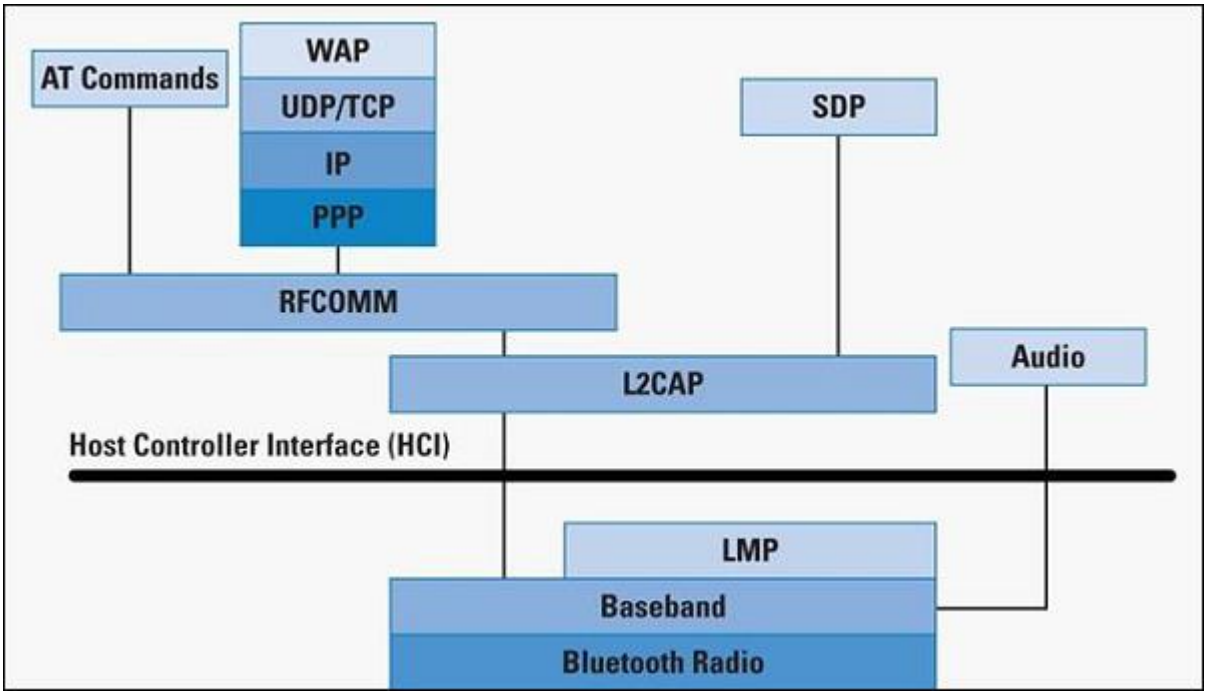

***Figure 216. Bluetooth Protocol Stack* []**

In Figure 16, we can see a description of the Bluetooth protocol stack. In this stack we its visible that the TCP/UDP Transport Layer can be accessed, but not from an IP socket as in WLAN protocol stack, but rather an RFCOMM socket.

Tests & Results

Since we have shown already how our application is working and how it handles all the required tasks for the achievement of our main goals, it is now wise to present the evaluations of the actual communication between our device and the remote embedded system.

In this Chapter we discuss the procedures we followed to evaluate each communication technology and the parameters that our evaluation tests were based on. Moreover, we present statistical results of the individual measurement that took place and we compare the two communication technologies according to their performance.

5.1. Communication Comparison

So far we have seen how we have constructed the application itself and all the consideration we had to make in order to achieve a secure communication and control of out remote embedded system. However, we our goal is to evaluate the methods and technologies we used and from the outcome of the results to decide the optimum solution for connecting to a remote system over a wireless communication technology. Therefore we have run some test considering the latency of each of the selected communication technologies as well as the achieved throughput.

5.1.1. Throughput

For the evaluation of the throughput of each of the communication technologies we used the iPerf3 bandwidth measurement tool. We ran three consecutive measurements for both Wi-Fi and Bluetooth technologies. The measurement time interval for each measurement was set to 60 seconds. Tables 7 and 8 show the results of the measurements for Wi-Fi and Bluetooth communication respectively.

Number of Measurement	Time Interval [seconds]	Data Transmitted [MBytes]	Throughput [Mbits/sec]
1	60.4	31.2	4.33
2	65.5	30.2	3.87
3	60.6	29.1	4.02

Table 7. Wi-Fi Throughput Measurement Results

Number of Measurement	Time Interval [seconds]	Data Transmitted [MBytes]	Throughput [kbits/sec]
1	80.2	3.88	406
2	60.9	4.24	585
3	72.2	4.32	501

Table 8. Bluetooth Throughput Measurement Results

The results of the two individual measurement show that the achieved throughput values of the Bluetooth connection are significantly lower than the ones for the Wi-Fi connection. On average, Bluetooth achieved throughput value is 497.33 kbits/sec where for the Wi-Fi the average value is 4.07 Mbits/sec. These results show that throughput-wise the Wi-Fi connection is the optimum solution as a choice for communication between devices.

5.1.2. Latency

For the latency tests we used the ping networking utility, which simply sends some ICMP echo request packets to the targeted host and waits for ICMP echo replies. We ran the ping utility 100 consecutive times for each of the communication technologies, sending a packet of 64 bytes each time. Tables 9 and 10 show the results of these measurements, both for Wi-Fi and Bluetooth respectively.

Packets Transmitted	Packets Received	Packet Loss [%]	Minimum Latency [ms]	Average Latency [ms]	Maximum Latency [ms]
100	99	1	48.256	434.349	1563.954

Table 9. Wi-Fi Latency Measurements Results

Packets Transmitted	Packets Received	Packet Loss [%]	Minimum Latency [ms]	Average Latency [ms]	Maximum Latency [ms]
100	88	12	2.452	99.728	1769.504

Table 10. Bluetooth Latency Measurements Results

As we can see from the tables above, latency-wise Bluetooth seems to perform significantly better than the Wi-Fi technology. The average latency for Bluetooth is almost four times lower than it is for Wi-Fi. However, the packet loss of Wi-Fi is twelve times lower than that of the Bluetooth. This mean that while the Bluetooth can may transmit data with lower latency, the total amount of data received can be significantly less than the amount of data transmitted.

5.2. Security Comparison

Considering the security of the two communication technologies under tests in this thesis, there is no optimal solution to choose from between the two. Both of the two communication technologies support security protocols and security mechanisms; however both of them are vulnerable to malicious attacks like man-in-the-middle, DOS attacks and other ways of surpassing the security provided by each protocol.

Nevertheless, Bluetooth might be a safer solution due to the fact that it is a short range wireless communication, which means that the potential malicious action has to take place in an area close to the targeted device, thus excluding attacks from a global wider area and reducing the number of threats a device connecting over Bluetooth can face.

Another argument to support the above stated suggestion is that due to Bluetooth's nature and low data rates, Bluetooth is not yet used for transferring data or setting up connections where important and big amounts of data need to be transmitted, thus making it less intimidating for hackers to develop methods and software to maliciously affect a Bluetooth connection. However, as technology progresses and Bluetooth data rates increase, the implementation of this technology as a medium of wireless communication will find use in a wider area of application and that may attract hackers to invest more on creating malicious software and techniques to surpass security mechanisms of the Bluetooth protocol.

On the other hand, with Wi-Fi being the most common wireless communication technology it is more than expected that there will be higher interest from malicious developers, to create software and methods to surpass the security protocols provided with the technology. However, it is worth mentioning that there is also bigger contribution from the telecommunication community, to constantly develop new security mechanisms that add extra protection to wireless network and extend their security. IEEE, which is responsible to manage the Wi-Fi standards constantly included new authentication schemes and encryption mechanisms with each new release of the Wi-Fi technology's standards, in an effort to continuously provide secure wireless networks.

Conclusion

This thesis is an attempt to evaluate the three wireless communication technologies supported by almost any modern smartphone device nowadays, and compare their performance while using them to securely connect and control a remote embedded system. We were able to create an application that's can use Wi-Fi and Bluetooth connectivity to establish a Secure Shell connection with our remote server and control it by running several commands. In addition to that, we were able to test the quality of each communication technology by measuring their throughput values and latency. We have proved that Wi-Fi perform better in data rates, but Bluetooth shows significantly less latency during data transmission. The outcome of the measurement fulfilled our initial expectations for the performance of each technology.

However, the implementation of the NFC connectivity used to run an SSH connection was not achieved. Due to the fact that NFC is not a TCP/UDP wireless communication technology, there is no standard method to implement an SSH connection over NFC. This scenario would require unorthodox means to achieve an SSH connection, which we were not available to implement during the time interval that this thesis was written in.

In the future, we would like to extend the functionality of this application by adding extra features considering accessing remote devices and work on implementing an approach to allow an SSH connection over NFC. This would allow the application to be used in home automation application as well as other types of application where secure remote access is required.

Bibliography & References

[1] MOVR. *"Mobile Overview"*. Report, January-March,2017

[2] Statista. *"Number of application available in leading app stores"*. Statista: The Statistics Portal, 2017, March, 2017.

https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/

[3] Android Team, *"Android 1.1 Version Notes"*. Google's Official Android Developer Webpage, December, 2011.

https://developer.android.com/about/versions/android-1.1.html

[4] Android Team, *"Android 1.5 Platform"*. Google's Official Android Developer Webpage, December, 2011.

https://developer.android.com/about/versions/android-1.5.html

[5] Android Team, *"Android 1.6 Platform"*. Google's Official Android Developer Webpage, December, 2011.

https://developer.android.com/about/versions/android-1.6.html

[6] Android Team, *"Android 2.0, Release 1"*. Google's Official Android Developer Webpage, December, 2011.

https://developer.android.com/about/versions/android-2.0.html

[7] Android Team, *"Android 2.0.1, Release 1"*. Google's Official Android Developer Webpage, December, 2011.

https://developer.android.com/about/versions/android-2.0.1.html

[8] Android Team, *"Android 2.1 Platform"*. Google's Official Android Developer Webpage, December, 2011.

https://developer.android.com/about/versions/android-2.1.html

[9] Android Team, *"Android 2.2 Platform"*. Google's Official Android Developer Webpage, December, 2011.

https://developer.android.com/about/versions/android-2.2.html

[10] Android Team, *"Android 2.3 APIs"*. Google's Official Android Developer Webpage, December, 2011.

https://developer.android.com/about/versions/android-2.3.html

[11] Android Team, *"Android 2.3.3 APIs"*. Google's Official Android Developer Webpage, December, 2011.

https://developer.android.com/about/versions/android-2.3.3.html

[12] Android Team, *"Android 3.0 APIs"*. Google's Official Android Developer Webpage, December, 2011.

https://developer.android.com/about/versions/android-3.0.html

[13] Android Team, *"Android 3.1 APIs"*. Google's Official Android Developer Webpage, December, 2011.

https://developer.android.com/about/versions/android-3.1.html

[14] Android Team, *"Android 3.2 APIs"*. Google's Official Android Developer Webpage, December, 2011.

https://developer.android.com/about/versions/android-3.2.html

[15] Android Team, *"Android 4.0 APIs"*. Google's Official Android Developer Webpage, December, 2011.

https://developer.android.com/about/versions/android-4.0.html

[16] Android Team, *"Android 4.0.3 APIs"*. Google's Official Android Developer Webpage, December, 2011.

https://developer.android.com/about/versions/android-4.0.3.html

[17] Android Team, *"Android 4.1 APIs"*. Google's Official Android Developer Webpage, December, 2011.

https://developer.android.com/about/versions/android-4.1.html

[18] Android Team, *"Android 4.2 APIs"*. Google's Official Android Developer Webpage, December, 2011.

https://developer.android.com/about/versions/android-4.2.html

[19] Android Team, *"Android 4.3 APIs"*. Google's Official Android Developer Webpage, December, 2011.

https://developer.android.com/about/versions/android-4.3.html

[20] Android Team, *"Android 4.4 APIs"*. Google's Official Android Developer Webpage, December, 2011.

https://developer.android.com/about/versions/android-4.4.html

[21] Android Team, *"Android 5.0 APIs"*. Google's Official Android Developer Webpage, December, 2011.

https://developer.android.com/about/versions/android-5.0.html

[22] Android Team, *"Android 5.1 APIs"*. Google's Official Android Developer Webpage, December, 2011.

https://developer.android.com/about/versions/android-5.1.html

[23] Android Team, *"Android 6.0 APIs"*. Google's Official Android Developer Webpage, December, 2011.

https://developer.android.com/about/versions/marshmallow/android-6.0.html

[24] Android Team, *"Android 7.0 for Developers"*. Google's Official Android Developer Webpage, December, 2011.

https://developer.android.com/about/versions/nougat/android-7.0.html

[25] Android Team, *"Android 7.1 for Developers"*. Google's Official Android Developer Webpage, December, 2011.

https://developer.android.com/about/versions/nougat/android-7.1.html

[26] Android Team, *"Android Oreo – Introducing Android 8.0 Oreo"*. Google's Official Android Developer Webpage, December, 2011.

https://developer.android.com/about/versions/oreo/index.html

[27] Michalis Katsarakis. *"An Introduction to Android"*, Electronic Presentation, October,2012

http://www.csd.uoc.gr/~hy439/labs/hy539AndroidIntro2012.pdf

[28] Prabhaker Mateli. *"Android OS"*. Electronic Presentation, June,2013

http://gauss.ececs.uc.edu/Courses/C653/lectures/PDF/android.pdf

[29] Christoph Krauss, Volker Fusenig, Rafael Fedler and Christian Banse. *"Android OS Security: Risks and Limitations"*. Fraunhofer Research Institution for Applied and Integrated Security, May, 2012.

[30] Android Security Team. *"Android Kernel Security"*. Google's Official Android Developer Webpage, December, 2011.

https://source.android.com/security/overview/kernel-security

[31] Android Security Team. *"Android Application Security"*. Google's Official Android Developer Webpage, December, 2011.

https://source.android.com/security/overview/app-security

[32] Ian Poole. *"IEEE 802.11 Wi-Fi Standards"*. Radio-Electronics.com, 2014

http://www.radio-electronics.com/info/wireless/wi-fi/ieee-802-11-standards-tutorial.php

[33] Bradley Mitchell. *"Wireless Standards 802.11a, 802.11b/g/n, and 802.11ac"*. Limewire.com, January,2018

https://www.lifewire.com/wireless-standards-802-11a-802-11b-g-n-and-802-11ac-816553

[34] T.Shidhar. *"Wi-Fi, Bluetooth and WiMAX"*. The Internet Protocol Journal, Volume 11, No. 4, December,2008

https://www.cisco.com/c/en/us/about/press/internet-protocol-journal/back-issues/table-contents-42/114-wifi.html

[35] IEEE Computer Society. *"IEEE Standard for Local and metropolitan area networks"*. Part 15.4,September,2011

http://ecee.colorado.edu/~liue/teaching/comm_standards/2015S_zigbee/802.15.4-2011.pdf

[36] Majid Abarghooei. *"Milti-Purpose Permission-Base Bluetooth Advertising System Based on SDP, RFCCOMM, and OBEX"*. ResearchGate Wepage,January,2015

https://www.researchgate.net/publication/283246318_Multi-Purpose_Permission-Based_Bluetooth_Advertising_System_Based_on_SDP_RFCOMM_and_OBEX

[37] NFC Forum. *"NFC - Protocol Technical Specifications"*. Nfcforum.com, 2015

https://nfc-forum.org/our-work/specifications-and-application-documents/

[38] Nokia. *"Introduction to NFC".* Version 1.0, April,2011

[39] William Stallings. *"Protocol Basics: Secure Shell Protocol"*. The Internet Protocol Journal, Volume 12, No.4, December 4, 2009.

https://www.cisco.com/c/en/us/about/press/internet-protocol-journal/back-issues/table-contents-46/124-ssh.html

[40] SSH Communications Security Team. *"SSH Protocol"*. SSH Communications Security, August 29, 2017.

https://www.ssh.com/ssh/protocol/

[41] . Lehtinen. *"The Secure Shell (SSH) Protocol Assigned Numbers"*. RFC 4250, January, 2006.

ftp://ftp.rfc-editor.org/in-notes/rfc4250.txt

[42] T. Ylonen. *"The Secure Shell (SSH) Protocol Architecture"*. RFC 4251, January, 2006.
ftp://ftp.rfc-editor.org/in-notes/rfc4251.txt

[43] T. Ylonen. *"The Secure Shell (SSH) Authentication Protocol"*. RFC 4252, January, 2006.
ftp://ftp.rfc-editor.org/in-notes/rfc4252.txt

[44] T. Ylonen. *"The Secure Shell (SSH) Transport Layer Protocol"*. RFC 4253, January, 2006.
ftp://ftp.rfc-editor.org/in-notes/rfc4253.txt

[45] T. Ylonen. *"The Secure Shell (SSH) Connection Protocol"*. RFC 4254, January, 2006.
ftp://ftp.rfc-editor.org/in-notes/rfc4254.txt

[46] T.Ylonen. *"Using DNS to Securely Publish Secure Shell (SSH) Key Fingerprints"*. RFC 4255, January, 2006.

ftp://ftp.rfc-editor.org/in-notes/rfc4255.txt

[47] F. Cusack. *"Generic Message Exchange Authentication for the Secure Shell Protocol (SSH)"*. RFC 4256, January, 2006.

ftp://ftp.rfc-editor.org/in-notes/rfc4256.txt

[48] SSH Communications Security Team. *"SSH Key"*. SSH Communications Security, September 2, 2017.

https://www.ssh.com/ssh/key/

[49] Android Team. *"Meet Android Studio"*. Google's Official Android Developer Webpage, December, 2011.

https://developer.android.com/studio/intro/index.html

[50] Android Team. *"App Manifest"*. Google's Official Android Developer Webpage, December, 2011.

https://developer.android.com/guide/topics/manifest/manifest-intro.html

[51] HTC. *"HTC Desire C"*. HTC Czech Republic Webpage

http://www.htc.com/cz/smartphones/htc-desire-c/

[52] Raspberry Pi. *"Raspberry Pi 3 Model B"*. Raspberry Pi Official Website

https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

[53] Android Team. *"Layouts"*. Google's Official Android Developer Webpage, December, 2011.

https://developer.android.com/guide/topics/ui/declaring-layout.html

[54] Android Team. *"Introduction to Activities"*. Google's Official Android Developer Webpage, December, 2011.

https://developer.android.com/guide/components/activities/intro-activities.html

[55] Android Team. *"Activity"*. Google's Official Android Developer Webpage, December, 2011.

https://developer.android.com/reference/android/app/Activity.html

[56] Martin Sauter. *"From GSM to LTE: An Introduction to Mobile Networks and Mobile Broadband"*. John Wiley & Sons, March, 2011

[57] TECHLEDGER. *"Networking 101"*. Techledger Webpage,March,2011.

https://techledger.wordpress.com/2011/03/01/networking-101/

[58] Ankit Sinhal. *"Closer Look At Android Runtime: DVM vs ART"*. AndroidPub Blogspot, April,2017.

https://android.jlelse.eu/closer-look-at-android-runtime-dvm-vs-art-1dc5240c3924