Daniel Holmström

Developing Android Application for Work Order and Work Hour Management

Helsinki Metropolia University of Applied Sciences

Master of Engineering

Information Technology

Master's Thesis

29 September 2017

Metropolia

Author	Daniel Holmström
Title	Developing Android Application for Work Order and Work Hour Management
Number of Pages	46 pages + 0 appendices
Date	29 September 2017
Degree	Master of Engineering
Degree Programme	Information Technology
Instructor(s)	Ville Jääskeläinen, Principal Lecturer

This Master's thesis deals with the development process of an Android application. The application is meant to be used as a tool for managing both a company's work orders and the individual work hours of the staff. First, the study gives a brief introduction to the company and a general overview of the thesis subject. Next, it goes deeper into the background of this thesis, explaining the need for the application and how it can help the company to achieve a more efficient work routine.

The development tools utilized in this project are also introduced. The application itself was developed using the Microsoft Visual Studio 2017 Professional Edition with the Xamarin plugin, the database utilized for storing work order and work hour information was a Microsoft SQL Server. In order for the application to be able to connect to the database, a Web service was created on a local web server. The study also goes through theory about the Android operating system, providing a brief history of the platform along with explanations regarding terms such as views, activities and intents.

Next the study deals with the actual implementation process of the application, starting with the planning stages where diagrams and timetables are laid forth. It is also explained why the development process slightly deviates from standardized development methods such as Scrum and Agile. The problems and challenges faced by the thesis author both in terms of technical issues and pressed timetables are also presented.

After the product was released, feedback was gathered from the end users and that together with potential scenarios for future development needs for the application are introduced.

Keywords	android, application, work order, work hour, programming, csharp, sql

Metropolia

Contents

Abbreviations

API	Application Programming Interface. A set of tools and utilities for developing different software applications.
ART	Android RunTime. An application runtime environment in the Android system, replaces the Dalvik runtime since Android version 5.x.
DB	Database. Refers to the Microsoft SQL relational database in this thesis.
EE	Ekenäs Energi Ab. A local electricity distribution company in the town of Raseborg (Raasepori) Finland.
MVC	Model-View-Controller. An architectural pattern used in user interface development.
ODBC	Open Database Connectivity. A standardized method for accessing and retrieving data from any database.
Open GL	Open Graphics Library. A cross language and cross platform tool for rendering 2- and 3D graphics.
SQL	Structured Query Language. In this Master's thesis it refers to the Microsoft T-SQL standard of the language.
SP	Stored Procedure. A static database query that has been created in order to speed up query execution times and reduce the risk of SQL –injection.
SSL	Secure Sockets Layer. A web based protocol that enables encrypted communication over a network.
UI	User Interface. Any information that is being displayed to the user on the screen.
VPN	Virtual Private Network. A technology that makes it possible to connect to a private or closed network from an outside source.

1 Introduction

The aim of this master's thesis was to examine and develop a solution for easy management of work orders and work hours that can be used anywhere, anytime. This chapter gives a brief introduction of the company involved and some background information on the thesis subject. The company for which the thesis was made is called Ekenäs Energi Ab in Swedish, or Tammisaaren Energia Oy in Finnish, and is abbreviated EE for future reference. EE is a company in the electrical/energy industry that owns the core electric grid and the district heating grid in the municipality of Ekenäs (Raseborg) in the southwestern part of Finland. The electrical grid consists of a network of electrical wires spanning 419 km in total with almost 7000 metering points along the way. EE is responsible for the maintenance of this grid and for continually supplying their grid customers with electricity. The district heating grid stretches throughout the city of Raseborg (Raseborg consists of the old municipalities Ekenäs, Karis and Pojo) with around 600 customers in Ekenäs, 170 customers in Karis and 60 customers in Pojo. In addition, EE also offers electrical installation jobs, mainly for schools, health care centers and other public institutions within the city. For private consumers, EE mainly offers installation services for electrical car charging stations and solar panels.

All in all, EE performs around 700 individual work orders including more than 50 000 work hours every year divided by a little over 20 workers in total. To make the management and follow up possible, there are two separate software programs in use, one for managing work hours and the other for managing work orders. Both of these programs were designed by the current IT staff of EE and are meant to work on devices that use the Microsoft Windows operating system and are written in the C# programming language. The work order management program was developed back in 2011 as a Bachelor's thesis project by the author of this Master's thesis. However, even though these programs are still in use on a daily basis and have helped the company management a lot, the drawbacks have become quite evident, they offer close to no mobility at all for the workers.

One way to deal with the mobility issue and thus improve the work efficiency, is to develop a new mobile application solution that serves as a compliment to the existing software. This also offers an opportunity to merge the work order and work hour soft-

ware into one application and it makes more sense to have only one application since they already are dependent on each other.

This thesis includes a study to define and implement the new mobile application. The new application will not be identical to the existing software solutions, rather, it will be a light-weight version were only the more basic functions are included. This is mostly due to the fact that it is very difficult to fit a large amount of information on a small mobile phone screen.

This thesis has been divided into six chapters. Chapter one introduces the topic of thesis on a general level while chapter two gives more background information about the topic. In chapters three and four, theory on the Android operating system is dealt with along with an introduction of the development tools utilized in this thesis as well as a preliminary timetable. Chapter five deals with the actual development process where the solutions for the UI and coding are presented. Several practical challenges regarding the project schedule and programming difficulties are also included in this chapter. The sixth chapter gives some final thoughts about the project as a whole and concludes this thesis.

2 Current Situation

Since the idea behind this project was to make the workflow more efficient through offering more mobility to the workers, a general understanding of what the functional requirements are needed to be studied. One of the best ways to accomplish that was to ask a selection of central users, or workers as they are referred to in this thesis, what their thoughts were. This was done by conducting personal interviews with three employees. During the interviews the following questions were asked:

- When considering the company's software for handling work orders and work hours, can you think of any practical issues with these programs, functional or other issues?
- How important is it, from your perspective, to address these issues?
- What is the current situation in regards to these programs?
- In your opinion, what could be done to develop a better solution?
- Can you specify which parts of these programs would need redesigning or added functionality?
- What are the company specific requirements for the outcome / solution?
- Which data / information will the company allow me to analyze or have access to?
- Which resources (work time / budget / help) is the company willing to invest into this project?

Note that these questions were mostly meant to establish a general understanding of the functional requirements to improve efficiency. Even before the interviews were conducted, the idea of developing a mobile application solution had been suggested.

After conducting the interviews and analyzing the results, a few key areas were discovered. One aspect that frequently came up was the lack of mobility, the current software solution can only be used on a device running the Windows operating system which in this case means desktop or laptop computers. This coupled with the fact that the software can only be used either while being physically inside the company's own network or through a VPN connection means that there is no practical way of utilizing it when working in the field. This could mean a field worker, especially the foremen, must make several extra trips back and forth between the office and the worksite. Other issues that were brought up dealt with improving or adding functionality to the software. These included having an option for taking pictures of work sites and adding them as an attachment to work orders, and adding a reporting function for field workers that are on call.

One way to solve the issues would be to remake the software in such a way that it adds mobility, in other words so that it can be used wherever and whenever. At the same time, adding requested additional functionality such as taking pictures and storing them in a specific work order and having an area for on call workers reports. Developing an Android application that is configured to communicate with the existing Microsoft SQL -database would immediately solve the issue of mobility and allow for an easy integration with the mobile phone camera for taking pictures of work sites. This solution was also the preferred one among the interviewees.

3 Development Tools and Timetable

To start off, one first needs to consider what kind of approaches and tools are available for Android application development. The main programming language for Android is Java coupled with xml for the user interface design. However, there are many software solutions available that allow the developer to do the programming in other languages which is then converted to Java code by the compiler.

The most commonly used platform among Android developers is Android Studio, a powerful development tool that allows for a “drag ’n drop” style design of the user interface and Java programming for the functionality of the application. Although this could have been chosen as the main development tool, considering the author’s previous knowledge of C#, choosing a development environment that utilizes C# programming seemed more appropriate. Therefore, the development tool of choice was Xamarin for Visual Studio Professional Edition. Xamarin is a plugin for Microsoft Visual Studio that enables for programming in C# instead of Java, the code is then translated to Java by the compiler when the application is built.

3.1 Xamarin for Visual Studio Professional Edition

The Visual Studio Professional Edition is a cross-platform development tool from Microsoft meant for small teams, a maximum of five developers according to the license agreement, and allows for the use of various plugins and programming in many languages, among others Basic, C++ and C#. Xamarin for Visual studio is a plugin tool that enables for mobile application development for iOS, Windows Phone and Android. Figure 1 demonstrates the project solutions that are available in the Visual Studio 2017 Professional Edition with the Xamarin plugin installed.

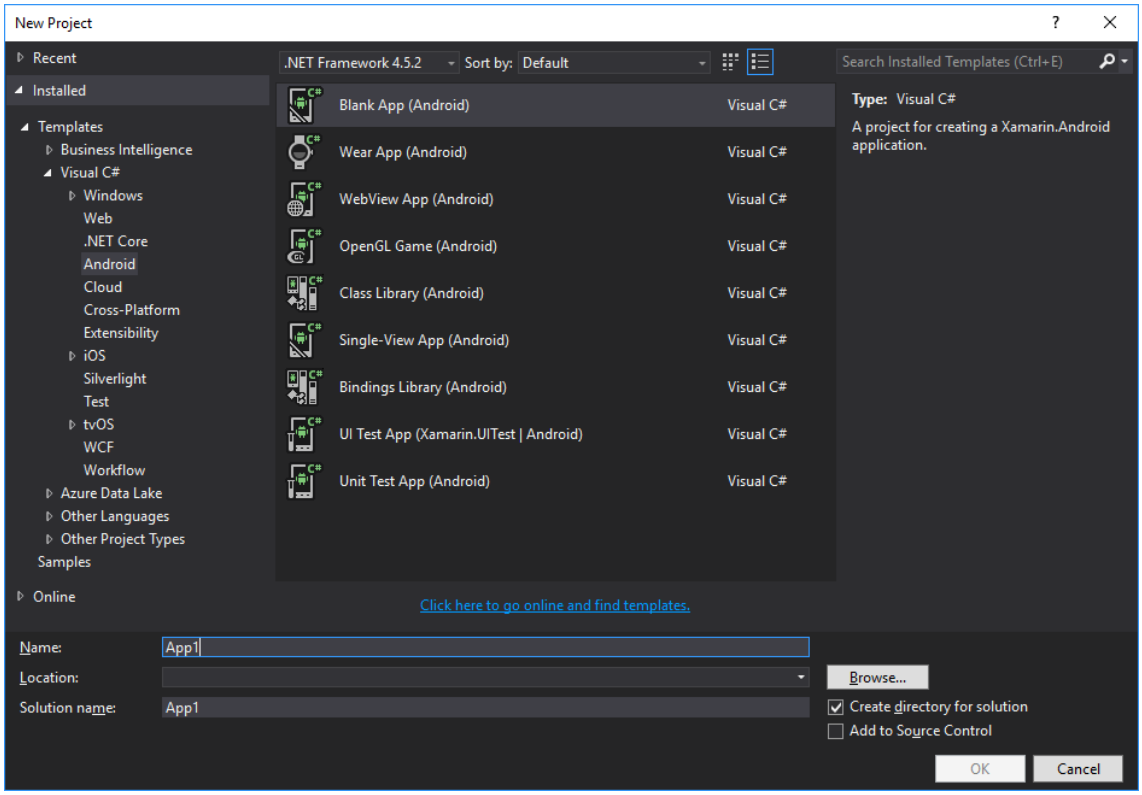

Figure 1. The “New project” window of the Visual Studio 2017 Professional Edition with the Xamarin plugin.

As can be seen in Figure 1, Visual Studio offers templates for a large variety of platforms such as Android, iOS, Windows and cross-platform solutions. In this study though, the focus was on the Android templates.

3.2 Microsoft SQL Server

The Microsoft SQL server is a relational database solution that offers the ability to run database queries using the transact SQL standard (T-SQL). The Android application will utilize an already existing Microsoft SQL database for storing information on work orders and work hours. This means all application queries and connection strings must be written to suit the existing table and field structures.

In order for the application to be able to access this DB, a Web service located on a separate server will be used, this enables the application to connect to the database from an outside source. In addition to establishing a link between the application and the DB, the Web service will also be responsible for sending query requests to the DB.

3.3 Timetable and Methods

The preliminary goal at the start of the study was to deliver the new product to the company by June 2017. The actual development process started back in December 2016 so the whole length of the project was planned to take just over six months. However, while the project was still in the planning stages at the end of February 2017 it became evident that the original timetable had been too optimistic, the new delivery date of the product was therefore moved to September 2017. The updated timetable can be seen in Figure 2.

Figure 2. The updated version of the timetable.

Due to the fact that there is only a single developer involved in the project, it does not directly follow any of the established project development models. Rather, the project consists of five parts referred to as phases. In the Scrum development model, the equivalent to these phases would be the sprints. However, the length of such a sprint is usually no more than two or three weeks, in this project the length of one development cycle, or phase as they are referred to, is roughly two or three times as long, around five to nine weeks. And also, since a sprint generally refers to something of a short duration, it seemed more fitting to use the term phase here.

The first phase mostly involved the planning and groundwork for the application. Interviews were conducted with the key staff that mainly use the application to find out what

the needs were and in accordance to those needs, design visual process flow charts and make a detailed project plan of key features. Research about development tools and environments was also included in this phase.

The second to fourth phases were about gradually implementing features, adding more functionality at each new phase and conducting testing at the end of each phase with a selection of workers. In the reporting of these phases the reader can also get an overview of different approaches to problem solving and some security considerations in programming.

The fifth phase was the final phase of the project and it included implementing any additional features that might have been requested during the previous phases and refactoring and reviewing the code. There was no testing at the end of this phase, however, as the finished application will be in production use, any potential further improvements or bug fixes will be dealt with as they are reported.

4 Developing Android Application

When it comes to Android application development, a vast amount of information is available for free from various web sites and discussion forums. The Xamarin team has their own web page with detailed tutorials about simple android application solutions. This chapter goes into more detail about the Android operating systems history and basic components and the framework.

4.1 Android Platform

Since the Android operating system first became available on mobile devices back in 2008, Android has become the dominant operating system for mobile devices world-wide with a market share that exceeds all the other mobile operating systems combined. The source code for the Android operating system goes under the so called “open source” license, meaning that not only is it free to use but anyone can also make changes to the code as they see fit. (Burton, 2015: 9-10)

The Android operating system is built on the Java programming language, however, due the extensiveness of the Java libraries only a smaller subset of Java classes have been included to eliminate any potential incompatibilities for mobile devices. (Burton, 2015: 12-13)

Since the Android operating system was first released as a beta version back in November 2007, the OS has gone through multiple changes and version updates. The Android platform includes both a version number and a platform number, for instance version number 5.1 where the number five represent the major version number and the number 1 is the minor version release number. The platform number or API number as it is referred to, is always represented by an integer, in other words a non–decimal number such as API 22. (Panigraphy, 2015: 2)

Android activities

In all Android applications there are at least one or several activities. An activity is basically a container that encapsulates an assigned user interface coupled with the code that runs in the background for that user interface. In general, one activity equals one page in an application (Burton, 2015: 13). In order for a user to interact with an activity,

a view is required, this could be compared to the MVC (Model-View-Controller)–model, the activity in this case represents the controller in that model (Panigraphy, 2015: 12).

Android fragments

For mobile devices such as phones where the screen size is limited, displaying one activity at any given time is usually appropriate, having more than one activity at a time on such a screen would make the outlook very cramped. However, on a device with a bigger screen such as a tablet PC there is more than enough room for several activities at a time. Since it is not possible to place several activities on one page the solution offered by Android is called fragments. A fragment is simply put a lightweight activity that only occupies a small part of your screen, allowing for multiple fragments to be visible at the same time on the same page. (Burton, 2015: 13-14)

Android intents

In Android, an intent is simply a way of sending commands or information within an activity or passing them on to another activity. There are two components that make up an intent:

- An action, which can be a view, edit or dial function.

- Data, which is the information contained within an action.

An intent can be, for example, a method within one activity that calls another activity to be started. (Burton, 2015: 14)

Android views

Activities, fragments and intents make up the core functionality of Android, however, to be able to see anything on the screen we need one more thing, a view. Views are assigned to a specific rectangular area of the screen and are usually one of these: TextView, ImageView, Layout or Button. (Burton, 2015: 15)

4.2 Android and Application Framework

The Android framework was developed to run on top of the Linux kernel, an open source operating system kernel. With the aid of a variety of different open source projects, the following features were included in the Android framework: (Burton, 2015: 27)

- **ART:** This is basically a collection of Java core libraries and the ART. In the Android versions predating 5.x, the Dalvik runtime was utilized instead of Android runtime.

- **Open GL:** The Open GL is both a cross-platform and cross-language API that is utilized in order to produce computerized 2D or 3D graphic.

- **WebKit:** This is a simple web browser engine that can be used to display web content and simplify loading of pages.

- **SQLite:** An open source version of the relational database engine mainly intended for use in embedded devices.

- **Media frameworks:** These frameworks allow for the playing and recording of both audio and video.

- **SSL:** A collection of libraries that are used for secure connections.

In order to for a developer to be able to utilize the above mentioned libraries, the Android platform has a built-in application framework that offer, among others, the following tools: (Burton, 2015:28)

- **Activity manager:** A tool for managing an activity's lifecycle.

- **Telephony manager:** As the name suggests, this tool provides access to telephony services including subscriber information such as phone numbers.

- **View system:** A tool for handling and adding components to the UI, such as views and layouts.

- **Location manager:** Utilizes GPS in order to find the actual location of the phone.

Figure 3 displays all of these components put together to form the complete framework.

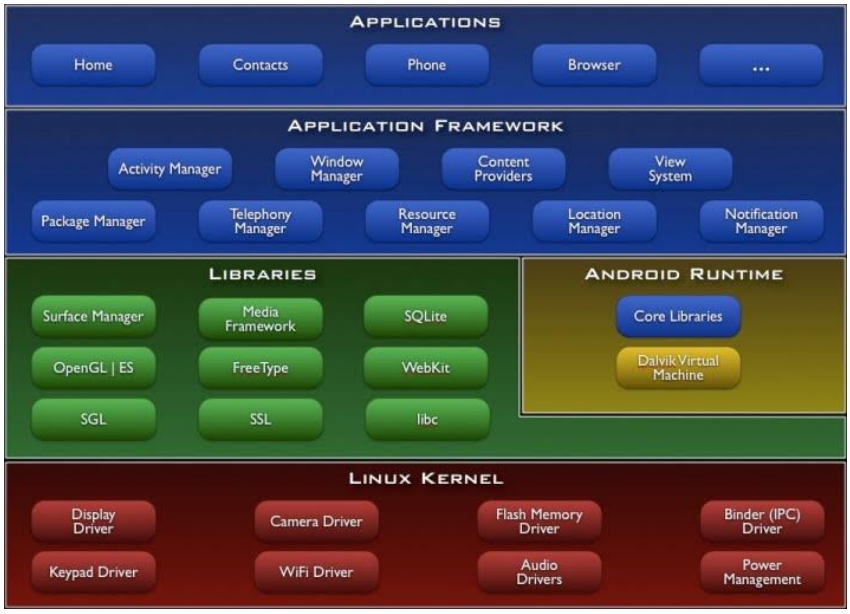

Figure 3. The different layers that make up the Android framework stack. (EE Daily news, 2011)

To put all of this into perspective, Figure 3 illustrates the different framework components as they are stacked on one another. As a developer, the topmost part of the stack represent the visual tools that are at disposal: (Burton, 2015:29)

5 Planning, Implementation and Testing

This chapter describes in detail the actual development cycle of the software, from the planning stages up until completion of the new application. Images of the actual code, different programming ideas and difficulties along the way are also included here.

5.1 Phase One: Planning

In Chapter 2, it was explained what kind of a solution was needed and the project then moved on to take a closer look at the functionality and visual appearance of the desired end product.

The visual layout of the finalized application takes into account the different processes involved when filling in and editing work orders and work hours. The design chart in Figure 4 illustrates the basic application layout and process flows. In the chart, names such as EditText and Spinner directly represent the controls that are used in Visual Studio, a control is a basically a component for user inputs in Android. For clarity, the adding of work orders processes have a light blue colored background while the adding of work hours processes have a light green background color.

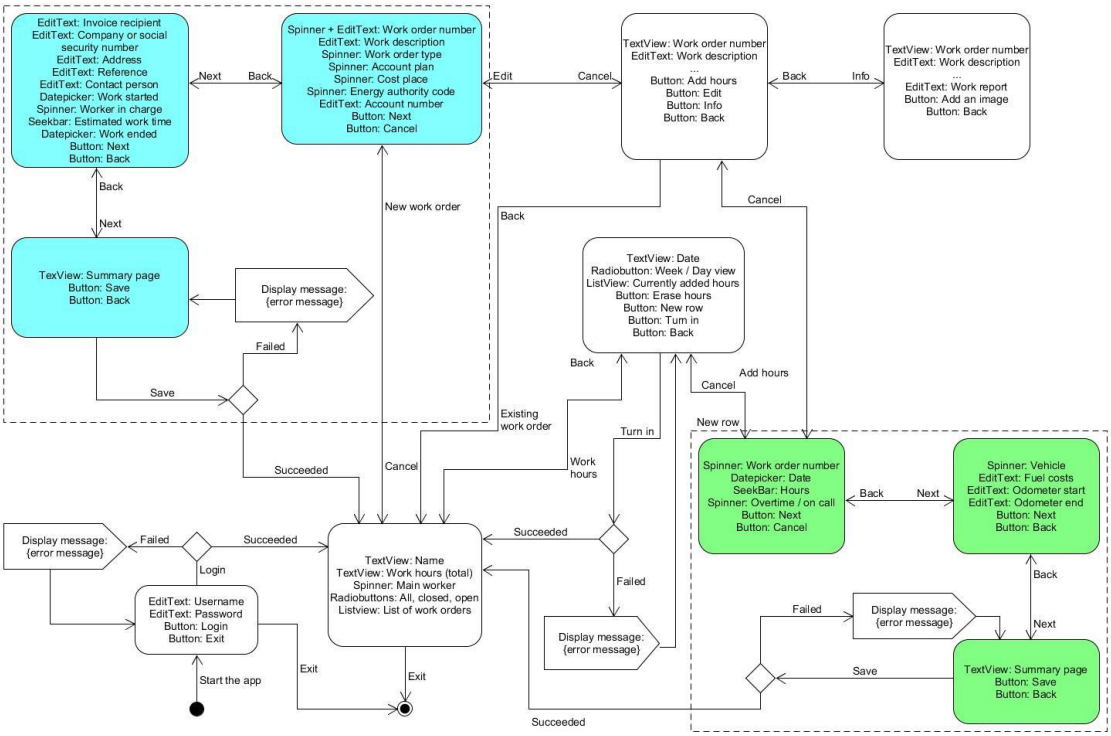

Figure 4. Application design as a process flow chart.

There are two distinct user categories in this application, regular workers and work foremen acting as administrators of the application. The work foremen have access to all available work order and work hour data whereas regular workers only have access to their own work hours and the work orders they are currently assigned to. When the application is opened, a simple login page is displayed where a username and password can be entered. Every user is authenticated to the SQL database and if the entered credentials are correct, the main window of the application is opened. Here, the full name of the logged in worker is displayed along with some basic information about the workers current weekly working hours. Below this information, a list of work orders are displayed in accordance with the selected filters.

Clicking on a the weekly hours opens up a more detailed view of the work hours including options for editing or erasing existing hours or filling in new ones. There is also an option for submitting a report of the hours to the assigned foreman for inspection. Editing or adding hours opens up a three step process which ends with a summary window and a *save* button that saves the information to the DB. Clicking on a work order from the list of work orders opens up a summary window of that work order with the option of editing and an *info* button for viewing more detailed information about the work order.

The more detailed view also includes the ability to fill in a work report and to take pictures and add them to the work order. In the more detailed view there is also an option for filling in work hours for that particular work order.

A button for adding work orders is available from the main window, clicking on this button will open up a three step process with a similar logic to that of the work hours. Editing existing work orders from the *edit* button will also start this three step process.

5.2 Phase Two: Implementation First Cycle

When the project plan had been created along with a time table and the theoretical functionality drawings for the application were ready, the development work started. Next, the systems for this project were set up, including installing the required software and configuring it.

5.2.1 Setting up Development Environment

The development system was a workstation with the Windows 10 64-bit operating system. In addition, the following tools were needed for the development setup: Microsoft Visual Studio Professional Edition 2017, Xamarin for Visual Studio plugin and a Microsoft SQL database server. The SQL database server was already installed and in use so no additional actions were required for this part. The Visual Studio Professional Edition can be downloaded from this website: https://www.visualstudio.com/downloads/ note however that the Professional and Enterprise Editions of Visual Studio are not free of charge. The Visual Studio Community Edition is the only choice for those developers who are looking for a free version of Visual Studio, however, it is not allowed to develop commercial products for large companies using that version. Figure 5 displays the components chosen for this install, the “Mobile development with .NET” –component is necessary to install in order to be able to utilize Xamarin Android development tools.

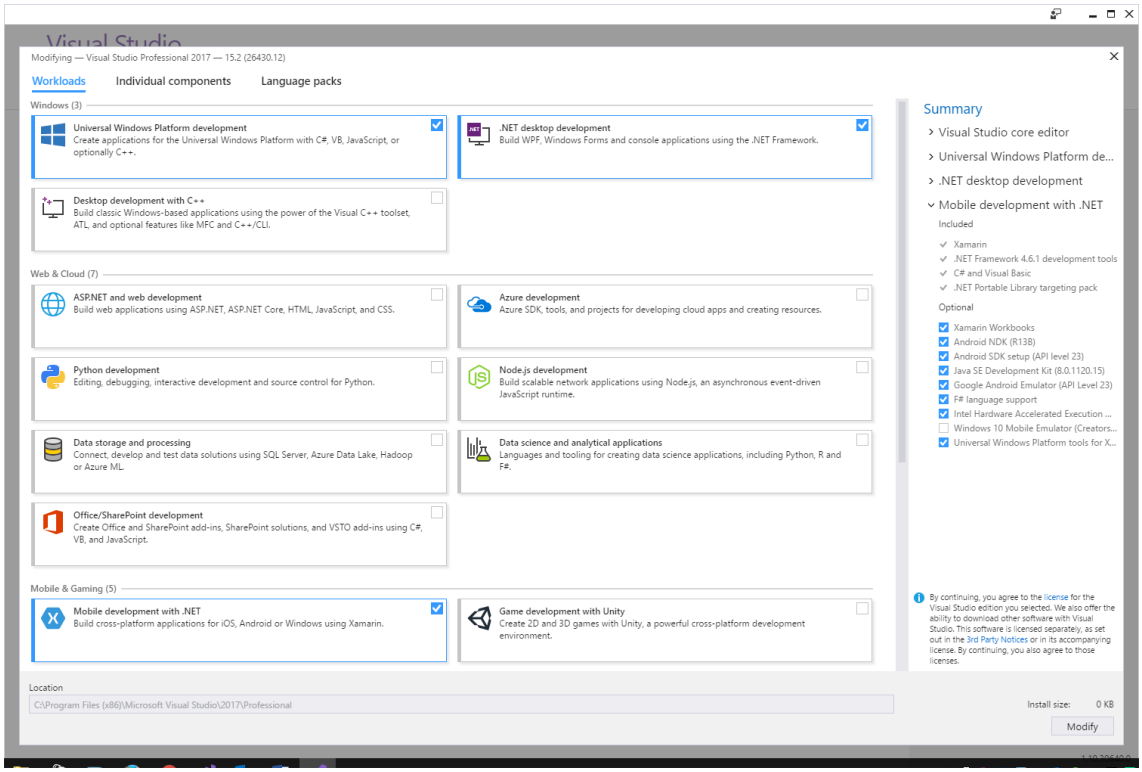

Figure 5. The install options of the Microsoft Visual Studio installer.

The installation process for Visual studio is rather straightforward and does not require any additional prerequisite software installations. Once all of these had been installed, the Xamarin plugin inside the Visual Studio Options menu needed to be configured, this menu is shown in Figure 6.

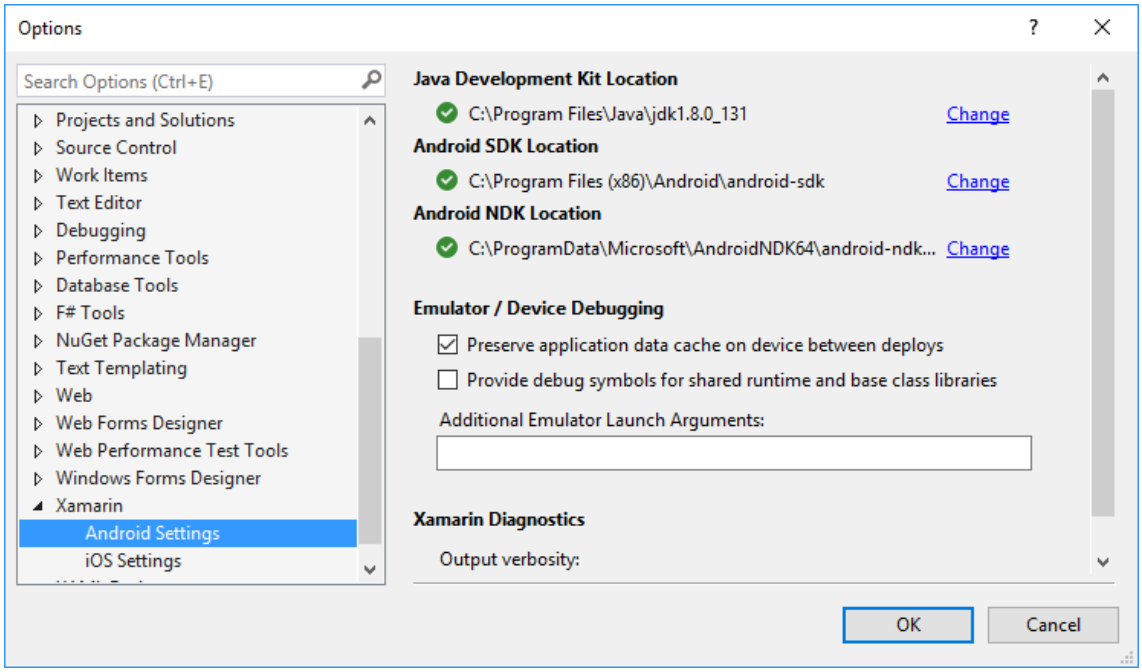

Figure 6. The Xamarin plugin options menu in Visual Studio

As seen in Figure 6, the paths to all three development kits needs to be checked and corrected if needed, indicated by either a green or red icon. If any of the three icons are red, it either means the path to the software is incorrect or that the software is not installed.

Next, the Android emulator was configured, this allows for a virtual Android device to be run on the workstation. The Add / Edit device option displayed in Figure 7 can be found in the Android Virtual Device -manager.

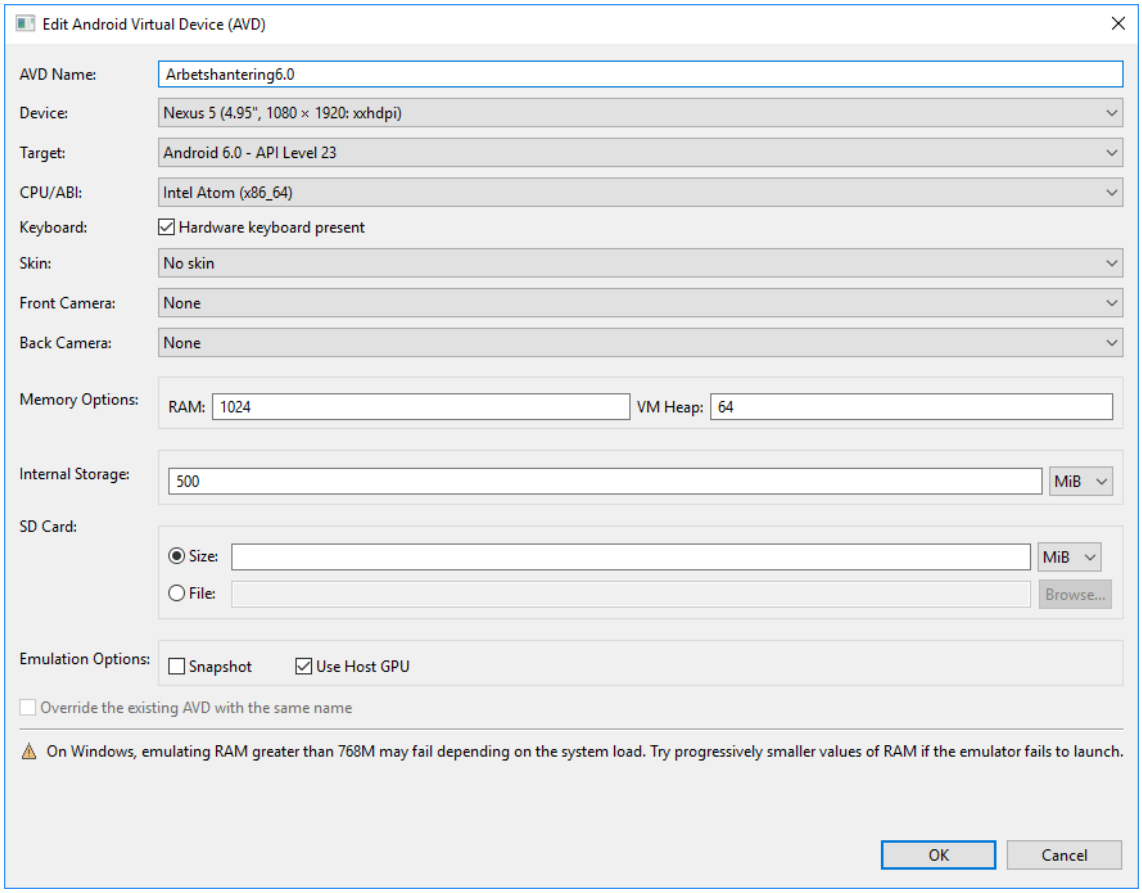

Figure 7. The Android Emulator Manager in Visual studio

Figure 7 displays the recommended setup of the virtual device as instructed from the Xamarin website. Take particular note of the settings “CPU/ABI: Intel Atom (x86_64)” and “Emulation Options: Use Host GPU”, these two options address the issue of the emulator operating extremely slowly. Next, the Intel Hardware Accelerated Execution Manager (HAXM) must be installed in order to see the “Intel Atom x86_64” -option, instructions can be found from the Xamarin website (Xamarin 2017). The Android version of choice here was 6.0 but a separate emulator running the newer Android version 7.0 was also utilized later in this thesis.

When configured correctly, the difference in speed is like night and day, going from an extremely slow, almost unusable emulator to a fast and more stable emulator. Though, depending on the emulated device and Android version, the option for “Intel Atom x86_64” might not always be available, but for example the “Android Wear Intel Atom x86” should also work just as well. Now when the environment was ready and the emulator was running at an acceptable speed, the creation of a new project and implementation of some basic elements could begin.

5.2.2 Implementing Primary Activities, Views and Database Login

Among the Android development community, discussions can be found regarding whether it is better to use a single activity for the whole app, use one activity with fragments or simply have many activities for different screen layouts (Stack overflow 2010). All of these options have their own advantages and drawbacks and in the end, which to choose comes down to personal preference. The path followed in this thesis was the last option; one activity for one layout with a couple of exceptions. The activities for creating and editing work orders and work hours have three layouts controlled by one activity. The reason why this way was chosen was mainly because it corresponds well with the Windows way of thinking in C#, where one form (layout in the Android world) has an underlying code base (activity in the Android world). Another reason is that having all code in a single activity can be messy and hard to maintain, dividing the code into multiple places gives a seemingly more organized overview of the system. The issues that could potentially arise from this are, for example, that running multiple activities at the same time eats up more system memory than necessary. One way to limit the use of memory is to “clean up”, in other words dispose of activities immediately after the worker has finished working with that particular layout and limit the amount of simultaneous activities running.

The project started by creating .axml- (layout) and .cs (code) –files that correspond to the intended layout in Figure 4. Since one .cs –file can be utilized to operate several .axml –files, there were seven .cs-files and 11 .axml-files in this project. If the intention had been to develop a cross-platform application (useable on both Android and iOS) the .xaml file format would have been used instead of the .axml format. However, since this was a pure Android project, the .axml format made more sense to use.

After the basic layout and code files had been created one could move on to implement some basic functionality such as log in with a user name and a password. The log in page can be seen in Figure 8.

Figure 8. The log in page of the application.

The project immediately ran into some difficulties though, since this was not a pure Windows application but rather an Android application and there is no native way of connecting to a Microsoft SQL database server from the application itself. One must instead rely on Web services for this operation to succeed. In short, a Web service is an interface, usually situated on the same server as the SQL database that acts as an intermediate layer for connecting to the DB from an outside source.

The Web service in this case consists of two .php –files, one that holds connection information to the DB itself and the main .php -file that contains all queries, execution and results return parameters. Initially, the main .php -file calls the connection script which in turn establishes the DB server connection using the ODBC interface. After the connection to the DB has been established, the correct query is chosen by a parameter sent from the application. After the query has been executed, the result from the DB is sent back to the application as a string.

The communication to the Web service is made possible by using a set of predefined methods uploadValuesCompleted and uploadValuesAsync along with the WebClient library. In order to be able to pass parameters to the Web service, the NameValueCol-

lection library is utilized. Figure 9 illustrates a part of this code. It is worth taking particular note of the [Export("name of the method")] that is written just before the method is invoked, without this export tag, the application would encounter an error when the worker clicks on the *log in* button, labeled "Logga in". This is a general rule for all methods that are user input triggered, the compiler will not process such methods without the export tag and therefore any such method will fail when called. In addition, these methods must be public, not private.

```
public WebClient dbClient = new WebClient();
public Uri       = new Uri("_____________________________________");


[Export("logInClick")]
0 references
public void logInClick(View v)
{
    FindViewById<Button>(Resource.Id.buttonLogIn).Enabled = false;

    try
    {
        //Create a namevaluecollection for use in sql query
        NameValueCollection parameters = new NameValueCollection();

        parameters.Add("userName", FindViewById<EditText>(Resource.Id.editTextUser).Text);
        parameters.Add("passWord", FindViewById<EditText>(Resource.Id.editTextPass).Text);
        parameters.Add("query", "logIn");
        dbClient.UploadValuesCompleted += client_UploadValuesCompleted;
        dbClient.UploadValuesAsync(      , parameters);
    }
    catch (Exception)
    {
        throw;
    }
```

Figure 9. The method that initially connects to the Web service, some elements have been hidden for company privacy reasons.

When a worker logs in, the Web service checks the workers' credentials and if they match the information stored in the DB, user information along with information about every work order from the past three months is returned as a string to the application. After successfully logging in, the page illustrated in Figure 10 is shown.

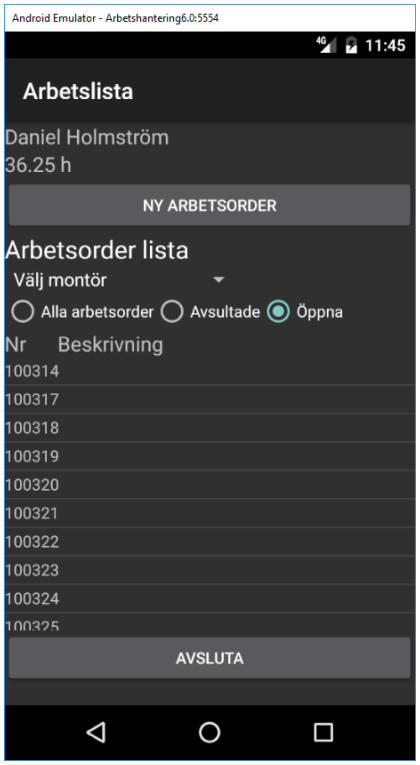

Figure 10. The main page of the application showing ongoing work orders. The contents of work orders have been hidden for privacy reasons.

To utilize the information sent back from the DB, one needs a way to extract the unique fields of the returned string. Since the individual fields are separated by a semicolon (;), it is possible to loop through the string using a few simple “foreach” - loops that break on every semicolon. In addition, to separate queries from one another, the vertical bar character (|) is utilized. Figure 11 displays a part of the code used when a worker logs in to the application.

```
private void client_UploadValuesCompleted(object sender, UploadValuesCompletedEventArgs e)
{
    try
    {
        returnVal = Encoding.GetEncoding(28591).GetString(e.Result);
        //Check if return string is empty (= no such user, invalid credentials or user is inactive)
        if (returnVal != "")
        {
            stringPos = 0;
            stringPrevPos = 0;
            stringDiv = 0;

            foreach (char c in returnVal)
            {
                //Check for query break in string (= '|')
                if (c.ToString() == "|")
                {
                    nameInfo = returnVal.ToString().Substring(0, stringPos);
                    workOrderInfo = returnVal.ToString().Substring(stringPos + 2, returnVal.Length - (stringPos + 2));
                    break;
                }

                stringPos++;
            }

            stringPos = 0;

            foreach (char c in nameInfo)
            {
                //Get name, position, unit and workhours from nameInfo
                if (c.ToString() == ";" || stringPos == nameInfo.Length - 1)
                {
                    if (stringDiv == 0)
                    {
                        firstName = nameInfo.Substring(0, stringPos);
                        stringPrevPos = stringPos;
                        stringDiv++;
                    }
```

Figure 11. A part of the “foreach” -loop code used to extract user information.

A major part of this application centers on the use of intents, which were briefly introduced in Chapter 4. In order to pass the DB values received from one activity to the next activity, the Bundle() library can be utilized. For instance, by using the “Intent.PutString” command this function generates a “package” of strings of information that can be sent to any other activity and then extracted by using the reverse command “Intent.GetStringExtra”. Figure 12 demonstrates how this was implemented in the “log in” -page.

```
//Send parameters to workList and start it
fullName = firstName + " " + lastName;
var workListOpen = new Intent(this, typeof(workList));
Bundle userParams = new Bundle();
userParams.PutString("name", fullName.TrimEnd(';'));
userParams.PutString("isAdmin", admin.TrimEnd(';'));
userParams.PutString("isWorker", worker.TrimEnd(';'));
userParams.PutString("unit", unit.TrimEnd(';'));
userParams.PutString("workhours", workhours.TrimEnd(';'));
userParams.PutString("worker_id", worker_id.TrimEnd(';'));
userParams.PutString("workOrderInfo", workOrderInfo.TrimEnd(';'));
userParams.PutString("namesArray", namesArray.TrimEnd(';'));
userParams.PutString("refresh", "");
workListOpen.PutExtras(userParams);
StartActivity(workListOpen);
Finish();
```

Figure 12. Using intents in order to pass information on to other activities.

In this application, it was very important to keep track of the parameters that pass information between activities. Since some information was only acquired once from the DB, such as the names of the workers, the parameters needed to be passed on every time a new activity started in order preserve the information along the way.

5.2.3 Concerns Regarding Web Services

One issue that immediately surfaced when designing the Web service was the apparent lack of security measures. Since the idea behind Web services is that one is able to connect to resources on a private network from a public network, security should be one of the main focuses when implementing them. The immediate problem is that connections to the Web service can be established by using simple http only, meaning they are not protected or encrypted in any way thus opening a way for potential attackers. Using https is preferable, however, it was decided that this was not enough on its own and the DB needed more protection.

After doing some research on the matter, a few solutions came up. Most people recommended to use so called prepared statements to combat SQL injections. An SQL injection refers to a malicious procedure, often when a user input is required that runs an SQL query in the background, that makes it possible for an attacker to insert (inject) an additional command at the end of the query. This additional command could for instance be “drop table x” or simply something like commenting out the latter part of the query by adding the following: “’—“. The drop table command basically deletes the

whole DB table along with its contents which could have catastrophic consequences since a large amount of data could be lost forever (Microsoft TechNet 2011). This would of course require the attacker to have knowledge about the names of the tables within the DB. A prepared statement works in such a way, that instead of directly using real input data for table fields, placeholder names are specified and they can be in the form of simply a “?”. The prepared statements are then sent to the DB for execution only once and then cached for later reuse. In theory, this should prevent SQL injection from ever occurring since the parameters need to be properly escaped. This also improves the query execution times since the query itself does not need to be resubmitted to the DB every time it executes (W3Schools, 2017).

With security in mind, using prepared statements seemed like a step in the right direction, however, most of the examples were written for MySQL -based solutions, not Microsoft SQL server. When it comes to a Microsoft SQL server though, finding information and examples on prepared statements was quite challenging. On Microsoft’s website however, they recommended validating all inputs before executing the query. This includes validating data size, type or content and using stored procedures. A stored procedure is similar to a prepared statement in the sense that it is created and stored on the DB server only once and then accepts incoming parameters only, so one never needs to send an entire query directly to the DB. (Microsoft TechNet 2011)

5.2.4 Setting up Stored Procedures

Before one begins to import the SQL queries directly to the stored procedures, there are a couple of things that should be kept in mind. It is imperative that one implements a correct syntax when assigning variables to a specific field in the DB. For example the following query: “SELECT * FROM UserTable WHERE UserName = ‘’’ + @username + ’’’ AND PassWord = ‘’’ + @password + ‘’’;” contains two variables from a user input, @username and @password. As such, the query would work fine, however, if one were to write “Brian’—“ as our user name input variable, the rest of the query would be commented out. This means that regardless of what the user input is in the password field, if the DB contains a user called Brian access is granted without ever checking the password! A better way to construct the above query would be as follows: “SELECT * FROM UserTable WHERE UserName = @username AND PassWord = @password;”. In this case even if a user submits the value “Brian’—“ as the user name variable, it will

only be treated as a user input and not as a part of the query itself. (Microsoft Developer 2011)

Adding a stored procedure to the DB is quite simple and straightforward. After logging in to the Microsoft SQL Management studio tool and selecting the correct database, right clicking on Programmability -> Stored procedures and selecting “Create to new window” will generate a template for adding a stored procedure to the selected DB. It is however important to remember that stored procedures operate around variables which are basically user input sent to the stored procedure for execution. The variables need to be sent to the DB in the exact same order as they are specified in the stored procedure.

5.2.5 Phase Two Testing

The first test version of the application was released to two test workers, both having the position of foremen, in early April 2017. In this version, the workers were able to:

- Log in to the application (DB) using their existing credentials.
- View a list of all work orders from the past month and filter them by:
    - Open: work is ongoing.
    - Closed: work is finished.
    - All: shows all work orders, regardless of status.
    - Field worker: name of the field worker in charge.
- Navigate to some pages: New work order and Add work hours, both of these were empty pages without functionality at this time.

After testing for a few days, the initial feedback from the workers was overall positive, although some improvements were immediately proposed. The test workers wanted the application to display work orders dating further back than just one month, a three month period was requested by both test workers. The test workers were also in agreement that the display of the end date (if applicable) of a work order in the list was

unnecessary since the worker already could filter work orders by their status (open or closed). It was then decided that this field would be removed from the list in order to save screen space. No other major issues were found in the application at this stage, however, the test workers expressed their interest in seeing and trying out more features, such as adding new work orders and work hours in the application.

In summary, after a couple of weeks of testing and receiving feedback the test could be considered a success. Since the first version of the application was officially accepted by the test workers, the project then moved on to phase three.

5.3 Phase Three: Implementation Second Cycle

Next, to continue the development process in the second cycle, the “Add new work order” functionality as well as viewing more detailed information about a specific work order was implemented. A function for writing a work report and taking pictures of worksites with the mobile phone camera was also added. The estimated amount of time for this phase was around one month.

5.3.1 Adding New Work Order

The procedure of adding a new work order actually consists of a three step process, each step is represented by a new application window with back and *next* buttons, labeled “Tillbaka” and “Nästa” respectively. The exceptions are the first window where *cancel* button, labeled “Ångra”, is in place of the *back* button and the third window where a *save* button, labeled “Spara”, is in place of the *next* button. Clicking the button for adding a new work order, labeled “Ny arbetsorder”, will open up a new application window where workers can input basic information about the work order, such as work order number and account number information. All input fields in this window are mandatory and cannot be skipped as seen in Figure 13.

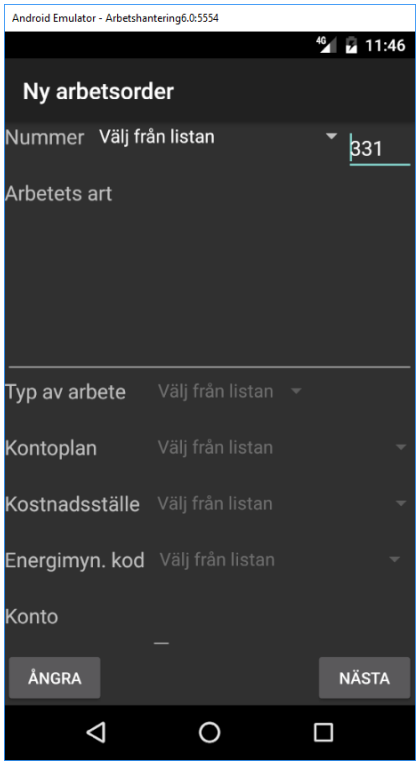

Figure 13. The initial page of the process for adding a new work order.

All of the input fields using a “drop down” style list were created by using a control called a “spinner”. Since these “drop down” lists consist of predefined values, an adapter needs to be used in order to extract information stored within an array and add it to the spinner. Figure 14 shows how this was implemented. The “if” clause was utilized in order to filter the results of a work order according to the logged in worker by setting the default value of the “drop down” list.

```
//Create spinner and bind to mainWorker
Spinner spinner = FindViewById<Spinner>(Resource.Id.mainWorker);

spinner.ItemSelected += new EventHandler<AdapterView.ItemSelectedEventArgs>(spinner_ItemSelected);
var adapter = new ArrayAdapter(this, Android.Resource.Layout.SimpleSpinnerDropDownItem, orderNamesArray);

adapter.SetDropDownViewResource(Android.Resource.Layout.SimpleSpinnerDropDownItem);
spinner.Adapter = adapter;

if (unit == "el" || unit == "service" || unit == "heat")
{
    int spinnerPosition = adapter.GetPosition(Intent.GetStringExtra("name"));
    spinner.SetSelection(spinnerPosition);
}
```

Figure 14. Binding an array adapter to a spinner control.

The work order number is a six digit number where the first three numbers represent a cost place and an account plan, these are chosen from a dropdown style list of availa-

ble numbers. The latter three numbers however, are represented by a continuous number system that start from 100 (one hundred), this resets every year so it is possible to find the same number from two or more different years.

But, since work order numbers must be unique during the same year, it was important to implement some functionality that prevents duplicate number input by accident. Therefore, one needed to implement a DB query that picks the highest available number based on the last three digits and then adds one (+1) to this number. The code in Figure 15 shows how this was implemented in the application. This query is not only called when the worker clicks on the *add new work order* button, but also just before saving a new work order to make sure the highest available number matches the input data. The application again uses the NameValueCollection library for sending parameters to the DB. Note that this method is a general DB query method, meaning a parameter needs to be given to this method (queryName) in order for the method to choose the correct parameter(s) to send to the Web service.

```
private void checkFreeNr()
{
    try
    {
        parameters.Clear();
        parameters.Add("query", "nextFreeNr");
        dbClient.UploadValuesCompleted += client_UploadValuesCompleted;
        dbClient.UploadValuesAsync(      , parameters);
    }
    catch (Exception)
    {
        throw;
    }
}
```

Figure 15. The method that sends the request for the next free work order number to the Web service.

After the worker has filled in the basic information about the work order and clicked on the *next* button, a second window appears where more detailed information about the work order can be defined. These include, among others, the name of the client, address, social security or company number, starting date, name of the field worker in charge and an estimated amount of work hours. None of these fields are mandatory at this stage and can be left blank if necessary.

Again, after the worker has clicked the *next* button on the second window, a third window opens. This window merely displays a summary of all the information the worker has filled in. The worker now has a chance to check if everything seems to be in order and in that case, the worker can then proceed to click the *save* button, which will save the new work order to the DB and return the worker to the main window of the application. If something is amiss with the values, the option of going back one step is still available before saving by clicking the *back* button. Figure 16 shows a part of the method used for either getting the next free number or updating and saving a work order to the DB.

```
private void queryDB(string queryName)
{
    if (queryName == "nextFreeNr")
    {
        parameters.Clear();
        queryID = queryName;
        checkFreeNr();
    }

    else if(queryName == "saveToDB" || queryName == "updateToDB")
    {
        queryID = queryName;

        try
        {
            parameters.Clear();

            checkFreeNr();

            parameters.Add("query", queryName);
            parameters.Add("workOrderNr", workOrderNr.ToString());
            parameters.Add("description", description.Replace("\"", "").Replace("\'", ""));
            parameters.Add("account", accountNr);

            if (unit == "el")
            {
                parameters.Add("isEl", "1");
            }

            if (unit == "service")
            {
                parameters.Add("isService", "1");
            }

            if (unit == "heat")
            {
                parameters.Add("isHeat", "1");
            }

            parameters.Add("printedBy", name);
```

Figure 16. Part of the method for saving or updating a work order to the DB.

As can be seen from the code in Figure 16, different parameters are added to the DB query depending on if the worker has recently opened the page for adding a work order

or has clicked on the button for saving the work order. The information stored in the text string “queryName” is used by the Web service in order to determine what stored procedure should be executed.

5.3.2 Character Conversion Issues with Database

After saving the very first work order to the DB it became clear that some fields had incorrectly stored values and characters in the DB. The affected fields were the end and start dates, which were all given a default value of “01-01-1900” when no date was specified in the application. This was actually quite easy to remedy since the issue was down to the PHP script interpreting null values as empty strings, causing the DB to default these empty strings to “01-01-1900”. By adding a CASE WHEN clause to the DB query it was possible to force a null value entry directly to the DB when the input was an empty string.

The second issue was a little more complex to solve. For some reason, all Scandinavian letters (å, ä, ö) did not appear correctly in the DB, rather as unintelligible symbols. This was obviously a character encoding issue, but finding out whether the problem was in the application code itself, the PHP-scripts or the ODBC connection to the DB was a lot more work than anticipated. The application code was ruled out after using the PHP print to file function on the parameters that were sent to the web service. There one could clearly see that the Scandinavian letters were still correctly displayed so the problem had to be the ODBC connection to the DB. After several days of research and trials without any result, the decision was made to use SQLSRV instead of ODBC. The SQLSRV works in a similar fashion to ODBC but it is possible to specify the character set in the connection string parameters of the SQLSRV_connect command (Johnro 2012). Setting the character set to UTF-8 in the connection string solved the problem and å, ä, ö were displayed correctly in the DB.

5.3.3 Edit or View More Details about Specific Work Order

If a worker wanted to add a work report, edit the contents or see more details on a specific work order they can click on the work in the list of work orders on the main page. This opens up a new page, shown in Figure 8, where the work number and the description of that work order is displayed on the top half of the screen. Below this information, four buttons are visible. The topmost button adds work hours to the work order, the

following one edits the work order contents, the third open up a new page where the worker can write a work report or add a picture and the final button closes the current page. The button for adding work hours will remain without functionality for now since it is a part of the fourth phase of this project.

Figure 17. The page for editing and viewing more details about a work order. Some contents has been removed for privacy reasons.

In order to change details such as description, start date and estimated amount of work hours, the worker can press the edit work order contents button, labeled “Ändra arbetsorder”. This will open the same page as described in Chapter 5.3.1, sending information about all filled in fields as parameters to that page. The worker can then go through the same three step process and save the updated information to the DB.

Clicking on the button for writing a work report and adding a picture, labeled “Skriv arbetsrapport / ta bild”, opens up a new page, shown in Figure 18. Here the worker is able to write an optional report regarding the progress of the work order or any other particular things that need to be observed. The top most button on this page is meant for taking a new picture but will remain without functionality for now. The one below that saves any changes made to the work report to the DB and the last button closes this

page and goes back to the editing and viewing more details about the specific work order.

Figure 18. The page for writing a work report and adding a picture.

The initial idea in this phase was to also add the functionality for taking pictures and then storing them to the DB. However, due to both a pressed schedule and running into some issues with the varbinary(max) column, which is the recommended way of storing images in the DB, this functionality will instead be implemented at phase five of the project (Microsoft TechNet 2013).

5.3.4 Phase Three Testing

The testing for this phase had the same structure as the previous one, two foremen at EE tested the product and gave feedback on the new functionality. In addition to the functions available in the phase two testing, the workers were now able to:

- Add new work orders to the DB.

- Click on work orders in the list of the main application window to open up more details about that particular work order.

- Edit an existing work order.

- Write and save a work report.

The testing officially started on July 25 2017 and the first feedback received from this testing phase was that the input field estimated worktime was obsolete and could therefore be removed completely. In addition to being unnecessary, the Android control used for this field could not handle decimal inputs either, which meant that half hour work time estimations were not possible to add without using additional mathematical calculations. Another issue was that the user interface looked slightly different depending on the screen size of the mobile phone, for example on a smaller screen, some of the buttons and fields in the add new work order pages were not visible or only partly visible.

Other than this, no further bugs or improvements were brought to light and testing was officially concluded on 25 August 2017. Now, the project shifted focus to the adding of work hours to a work order and dealing with overtime and on call hours as well as specifications of vehicles used during the work time.

5.4 Phase Four: Implementation Third Cycle

At this point, mainly due to summer vacations, the project had fallen a few weeks behind the schedule, it seemed reasonable to aim for a late September or early October release of the final application version at. In this phase viewing capabilities for weekly work hours, adding work hours to a work order and submitting the work hour report to a foreman were implemented. After this phase had been concluded, the application with reservations regarding taking pictures and other potential functions and fixes, could be considered a fully functional product.

5.4.1 Viewing Work Hours

Any work order can never be considered complete without having some sort of specification regarding how many hours a worker has spent on that specific job. Therefore, there is a need to implement an additional UI and page that can both display information regarding existing work hours in the DB and enable the worker to add work hours to a work order. First, the focus was on the page for viewing existing work hour

information. The first thing to do was to add a new button directly above the button for adding a work order, by default, pressing this button, labeled “Granska arbetstimmar”, will take the worker to a new page that displays the status of the workers own work hours for the current week.

This page is similar to the main page of the application since it also consists of a list - type overview of the work hours for the current week with an option for filtering the results by week. Additionally, the sum of remaining hours for the chosen week is displayed above the list so the worker has an easier time to keep track of his weekly work hours. For example, if a worker has a designated 40 hour work week and has worked 16 hours so far on the selected week, the application would display 24 hours remaining above the work hour list. A dropdown style list of available workers is also visible on the topmost part of the page but only a worker with foreman privileges can select and view another workers work hours from the list. For regular workers this list is disabled. At the bottom of the page one can find a button for submitting the work hour report, labeled “Lämna in”, adding new work hours, labeled “Lägg till timmar”, and a *back* button that closes this page and goes back to the main page. Figure 19 shows this page as it appears to the worker.

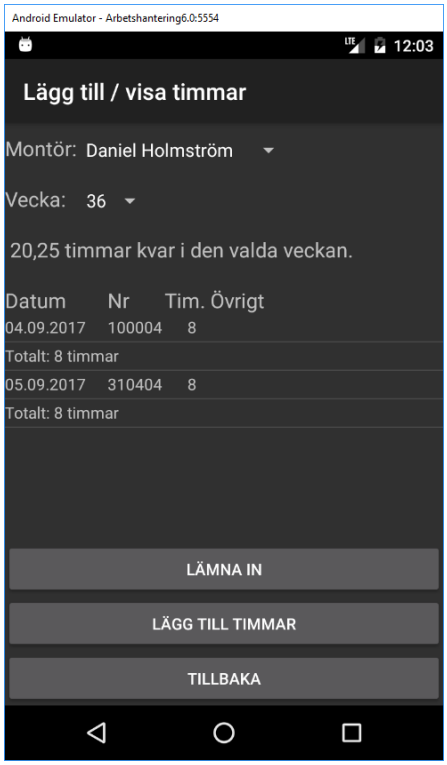

Figure 19. The page for viewing work hours during a specified week.

Inside the work hour list itself, the worker can see information regarding the selected week. This includes a date, the work hour number and how many hours that worker has spent on working that day as well as an optional field for comments. A worker can add several rows of information on the same day if needed. The total sum of hours for each day is displayed as an extra row beneath the last inserted row of work hours for that day.

5.4.2 Dealing with Week Numbers

A stored procedure is utilized to extract the workers names, ID: s and the required weekly work hours for each worker as well as detailed information on past work hours. However, in order to figure out what week of the year a certain date belongs to, the following code is needed: CultureInfo.InvariantCulture.Calendar.GetWeekOfYear(date, CalendarWeekRule.FirstFourDayWeek, DayOfWeek.Monday). The variable date represents the date from which the week number is requested, this variable needs to be in the format of DateTime. The problem with this approach is that in some rare cases, if the last week of the year happens to cross over into the following year and the week number is 1, the application interprets that specific week to be week 53 instead of week 1. In this particular project though, it does not really have any impact on the functionality since week numbers are not utilized from the DB, although in the authors' opinion this issue should be brought to light since it can be beneficial for others to know. The reason why week numbers are shown incorrectly in this case is because the week numbering of the ISO8601 and the .NET are different (Stack overflow 2012). Figure 20 illustrates a simple approach that shows the correct week at the end of a 53 week year and additionally converts the last week to 52 in case the week number is 1. The variable "time" is always set to the 31 December of the current year.

```
public void GetIso8601WeekOfYear(DateTime time)
{
    DayOfWeek day = CultureInfo.InvariantCulture.Calendar.GetDayOfWeek(time);
    if (day >= DayOfWeek.Monday && day <= DayOfWeek.Wednesday)
    {
        time = time.AddDays(3);
    }

    //Check if current year has 52 or 53 weeks
    if (CultureInfo.InvariantCulture.Calendar.GetWeekOfYear(time, CalendarWeekRule.FirstFourDayWeek, DayOfWeek.Monday) != 52)
    {
        if (CultureInfo.InvariantCulture.Calendar.GetWeekOfYear(time, CalendarWeekRule.FirstFourDayWeek, DayOfWeek.Monday) == 1)
        {
            amountOfWeeks = 52;
        }

        else
        {
            amountOfWeeks = 53;
        }
    }

    else
    {
        amountOfWeeks = 52;
    }
}
```

Figure 20. How to extract the correct week number using the ISO8601 standard.

In this project though, the week numbering will always be from 1 to 53, regardless of whether the last week is actually 52 or the last week is divided between two years.

5.4.3 Adding and Editing Work Hours

Pressing the *add new work hours* button will once again open up a new page where the worker can fill in his personal work hours for one day at a time. From the top of the page, the worker will find a dropdown list of work orders that have the status of open, it is mandatory to select a work order from this field. Below this list a spinning style date picker can be found, the default value of this control is the current date. Going even further down the page one will come across a slider style control called a “Seekbar” in Xamarin. The amount of hours is shown next to this slider and the worker can change the value by sliding the pin with his finger to the right or left. The slider can show values between 0 and 24 with a .5 precision, the default value of this control is 0. The last control of this page is another dropdown list. This should only be selected if the worker has worked overtime or has been on call since these scenarios affect the hourly salary. Choosing a value from this dropdown will make two additional time pickers appear below, the idea is that the worker will then choose a start and end time for the overtime or on call hours. This page is illustrated in Figure 21.

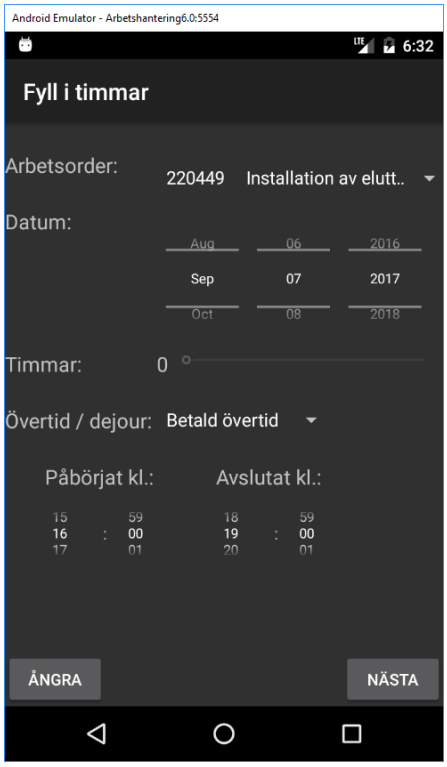

Figure 21. The first page of the adding of new work hours process.

The add hours process follows the same logic as the add new work order process, meaning there is a three step process to go through before the worker can save the new hours to the DB. Clicking on the *next* button on the first page will open the second page where more information regarding the use of a company or private vehicle as well as an additional text field for comments can be found. After the worker has clicked on the *next* button on the second page, the third and last page is shown where a summary of the filled in information can be seen and an option to save it to the DB. The worker can also choose to go back to the previous page by clicking the *back* button in case something is incorrect.

And in case a worker wishes to edit the already existing work hours in the DB, clicking on a row in the list view of work hours will again open up the same three step process that is used when adding work hours. This should be quite familiar to workers already since the same mechanism is in place when editing an existing work order.

The use of “if” clauses have been quite frequent in this project since it is a pretty straightforward way of getting a different outcome depending on the input values. In some cases however, the “if” clause might not be the best or at least most efficient way

of accomplishing variable outcomes. For example, the value for the “on call” list needs to be converted before it is stored in the DB, one of the fastest ways to accomplish this is by using a “switch-case” clause illustrated in Figure 22.

```
switch (overtimeDescription)
{
    case "Betald övertid":
        overtimeDescription = "paid";
        break;

    case "Övertid -> ledig":
        overtimeDescription = "free";
        break;

    case "Dejour":
        overtimeDescription = "service";
        break;
}
```

Figure 22. A “switch-case” –clause that changes the value extracted from the “on call” list.

The “switch-case” clause has a similar structure to the “if” clause, as can be seen in Figure 22. On important difference to note is the “break” command at the end of each “case”, this terminates the execution in order to prevent the method from looping through unnecessary rows of code.

5.4.4 Displaying Decimal Number Values with Seekbar Control

By default, the Seekbar control has no built in support to show numbers containing decimals meaning an alternative approach is needed to simulate the appearance of decimals. One option is to utilize a little mathematics by first setting the maximum allowed number for the Seekbar control to a value ten times higher than the expected maximum, 240. After this, as shown in Figure 23, the inserted number is divided by the static value seekBarSteps, which is set to 5 in this case. In turn, the outcome number is multiplied again with 5 and then finally divided by ten. This number is then shown to the worker as a decimal number, for example, an input of 75 in the Seekbar is displayed as 7.5 hours. (Stack overflow 2011)

```
public void OnProgressChanged(SeekBar seekBar, int progress, bool fromUser)
{
    if (fromUser)
    {
        workedHours = (Math.Round((Convert.ToDecimal((string.Format("{0}", seekBar.Progress))) / seekBarSteps)) * seekBarSteps) / 10;
        FindViewById<TextView>(Resource.Id.textViewSelectedHours).Text = workedHours.ToString();
    }
}
```

Figure 23. The formula for converting a non-decimal number into a decimal number.

This approach works well here since there is no need for a higher number input than 240. That being said, it would have been a less than ideal choice to use in the estimated work hours function that was implemented at first in phase 2, but later skipped. This is because the maximum number there would have been required to be at least 3000, which would have made precision input with one's finger almost impossible, a shift of merely a millimeter or two on the slidebar pin would have resulted in a jump of several hours.

5.4.5 Submitting Work Hour Report

The company expects its workers to submit a summary report of all work hours on a weekly basis. In itself, submitting the work report is quite simple, after the worker has finished filling in all work hours for the week, the worker can simply press the *submit work report* button. This procedure will store all work hours from the given week in another table in the DB and additionally add a time stamp with the date and time of occurrence. This will also enable the foreman to check each workers hours before approval.

5.4.6 Phase Four Testing

On 13 September 2017, as the final testing began for this project, the focus shifted to the adding and editing work hours functionality as well as reviewing the application as a whole. The test group remained the same as before, two workers with foreman position in the company tested the application for about two weeks before giving the final feedback.

In this testing phase, the following functionality was added:

- A page for checking the status of individual work hours including filtering options on worker name and week.

- Ability to add work hours.

- Editing existing work hours by selecting it from the list in the status page for work hours.

- Sending the work report.

The duration of this test was merely five days and at the end of the test, only a couple of bugs were reported. The only feedback received from the test workers were on minor adjustments to the UI and changing the functionality on some controls. For example, on the first page, the rightmost radio button filtered results based on the main worker for that project. This functionality was changed so that it shows the results based on who added the work order instead of who was the main worker. The project now continued to the fifth and final phase.

5.5 Phase five: Finalizing

The project had now reached the final milestone and before it was released for production use, some final adjustments to the code and refinement of the user interface were made. Some bugs were also fixed here since it is important that the application functions without any glitches after this point.

5.5.1 Cleaning up Code

So far, the focus had been merely on getting a functional application running without putting too much effort into optimizing the way variables, pages and memory are handled within the application code. The time had now come to take a closer look at what could be done more efficiently within the system itself.

Starting with the variables, when calling a specific control in this project, the FindViewById<> -command has been used frequently. For example FindViewById<EditText>(Resource.Id.EditText1), this is basically a call to a specific text input field labeled EditText1, but the FindViewById<> command could also be used to call a spinner or seekbar among other controls. However, instead of writing that whole sequence over and over again, it can simply be called once and shortened to a variable, for ex-

ample as follows: _EditText1 = FindViewById<EditText>(Resource.Id.EditText1). After doing this, only the much shorter _EditText1 needs to be called in order to perform different operations on the control.

As mentioned previously, managing activities in order to reduce memory consumption is recommended. An easy way to accomplish this is by always using the Finish() command on the previous activity when starting a new one. The Finish() command disposes of the currently active activity. This of course means that all variables and stored data in the previous activity is lost and the activity itself needs to be recreated every time the worker needs to go back to that page. A way to prevent any important data loss that has a limited impact on memory usage is to send array strings of information, such as work order and work hour lists, to the new activity.

5.5.2 Storing Pictures in Database and Final Bug Fixes

At this stage of the project the idea was to get back and tackle the taking pictures functionality that was put on ice in phase two. However, since the project had already been delayed for several weeks and the author could not reliably predict how long it would take to implement this functionality, it was decided that the add pictures functionality would be put on hold and only implemented after the application has been released for production use. In other words, the add pictures to a work order functionality will now only be mentioned in Chapter six regarding future development needs and not a part of this thesis in any other way.

Surprisingly few bugs were detected throughout the course of this project and the final testing phase was no exception. A few things did, however, come up, for example, when the worker first navigated to the “add new work order” page and directly after this to the “add work hours” page the list of available work orders was empty. This was due to an intent not being sent correctly from and to the “add new work order” page resulting in loss of information. Fortunately, this was quite easy to correct.

On 20 September 2017, the application was officially released for production use, however, the work will still continue in terms of continually improving the application and fixing any further bugs that may show up.

6 Discussions and Conclusions

The overall reception of the Android application has been a positive one. One test user commented: “Now I can see all my work orders regardless of where I happen to be at the time, something that was not possible before”. In general, the workers that acted as testers thought that the adding of work hours functionality would mostly be used by the regular workers whereas the adding of work orders would mostly be used by the work foremen. This was not really surprising since the work foremen are in charge of adding and monitoring work orders while every individual worker is responsible for their own work hours.

When it comes to adding more functionality in the future, the obvious thing to start with is the ability to take pictures of work sites and attach them to a work order. This will be the next challenge for the author to tackle. After this, a review of the current functionality of both the work hours and work order parts of this project would be useful, especially when it comes to the adding of work orders where some input fields might need to be removed while others can be added. Another thing that might come to be requested in the future is a function for exchanging overtime hours into work leave, where 8 hours of overtime translates in one extra day of leave. Though, this is something that would need to be planned carefully before being implemented in order to avoid a chaotic looking UI.

Looking back to December 2016 when this project officially started, it can be concluded that a lot of things have occurred along the way, some events were expected while others came as a complete surprise. The main goals of this project was, as stated in the Introduction, to develop a solution for easy management of work hours and work orders that can be used anywhere and anytime. Judging by the feedback collected from the workers, it is not hard to draw the conclusion that all of these goals have been fulfilled. The Android application is fairly easy to use and understand without too many advanced functions and workers now have access to their work orders and work hours from any location and at any given time.

If at least one lesson is to be taken it should probably be that managing the project schedule is very important and having too optimistic goals for the project will most certainly lead to delays. This is not to say that having optimistic goals is a bad thing, on

the contrary, it is a good thing but only if accompanied by the internal knowledge that chances are very high that the deadlines will be pushed forward at some point.

The outcome of this project has been a positive experience in many ways, having to overcome challenges, prioritize work in order to meet the schedule deadlines, learning about new technologies and finally having a sense of accomplishment when the product was finalized.

References

Burton, M. Android App Development for Dummies, 2015.

EE Daily news, 2011 Global Unichip moves to ASIC design services, acts as a virtual IDM with TSMC. URL: http://www.eedailynews.com/2011/10/ Accessed 12 April 2017.

Johnro, 2012 PHP: sqlsrv problems with UTF-8 values? Set your CharacterSet option!. URL: http://www.johnro.net/2012/07/25/php-sqlsrv-problems-with-utf-8-values-set-your-characterset-option/ Accessed 24 May 2017.

Microsoft Developer, 2011 Do Stored Procedures Protect Against SQL Injection?. URL: https://blogs.msdn.microsoft.com/brian_swan/2011/02/16/do-stored-procedures-protect-against-sql-injection/ Accessed 14 April 2017.

Microsoft TechNet, 2011 SQL Injection. URL: https://technet.microsoft.com/en-us/library/ms161953(v=sql.105).aspx Accessed 14 April 2017.

Microsoft TechNet, 2013 Storing Images and Photos in SQL Server. URL: https://social.technet.microsoft.com/wiki/contents/articles/6519.storing-images-and-photos-in-sql-server.aspx Accessed 21 July 2017.

Panigraphy, N. Xamarin Mobile Application Development for Android – Second Edition, 2015.

Stack overflow, 2010 Android: What is better – multiple activities or switching views manually?. URL: https://stackoverflow.com/questions/2072244/android-what-is-better-multiple-activities-or-switching-views-manually Accessed 28 March 2017.

Stack overflow, 2012 Get the correct week number of a given date. URL: https://stackoverflow.com/questions/11154673/get-the-correct-week-number-of-a-given-date Accessed 28 August 2017.

Stack overflow, 2011 SeekBar with decimal values. URL: https://stackoverflow.com/questions/6197674/seekbar-with-decimal-values Accessed 28 August 2017.

W3Schools, 2017 PHP Prepared Statements. URL: https://www.w3schools.com/php/php_mysql_prepared_statements.asp Accessed 16 June 2017.

Xamarin, 2017 Accelerating Android Emulators. URL: https://developer.xamarin.com/guides/android/getting_started/installation/accelerating_android_emulators/ Accessed 17 March 2017.

Xamarin, 2017 Guides – Android. URL:
https://developer.xamarin.com/guides/#android Accessed 17 March 2017