



Universidad De Valladolid

E.T.S. DE INGENIERIA INFORMATICA

Grado en Informática

Aplicación móvil Android para la gestión de peticiones de libros

Alumno: Vincent Risack

Tutor: Dr. Quiliano Isaac Moro Sancho

Abstract in English

This bachelor thesis is about an Android application for managing book purchase requests. It will cover everything from the Design and coding to the implementation and usage of the application. The Usage of the application will be explained through a user manual. The code will be explained in the general description of the product. The user requirements are also explained in this book. There's also an explanation for the minimum required specifications of the mobile phone and an installation manual. There will be added illustrations to show and explain the design and some of the used code.

Keywords

Android, Mobile book purchase request, managing, scan barcode, ISBN

Citation

Vincent Risack: Aplicación móvil Android para la gestión de peticiones de libros, Bachelor Thesis, Valladolid, E.T.S. De Ingenieria Informática, 2015

Announcement

“This final project was an exam. The discovered errors during the presentation are not included.”

Dedication

I would like to thank the professors in Vives Ostend for giving me the opportunity to finish my studies with an Erasmus+ program. More precisely would I like to thank my internal promoter Cordemans Tom, and supervising professor Vanhee Luc. I would also like to thank my parents for allowing me this amazing experience. Also would I like to thank my girlfriend. It would not have been possible for me to finish my studies (and this final project) without their constant motivation and support.

Also would I like to thank my Spanish Supervisors Dr. Quiliano Isaac Moro Sancho and Javier Bastida Ibáñez for their help choosing the project and their expertise on the subject. Whenever I had a problem I was always welcome to ask them for help to find a solution. This final project was a perfect way to get to know a whole new working culture. This will be a great asset on the job market later. I have also met some friends here, who I will cherish for the rest of my life. This was a “once in a lifetime” experience.

© Vincent Risack

This work was created as a thesis at the University of Valladolid, Departamento de Informática. Copyright and law protect this work, its use without authorisation by the author is illegal, except in cases defined by law.

Table of content

1	INTRODUCTION	3
2	GENERAL DESCRIPTION OF THE PROJECT	7
2.1	OBJECTIVES	7
2.1.1	<i>Create a log in.....</i>	7
2.1.2	<i>Implement barcode scan application.....</i>	7
2.1.3	<i>Make connection using an API (Application Programming Interface).....</i>	7
2.1.4	<i>Retrieve the book information.....</i>	7
2.1.5	<i>Have the possibility for entering book data manually.....</i>	8
2.1.6	<i>Create a preview to check the retrieved information.....</i>	8
2.1.7	<i>Make connection between the application and the server.....</i>	8
2.1.8	<i>Create a link to the Google Books Database.....</i>	8
2.1.9	<i>Request a book.....</i>	8
2.1.10	<i>Create the possibility to cancel a requested book</i>	8
2.1.11	<i>Oblige the user to select a book use.....</i>	8
2.1.12	<i>Make sure the application respects the android conference.....</i>	9
2.2	METHODOLOGY.....	9
2.2.1	<i>Connecting to the database: login control.....</i>	9
2.2.2	<i>Connecting to Google Books Database</i>	10
2.2.3	<i>Connection to the database: storing book information.....</i>	11
2.2.4	<i>Connection to the database: cancelling book request.....</i>	12
3	GENERAL DESCRIPTION OF THE PRODUCT.....	13
3.1	PROGRAMMING LANGUAGES AND TOOLS.....	13
3.1.1	<i>Android Studio.....</i>	13
3.1.2	<i>Xampp</i>	13
3.1.3	<i>Java (android).....</i>	13
3.1.4	<i>PHP</i>	13
3.1.5	<i>MySQL.....</i>	13
3.2	FLOWCHART FROM COMPLETE APPLICATION	14
3.3	EXPLANATION OF THE LAYOUT	16
3.3.1	<i>Login activity.....</i>	16
3.3.2	<i>Main activity (library scanner).....</i>	19
3.4	EXPLANATION OF THE CODE	24
3.4.1	<i>Login</i>	24
3.4.2	<i>Main activity (library scanner).....</i>	29
3.4.3	<i>Json parser.....</i>	42
4	USER AND SYSTEM REQUIREMENTS.....	44
4.1	USER REQUIREMENTS.....	45
4.1.1	<i>Login credentials.....</i>	45
4.1.2	<i>Internet connection</i>	45
4.2	SYSTEM REQUIREMENTS	45
4.2.1	<i>Minimum required specifications.....</i>	46
4.2.2	<i>Recommended specifications</i>	46
4.2.3	<i>Server requirements.....</i>	47
5	PLANNING	49
5.1	FEBRUARY.....	49
5.2	MARCH	49
5.3	APRIL	50
5.4	MAY	51

6	SOFTWARE TESTS PERFORMED.....	53
6.1	TEST OF THE BOOK RETRIEVAL.....	53
6.2	TEST OF THE CONNECTION AND TRANSMISSION OF DATA TO THE SERVER.....	54
7	USER MANUAL.....	55
7.1	OPEN THE APPLICATION LAUNCHER AND SELECT THE LIBRARY SCANNER APP.....	55
7.2	LOGIN USING THE CREDENTIALS GIVEN TO YOU BY THE SYSTEM ADMINISTRATOR.....	55
7.2.1	<i>Login server not found.....</i>	55
7.2.2	<i>Change the log in server.....</i>	56
7.2.3	<i>Logging in with username, password or both blank.....</i>	56
7.2.4	<i>Logging in with wrong credentials.....</i>	57
7.3	MAIN APPLICATION PAGE.....	58
7.3.1	<i>Retrieve book information online.....</i>	58
7.3.2	<i>Enter the data manually.....</i>	60
7.3.3	<i>Send data to database.....</i>	61
7.3.4	<i>Link to the found book.....</i>	62
7.3.5	<i>Clear the retrieved or manually entered details.....</i>	63
7.3.6	<i>Cancel one of the book(s) you've requested.....</i>	63
7.4	CLOSING THE APPLICATION.....	64
8	INSTALLATION MANUAL.....	65
8.1	DOWNLOAD THE LIBRARYSCANNER.APK TO YOUR COMPUTER.....	65
8.2	TRANSFER THE LIBRARYSCANNER.APK TO YOUR MOBILE PHONE.....	65
8.2.1	<i>Windows computer.....</i>	65
8.2.2	<i>Mac computer.....</i>	65
8.3	ENABLE INSTALLATION FROM UNKNOWN SOURCES.....	66
8.3.1	<i>Android 5.0 (Lollipop) and newer versions.....</i>	66
8.3.2	<i>Android 4.X versions (Ice Cream Sandwich, Jelly Bean and Kit Kat).....</i>	67
8.4	LOCATE THE INSTALLATION PACKAGE ON THE PHONE AND EXECUTE IT.....	67
9	CONCLUSION.....	69
10	LIST OF FIGURES.....	71
11	LIST OF TABLES.....	73
12	BIBLIOGRAPHY.....	73

1 Introduction

Since touchscreen mobile phones or, as we call them, smartphones were invented there have been multiple operating systems that have been developed as well. Nowadays we have three big mobile operating systems: the global market leader Google's Android OS holding a 76.6% market share, Apple's IOS holding a 19.7% market share and Windows Phone with 2.8%. The other mobile operating systems are negligible. This final project will be about the Android OS.

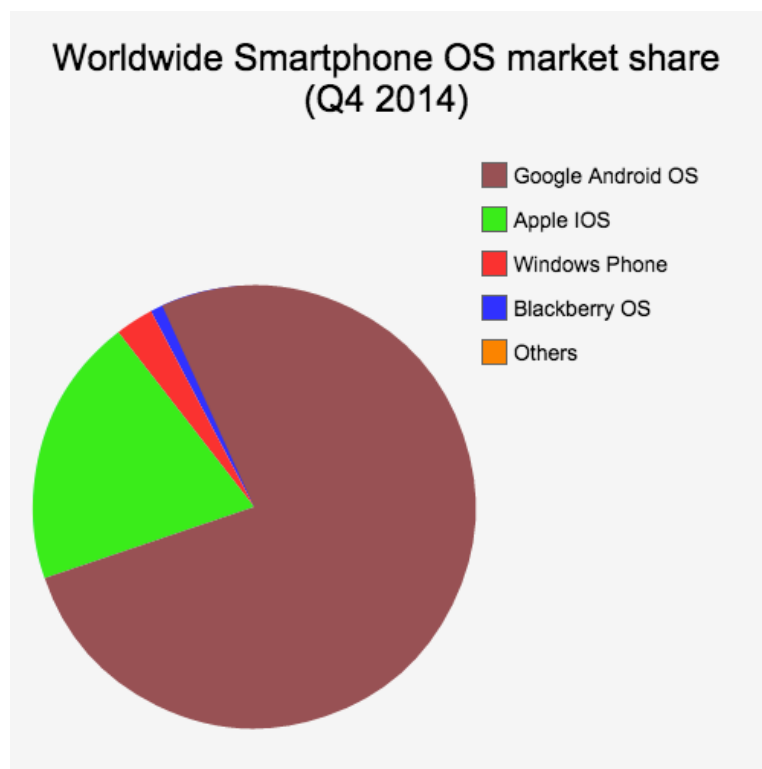


Fig. 1: Graph from the market share (1)

Since the beginning of the development of Android it has been used to create applications to help users make their lives easier. Because of the constant improvement of the hardware (there are already cell phones with quad core processors and up to 4 gigabytes of RAM, bigger screens with up to 4K ultra high definition screens, improved touch screen digitizers with multi touch for up to 10 fingers...) from the cell phones, these applications have as well become more powerful and a lot faster. Even the most processor consuming tasks, which on the first Android based smartphones would have taken up to a minute will now be performed in the blink of an eye.

Imagine you're in a bookstore, or the library of another university you're visiting, or even the house of a colleague or friend. You're browsing through the book collection and all of a sudden you find a book that your university or company does not have. A book that could be very useful either for the other professors, the students or colleagues. Wouldn't it be easier to just be able to scan the barcode and with the barcode send all the details from the book directly to the person responsible for ordering the books? Mobile Internet or Wi-Fi are available almost everywhere and it saves a lot of time (considering you don't have to write all the details down and later type those in an e-mail). This final project will be a great example of how modern technology makes even simple tasks easier and also faster.

I was asked to create a mobile application to manage these book purchase requests made by professors in the university. The application I have developed has the purpose of making life easier for the professors using it. With this application they can, whenever they see a book they're interested in, scan the ISBN (International Standard Book Number) code. The details from the book are then downloaded into their cell phone. If the book can't be found online they can manually edit it. Afterwards they can send it to a database where the University can decide whether or not the book can and will be bought. This saves a lot of valuable time.

For the creation of this application I had to incorporate my knowledge of Android, PHP and MySQL. It was very interesting to see the progress these coding languages have made over the past few years. They have all become more powerful and faster. In the future this kind of applications will be even faster to create and to execute.

The application was designed for Android 4.4 (Kit Kat) but will work on any Android compatible Cell phone From Android 4.0 (Ice Cream Sandwich) and upwards to Android 5.1.2 (Lollipop). It has been tested on devices running Android 4.4.4 (An updated version of Kit Kat) and Android 5.0.2 (An updated version of Lollipop). This application has also been tested on Cyanogenmod CM11S (based on Android 4.4) and Cyanogenmod CM12S (based on Android 5.0). These are custom ROM's based on Google's Android mobile system. Because of the support starting from Android 4.0 till 5.1, this application can be used on 93.2% of the smartphones.

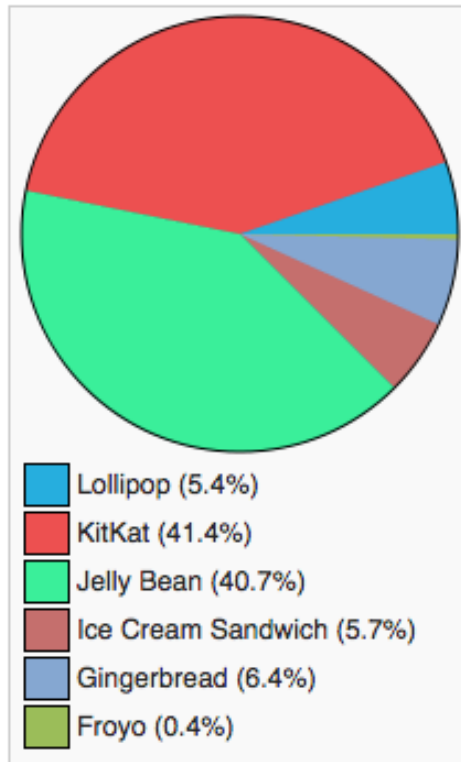


Fig. 2: Graph from the Android version spread as of 04/2015 (2)

The database was written in MySQL and provided to me by Dr. Quiliano Isaac Moro Sancho. During the creation of this final project it was executed on a local virtual server for which the application XAMPP version 5.6.8 was used which incorporates an Apache Server and MySQL. The MySQL database will be managed using MyPHPAdmin that is also incorporated into XAMPP. The PHP version was also version 5.6.8. All the connections have been made to this local server for the ease and also the speed of working on a local network. The application can be edited to connect to an external server while running. This function is to be explained later in this final project.

2 General description of the project

The purpose of this project was to create a bachelor thesis. I had 4 months to complete the project. The project was mainly coded in Android (a java based language) but I have also used PHP and MySQL. The PHP code was used for the server side. It was used to make the connection from the server to the database and make it able to be altered. The MySQL code was used for the actual altering of the tables.

2.1 OBJECTIVES

During this part of the chapter I will explain the objectives of the application I had to create. I will also explain the ways I have decided to fulfil these objectives.

2.1.1 Create a log in

Since this application will be used for making book purchase requests, I had to implement some security measures. For this I decided to create a log in system. Because of this system it would be impossible for users who aren't allowed to make book requests to do so. It was also for this reason that I decided not to create a register user function.

2.1.2 Implement barcode scan application

First of all I had to find a way to create an application that could scan a barcode. After doing some research I have decided to use the intent method to implement an existing barcode scanner into the project.

2.1.3 Make connection using an API (Application Programming Interface)

The application had to be able to retrieve book information automatically. For this I had to be able to make a connection to the Internet. I decided to use the Google Books API since this was well documented. The database from Google Books is also one of the most diversified ones. This was also a great asset for using this API.

2.1.4 Retrieve the book information

For the automatic book information retrieving I had to find a way to retrieve all this information from the Google books database. This was very good documented in the Google Books API documentation.

2.1.5 Have the possibility for entering book data manually

The Google Books database is very large but, of course, it is impossible to have all the books in the world in one database. If the book was not encountered in the database, the user must also have the possibility to enter all the book information necessary for making the request to the server to purchase the book manually.

2.1.6 Create a preview to check the retrieved information

After the book information has been retrieved from the Google Books Database the user must first be able to check the data. If he would encounter an error he must have the ability to make changes to this data before it being sent to the database. The preview will be shown in the main screen of the application.

2.1.7 Make connection between the application and the server

The application has to be able to connect to the server where the database is stored. Without this connection the application would be rather useless since its main objective would be lost. The application sends all the data to a PHP script using the http POST transfer protocol. This connection will be used for both sending the data retrieved from the book and also for sending the cancel requests.

2.1.8 Create a link to the Google Books Database

The book was found in the Google Books Database and the information shown in the preview is not detailed enough for the user. He can now open the link to the book page in the Google Books database. With this link he can view more details about the book and also locations and information on where to buy it.

2.1.9 Request a book

When the user has checked the retrieved or manually entered information he must be able to make the book request. This will send the details to the server. After doing this, the appropriate person can progress the request.

2.1.10 Create the possibility to cancel a requested book

The user must, of course, also be able to cancel a book he has requested. The database had a built in status code for cancelled books. The cancel button will change the status of a book, ordered by the proper user, to "cancelled".

2.1.11 Oblige the user to select a book use

The user must choose the reason for the book request. Will he be using the book for research or for teaching? Without the selection of the reason, the book may not be stored in the database.

2.1.12 Make sure the application respects the android conference

Control the code with the code inspector to make sure no infractions against the Android conference have been made.

2.2 Methodology

In this part of the chapter I will elaborate more on the ways to connect the application to both the Google Books database and the server. I will be using diagrams to substantiate this explanation.

2.2.1 Connecting to the database: login control

The application makes a connection to the specified URL. When the login button is pressed a connection is established and the username and password are sent to the server. The server uses the checklogin.php file to perform the necessary actions.

```
1 <?php
2 mysql_connect("localhost", "root", "") or die(json_encode("cannot connect"));
3 mysql_select_db("books") or die(json_encode("cannot select DB"));
4
5 $username=$_POST['username'];
6 $password=$_POST['password'];
7
8
9 $query = " SELECT * FROM personal WHERE username = '$username'and clave='$password'";
10 $sql1=mysql_query($query);
11 $row = mysql_fetch_array($sql1);
12 if (!empty($row)) {
13     $response["success"] = 1;
14     $response["message"] = "You have been sucessfully logged in";
15     die(json_encode($response));
16 }
17 else{
18     $response["success"] = 0;
19     $response["message"] = "invalid username or password ";
20     die(json_encode($response));
21 }
22
23 mysql_close();
24
25 ?>
```

Fig. 3: Screenshot from checklogin.php

During the first two lines of code the PHP script is connecting to the MySQL server. If it cannot connect a message will be encoded telling the user ("cannot connect"). If the connection is successful the script will try and select the table "books". If this is not possible a message will appear "cannot select DB". The 2 variables username and password are created and the values received from the application through http post are stored.

The PHP script will now perform a query to the table personal to retrieve al the information from the row where the field username equals the value stored in the

variable username and the field clave equals the value stored in the variable password. This information is stored in the variable sql1. The information from sql1 is now being grouped in the variable row. This variable is now checked. If it is not empty it means the username and password have been encountered. The PHP script sends this back in a json-encoded message. However if the row is empty it means the username and password combination has not been detected. The PHP script will send a json-encoded message back to the application informing it that the username or password is invalid.

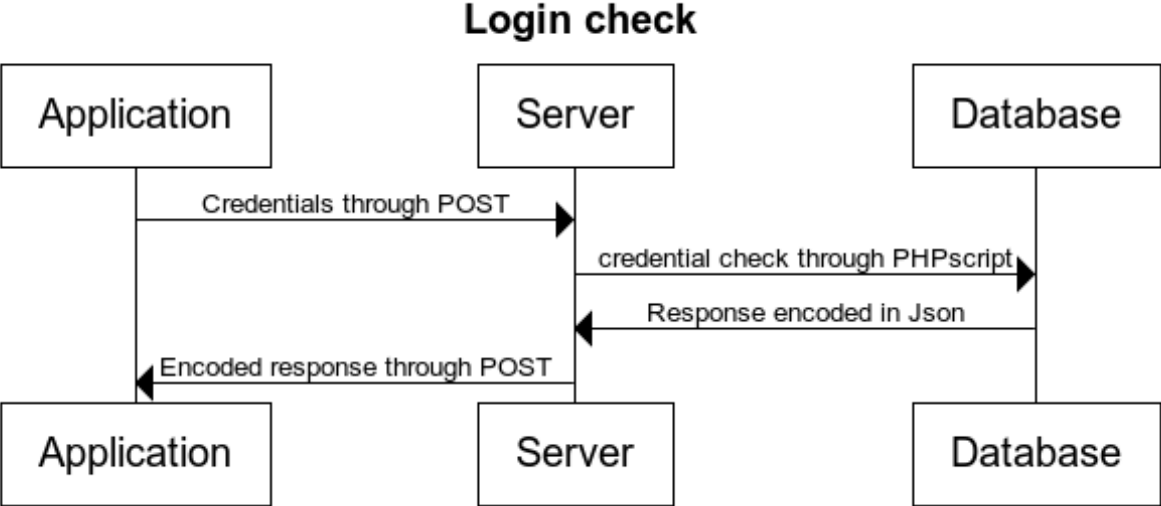


Table 1: Sequence diagram for login.

2.2.2 Connecting to Google Books Database

The application also has to be able to connect to the Google Books Database in order to be able to retrieve the book information. The application uses an http GET method since the details of the request does not have to be confidential. The connection and retrieving of the details happen as shown in the following sequence diagram.

Google Books connection

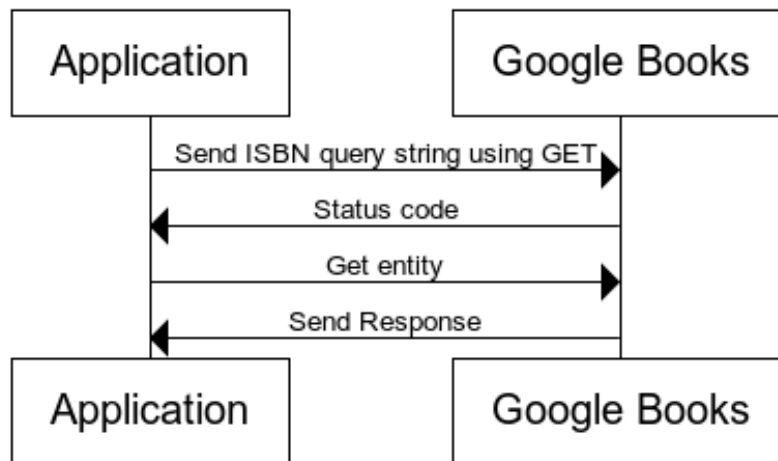


Table 2: Sequence diagram from the connection and retrieving of book details

2.2.3 Connection to the database: storing book information

The connection to the database for sending the book details is the same as for checking the login credentials. The only difference is that there are more parameters being sent and that they are being handled by another PHP script named `addtodatabase.php`. The server will also send a response to whether the storing of the details was successful or not.

```

1  <?php
2
3  mysql_connect("localhost", "root", "") or die("cannot connect");
4  mysql_select_db("books") or die("cannot select DB");
5
6  $author=$_POST['author'];
7  $title=$_POST['title'];
8  $year=$_POST['year'];
9  $isbn=$_POST['isbn'];
10 $usrname=$_POST['usrname'];
11 $docencia=$_POST['docencia'];
12
13 $query = " SELECT * FROM libros_total WHERE autor = '$author' and titulo='$title' and anno='$year' and estado='10'";
14 $sql=mysql_query($query);
15 $row = mysql_fetch_array($sql);
16 if (empty($row)) {
17 mysql_query ("INSERT INTO libros_total (id, pet_id, personal_id, fecha_pet, docencia, autor, titulo, anno,
18     editorial_id, isbn, precio, entidad_pago_id, proveedor_id, fecha_ped, fecha_lleg, libro_id, localizado,
19     signature, incidencias, estado, username) VALUES (DEFAULT, NULL, NULL, NOW(), '$docencia', '$author','$title',
20     '$year', NULL, '$isbn', NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, '10', '$usrname')");
21 $response["message"] = 1;
22 }
23 else {
24 $response["message"] = 0;
25 }
26 }
27 die(json_encode($response));
28 mysql_close();
29 ?>

```

Fig. 4: Screenshot from the `addtodatabase.php` script

The working of this script starts the same as the `checklogin.php` script. The difference is that the database will search the table `libros_total` for the book. It will search for a row where the author, title and year are the same as the ones provided by the application. It will also check if the user hasn't requested the book already (Status code 10). If this is the case the script will send back a message "0" which will then be interpreted by the application. If the script doesn't encounter the book in the table it will add the book details into it and send back a message "1".

2.2.4 Connection to the database: cancelling book request

This connection happens the same way as the one to the database for Storing book information and the login control. A different PHP script will once again handle the data received by the server. The script handling the cancel request can be seen in the screenshot underneath.

```
1 <?php
2 mysql_connect("localhost", "root", "") or die("cannot connect");
3 mysql_select_db("books") or die("cannot select DB");
4
5 $author=$_POST['author'];
6 $title=$_POST['title'];
7 $usrnme=$_POST['usrnme'];
8
9 $query = " SELECT * FROM libros_total WHERE autor = '$author' and titulo='$title' and username='$usrnme' and estado='10'";
10 $sql2=mysql_query($query);
11 $row = mysql_fetch_array($sql2);
12
13 if (empty($row)) {
14     $response["message"] = 0;
15 }
16 else {
17     $response["message"] = 1;
18     mysql_query ("UPDATE libros_total SET estado='999'WHERE autor = '$author' and titulo='$title'and username='$usrnme' and estado='10'");
19 }
20 }
21 die(json_encode($response));
22 mysql_close();
23 ?>
```

Fig. 5: Screenshot from the cancelorder.php script

This script will perform a query in the table `libros_total` to search for the book provided by the application. It uses the title, author and username provided by the application. It will also check if the state is still "10". This state means the request has been sent but the book has not been ordered yet. If the book is encountered and the status is "10" the status of the book is updated to "999". The script will than send a message "1" to the application. If the book is not encountered the script will send a message "0" to the application.

3 General description of the product

The application will be described here. I will first explain the tools and programming languages used for this application. Then I will use a flowchart to explain the basic operations of the application. After the flowchart and explication I will explain the code used to create this application. I will start with the XML or layout file and then go to the android code.

3.1 Programming languages and Tools

I will start explaining the tools I've used and will then explain the programming languages.

3.1.1 Android Studio

The Android application was created using Android studio. This is an IDE (Integrated Development Environment) based on the IntelliJ platform. This application is the official android development tool. It can be downloaded from the android developers website. You will also need the SDK (android Software Developer Kit).

3.1.2 Xampp

This was the tool I used for creating the server. It is a basic webserver with MySQL integration. This server runs the PHP scripts and the database.

3.1.3 Java (android)

The code used for creating android programs is Java. Java is a code that doesn't compile to native processor code but is ran on a virtual machine. On android this virtual machine is called Dalvik. Since android 5.0 there's a new virtual machine integrated in the android system called ART (Android RunTime).

3.1.4 PHP

PHP or Hypertext Pre-processor is a scripting language. For this project I have included the SQL injections into the PHP script files. These files are stored on the server and executed when there's data sent to them. It will make the server connect to the database and table and perform the actions.

3.1.5 MySQL

MySQL or My Structured Query Language is the most used open source relational database management system. In the application the server uses a MySQL Database for storing the books.

3.2 Flowchart from complete application

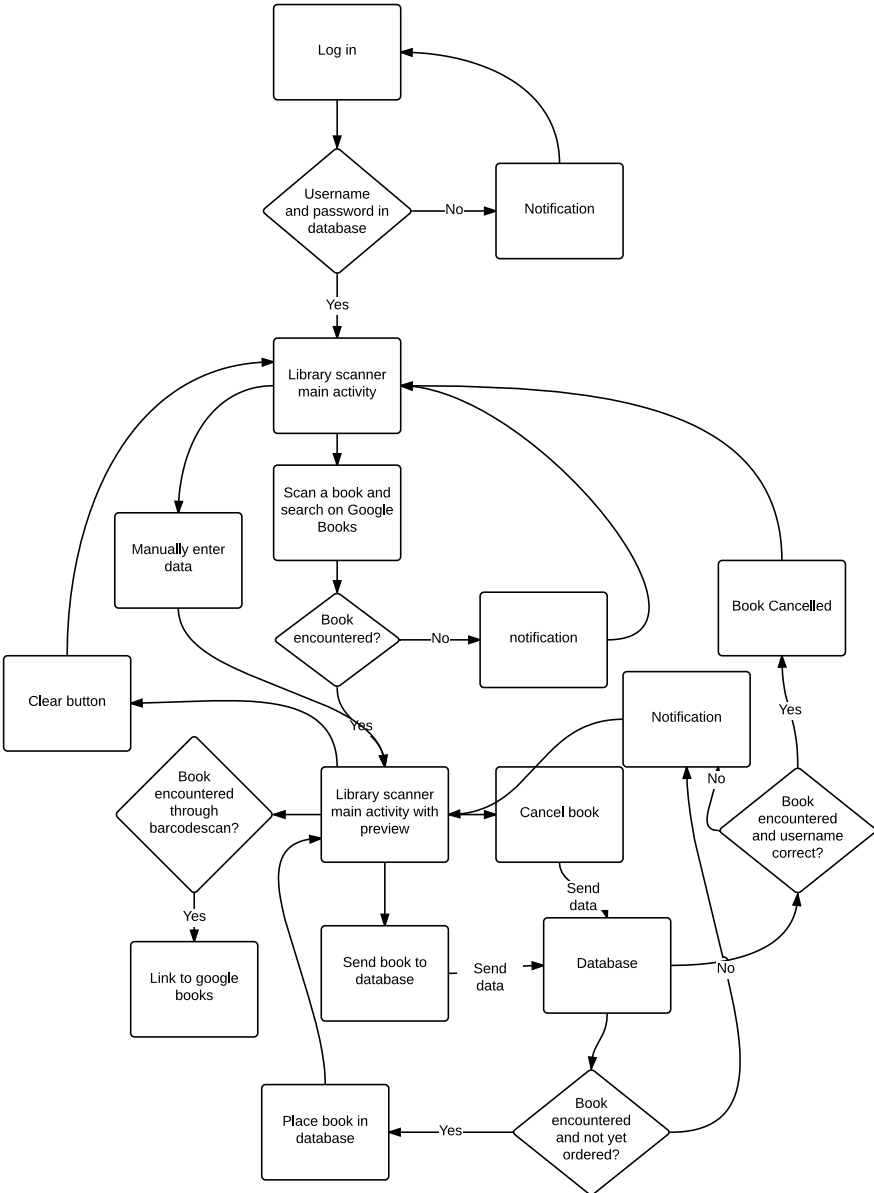


Table 3: Flowchart from the complete application

As you can see from the flowchart, the application boots to the login screen. It is impossible to utilise the application without being logged in. For the log in the application connects to the database in which the log in credentials are stored. If it encounter the login credentials provided it will open the library scanner main activity. If the credentials provided are not encountered the user will get a notification and the application will not open the main activity.

In the main activity you can choose one out of two options. You can choose to scan the ISBN barcode from the book or you can enter the required data manually. When chosen to scan the ISBN, the application sends the data to Google Books. If the book is not encountered in the database the application will give a notification and you will have to enter the data manually. If the application did encounter the book on Google Books You will return to the main activity but with a preview from the book.

On this main activity with preview you have some more options. If some of the data is incorrect you are able to alter it before sending it to the database. If the book was scanned and the information was found in the Google Books database you will now also see a link button named "Web". After clicking this button a web view will open with a link to the page of the book on Google Books.

When chosen to send the book to the database or to cancel the book the same action happens. The details that are filled into the application are sent to the database. Depending on which button you pressed the database will search if it already encounters the book. If you have clicked the "send to database" button and the book is not encountered in the database with a status code of "10" the book will be added. However if the book is encountered in the database with a status code "10" the application will give the user a notification and return to the main activity screen with preview. If you pressed the "cancel button", the application will search the database for the title and author provided by the application and the username of the person logged in. This is to make sure you cannot cancel a book request from another user. If these parameters are encountered in one row, the status from the book will be updated to "999" which is the status code for "cancelled". If the parameters are not encountered in one row the application will give a notification that the user hasn't requested the book.

There's also a clear button available. By clicking on this button all the entered data will be erased. You will basically return to the main activity screen like it was when you have just logged in. This is the basic principle of the application explained through a flowchart. You can exit the application at any time by pressing the home button or the back button.

3.3 Explanation of the Layout

During this part of the chapter I will explain the layout of both the main activity and the login. I will do this using screenshots and the XML files. I will begin explaining the login and then the main activity

3.3.1 Login activity

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <Button
        android:id="@+id/login"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignStart="@+id/password"
        android:layout_alignEnd="@+id/password"
        android:layout_alignLeft="@+id/password"
        android:layout_alignRight="@+id/password"
        android:layout_below="@+id/password"
        android:text="@string/Login" />

    <EditText
        android:id="@+id/username"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignStart="@+id/password"
        android:layout_alignLeft="@+id/password"
        android:ems="10"
        android:focusable="true"
        android:inputType="textVisiblePassword"
        android:hint="Enter username"
        android:imeOptions="actionNext" />

    <EditText
        android:id="@+id/password"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/username"
        android:layout_centerHorizontal="true"
        android:ems="10"
        android:hint="Enter Password"
        android:imeOptions="actionDone"
        android:inputType="textPassword" />
```

```

} <EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/ipAddress"
    android:hint="@string/ipaddr"
    android:text="@string/IpAddress"
    android:layout_marginTop="214dp"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:imeOptions="actionDone"
} android:inputType="textVisiblePassword" />

} <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Change Server IP"
    android:id="@+id/changeIp"
    android:layout_below="@+id/ipAddress"
} android:layout_centerHorizontal="true" />

</RelativeLayout>

```

Fig. 6: Screenshot from the login.xml file

As you can see from the XML file, the login screen is build up in a relative layout. I have only declared the vertical orientation (using `android:orientation="vertical"`) since I have forced the application to only work in the vertical orientation in the android manifest.

In the relative lay out there are two buttons and three edit Text fields. The edit Text fields are used as inputs for the login activity. The buttons are used to confirm the login or to change the IP address. The Password edit Text is aligned in the center of the screen. (`android:layout_centerHorizontal="true"`) and the username edit Text is aligned with the one for the password. This makes sure that even on smaller screens the login and username input will stay central. The edit Text also have a declared input type. For the username I chose text visible password. This is so that people using a non standard application as a keyboard (for example swiftkey) don't get the autocorrect and library. It is not best practise to have the autocorrect on while entering a username or password. For the password input I have chosen the input type text password. This makes sure that when you enter the password it is not visible. I also integrated ime options (Input Method Editor) to change the enter button. (10) While entering the username clicking the enter button will make you go to the password input. While entering the password,

clicking the enter button will make the keyboard disappear since the input of credentials will be done.

I have been using hints (`android:hint=""`) to inform the user what to enter where. When the fields are empty the hint appears in the input field. The IP address field is filled out on the boot of the application with the IP address integrated in the application. This can easily be changed in the code.

The buttons are not hardcoded. This means the values are not integrated in the code of the XML but in a separate file holding all the strings. This is conforming the android convention. Below you can see a screenshot of the lay out as displayed on a smartphone.

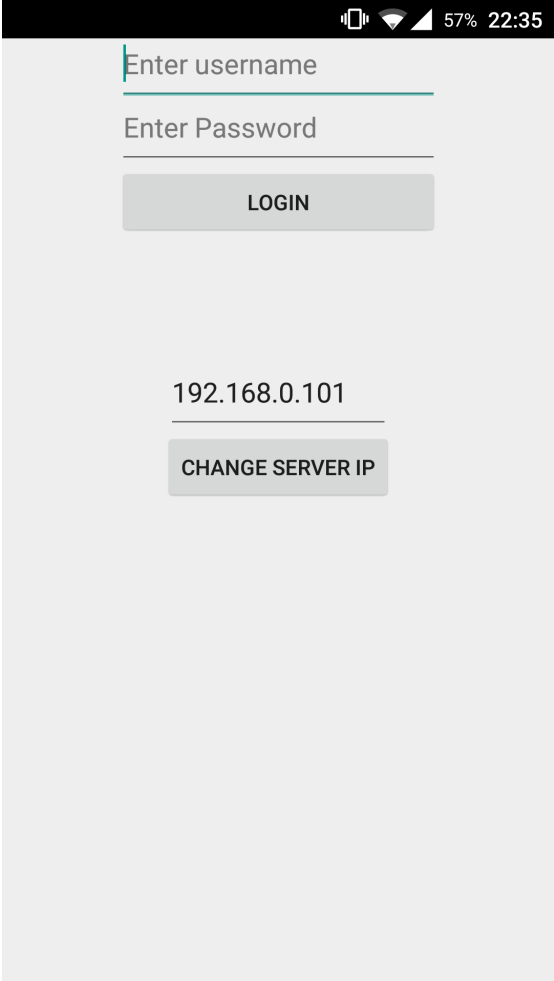


Fig. 7: Screenshot from the layout on mobile phone

3.3.2 Main activity (library scanner)

```
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:padding="10dp">

        <Button
            android:id="@+id/scan_button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_centerHorizontal="true"
            android:padding="10dp"
            android:background="#ff333333"
            android:textColor="#ffccccc"
            android:text="Scan a Book" />

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:orientation="vertical"
            android:layout_below="@id/scan_button"
            android:focusableInTouchMode="true"
            android:id="@+id/linearLayout">

            <ImageView
                android:id="@+id/thumb"
                android:layout_width="60dp"
                android:layout_height="80dp"
                android:contentDescription="Book thumbnail" />

            <EditText
                android:id="@+id/bookTitle"
                style="?android:attr/textViewStyle"
                android:background="@null"
                android:textColor="@null"

                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:textIsSelectable="true"
                android:textStyle="bold"
                android:inputType="textVisiblePassword|textCapWords"
                android:imeOptions="actionNext"
                android:hint="@string/title" />


```

Fig. 8: Screenshot from the first part of the XML file

In this part of the XML layout file I have declared the following components. I started with a scroll view inside which I have put all the following components. This was necessary if the text retrieved was very large, or for mobile phones with smaller screens than the ones the application was tested on. Without this scroll view a part of the retrieved details would be unable to read. It could also be impossible to reach the buttons at the bottom of the page.

In the scroll view I have integrated a relative view that fills up the entire scroll view with a 10 dp (Density-independent Pixels) padding on every side. Inside this relative view I first added the scan button. This button will be used for starting the integrated barcode scanner. The button is aligned in the center of the screen horizontally and the colours are darker than the other buttons.

In this relative view I also integrated a vertical linear layout. This is coded to have the same width as the parent, which is the relative view. Inside of the linear layout there's an image view and an edit text declared. The image view will be used for displaying the book cover. It has a fixed width and height of 60dp * 80 dp. The edit text has been altered to look like a text View. The reason I did this was to make it able to edit the retrieved information if something would be wrong. It has no background colour nor text colour this means it will use the standard ones from the theme. Its height is controlled by the content and the width is the same as the parent. The text is selectable otherwise it would be impossible to alter, and is written in bold. This is to make a clear difference between the title and the author. The input type will make sure the autocorrect option does not open and that the text is written in camel case (every word starts with a capital letter). The enter button has been replaced with a "go to next input" option.

```
<EditText
    android:id="@+id/bookAuthor"
    style="?android:attr/textViewStyle"
    android:background="@null"
    android:textColor="@null"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textIsSelectable="true"
    android:inputType="textVisiblePassword|textCapWords"
    android:imeOptions="actionNext"
    android:hint="@string/Authors" />
```

```

<EditText
    android:id="@+id/book_date"
    style="?android:attr/textViewStyle"
    android:background="@null"
    android:textColor="@null"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textIsSelectable="true"
    android:inputType="numberDecimal"
    android:maxLength="4"
    android:imeOptions="actionNext"
    android:hint="@string/Date" />

<TextView
    android:id="@+id/bookDescription"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textColor="#ff000000"
    android:textIsSelectable="false" />

<LinearLayout
    android:id="@+id/starLayout"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="horizontal">

    <TextView
        android:id="@+id/bookRatingCount"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textColor="#ff2d6994"
        android:textIsSelectable="true"
        android:textStyle="italic" />
</LinearLayout>

```

Fig. 9: Screenshot from the second part of the XML file

After the previous lines of code this text will explain the part of the XML file above. The edit Text for the Author(s) of the book is declared the same way as the one for the title. The big difference is that the text is not written in bold. The enter button has also been replaced with the “go to next input” option.

The next edit Text declared is the one for the publication year. It uses the same style as the title and author edit Text. The input has been changed to number decimal. This is to make sure only numbers are entered. The text is selectable to make it possible to alter the text if the one retrieved would be incorrect. The maximum number of decimals

entered is 4. This is because only the year is important to the application. The enter button has also been replaced with the “go to next input” option.

The text View is used to display the description from the book. I have not changed this to an edit Text because it is used for giving extra information that will not be used by the application. The text colour is slightly lighter than the rest of the texts. It is not selectable because the application will not use it. The height is adjusted by its content because you can never know how long a description will be. The width is the same as his parent.

Inside of this linear layout there’s another linear layout but this one is declared horizontal. The width and height of this layout are declared by its content. This linear layout will be used to show the stars from the rating. Inside of it there’s a text View. It is inside this text view that the stars will be displayed.

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal">

    <Button
        android:id="@+id/SendDataButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Send Data"
        tools:ignore="ButtonStyle"
        android:layout_weight="1" />

    <Button
        android:id="@+id/LinkButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/link"
        tools:ignore="ButtonStyle"
        android:layout_weight="1" />

    <Button
        android:id="@+id/ClearButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="clear"
        tools:ignore="ButtonStyle"
        android:layout_weight="1" />
</LinearLayout>
</LinearLayout>
```



```

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Cancel Order"
    android:id="@+id/CancelButton"
    android:layout_below="@+id/LinearLayout"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:layout_alignRight="@+id/LinearLayout"
    android:layout_alignEnd="@+id/LinearLayout" />
</RelativeLayout>

```

```

</ScrollView>

```

Fig. 10: Screenshot from the third part of the XML

In this last part of the XML the linear layout for the buttons is declared. The orientation of this linear layout is horizontal. The width is matching the parent and the height is to fill the parent. There are 3 buttons declared inside of this linear layout. The link button is hidden until the book has been encountered in the Google Books database. Each button will have a width equal to 1/2 of the width from the row when there are only two buttons visible or 1/3 of the width from the row when the three buttons are visible. This is because of the `android:layout_weight="1"` parameter. The buttons all have the same width and height. The text is soft coded on them. Both of the linear layouts are now closed.

The last button is placed inside of the original relative layout. It uses the complete width of his parent and places the text in the center of the button. This is the cancel button. It is placed underneath the linear layout. After this button the relative layout is closed and the scroll view as well.

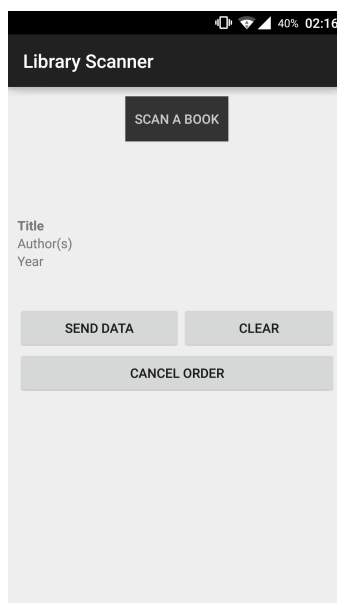


Fig. 11: Screenshot from the main activity implementing the XML

3.4 Explanation of the code

During this part of the chapter I will be explaining the code used for making the whole application work. The covered parts will be: Login, main activity (library scanner) and Jsonparser. The code will be divided in smaller parts, which will then be explained.

3.4.1 Login

```
import android.app.Activity;
import android.app.ProgressDialog;
import android.content.Intent;
import android.os.AsyncTask;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

import org.apache.http.NameValuePair;
import org.apache.http.message.BasicNameValuePair;
import org.json.JSONException;
import org.json.JSONObject;

import java.util.ArrayList;
import java.util.List;

import static android.widget.Toast.LENGTH_SHORT;
import static android.widget.Toast.makeText;
```

Fig. 12: Screenshot from the imports used by the login

The code above shows which parts of the android library are imported and thus used in this part of the application. You can see some of the very standard imports, like android.widget.button, edit Text and toast. But also the import that will be used for performing tasks in the background. This is the AsyncTask import. The intent import makes sure we can start another activity from this one. The apache imports are used to make a connection to the database to check the login credentials. This will be explained later in this part of the chapter.

```

public class Login extends Activity implements View.OnClickListener {
    private EditText username, password, ipAddress;
    private Button loginButton, ipChangeButton;
    private ProgressDialog progressDialog;
    private final JSONParser jsonParsing = new JSONParser();
    private String loginUrl = "http://192.168.0.101/library/checklogin.php";
    private static final String Success = "success";
    private static final String Message = "message";
}

```

Fig. 13: Screenshot from the declaration of the variables used by the login

In this part of the code the variables are declared that will be used globally. We can see 3 edit text fields, two buttons one progress dialog, one Jsonparser class and three strings. The login class will implement the on click listener. It is easier and more readable to declare the on click listener here. It simplifies the code. The last four lines of code will not be viewable for the end user.

```

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.login);
    username = (EditText) findViewById(R.id.username);
    password = (EditText) findViewById(R.id.password);
    ipAddress = (EditText) findViewById(R.id.ipAddress);
    loginButton = (Button) findViewById(R.id.login);
    ipChangeButton = (Button) findViewById(R.id.changeIp);
    loginButton.setOnClickListener(this);
    ipChangeButton.setOnClickListener(this);
}

```

Fig. 14: Screenshot from the declarations and click listeners

Here I will explain the on create method. The saved instance state is used to save the state from the application if it has to be rebuilt. This is useful if the application changes orientation. Because I've forced the application to only display vertical this bundle will always be null. The super method is to call the constructor from the parent class and the set content view is used to link the XML file to the code.

The find view by id is used to declare the buttons, edit texts... in the code above it links the buttons from the XML to the names of the variables in front. For example: the edit text field in the XML file named username will be linked to the username variable.

The set on click listener is used to declare that there will be an event that will happen after being clicked. The "this" keyword is a reference to the current object.

```

@Override
public void onClick(View v) {
    if (v == loginButton) {
        if (username.getText().toString().equals("") && password.getText().toString().equals("")) {
            makeText(Login.this, "Fill in Username and Password!", LENGTH_SHORT).show();
        }
        if (username.getText().toString().equals("") && !password.getText().toString().equals("")) {
            makeText(Login.this, "Fill in Username!", LENGTH_SHORT).show();
        }
        if (password.getText().toString().equals("") && !username.getText().toString().equals("")) {
            makeText(Login.this, "Fill in Password!", LENGTH_SHORT).show();
        }
        if (!username.getText().toString().equals("") && !password.getText().toString().equals("")) {
            new TryLogin().execute();
        }
    }
    if (v == ipChangeButton) {
        loginUrl = "http://" + ipAddress.getText().toString() + "/library/checklogin.php";
        Toast toast = makeText(getApplicationContext(), "Server IP changed to: " + loginUrl, LENGTH_SHORT);
        toast.show();
    }
}

```

Fig. 15: Screenshot from the code used by the login button and the IP change button

After a button has been clicked this part of the code will search for the id of the button that was clicked. If it was the login button, the code from the first “IF” sequence will be executed. If the clicked id equals the IP Change button the code behind the second major “IF” sequence will be executed.

When the login button is pressed this code will convert the text entered in the username and password text fields to a string object. It will then check if neither of these objects is null. If one or both of the string objects is null a warning message (toast) will appear. If both the strings are not null it will execute the try login class. This class will be explained here beneath.

When the IP change button is pressed the variable LoginUrl will be changed to the new value stored in the IP address text field. It will automatically add “http://” + the content of the IP address text field + “/library/checklogin.php”. There will also be a notification displayed at the bottom of the screen telling you that the server IP address has been changed to the variable LoginUrl.

```

private class TryLogin extends AsyncTask<String, String, String> {
    @Override
    protected void onPreExecute() {
        super.onPreExecute();
        progressDialog = new ProgressDialog(Login.this);
        progressDialog.setMessage("Log in in progress!");
        progressDialog.setIndeterminate(false);
        progressDialog.setCancelable(true);
        progressDialog.setCanceledOnTouchOutside(false);
        progressDialog.show();
    }
}

```

Fig. 16: Screenshot from the code before execution of the try login class

The class Try Login implements the AsyncTask. This is to make sure activities can be executed in the background. This part of the code will be executed before starting the in background process. This code will create a progress dialog named progress Dialog. This dialog will be available in the login activity. The message to be displayed is set to “log in in progress!). The progress dialog is also indeterminate. This means that it is impossible to calculate the progress it has already executed. Because of this it will not show a progress bar. The progress dialog is cancellable. This means you can dismiss it by pressing the back button once. The cancel on touch outside is set to false so that the progress dialog is not dismissed when the user clicks next to it. The progress dialog is now being showed. The user can now see it on his screen.

```
@Override
protected String doInBackground(String... args) {
    int success;
    String username = Login.this.username.getText().toString();
    String password = Login.this.password.getText().toString();
    try {
        List<NameValuePair> parameters = new ArrayList<>();
        parameters.add(new BasicNameValuePair("username", username));
        parameters.add(new BasicNameValuePair("password", password));
        JSONObject json = jsonParsing.makeHttpRequest(loginUrl, parameters);
        if (json != null) {
            success = json.getInt(Success);
            if (success == 1) {
                Intent ii = new Intent(Login.this, MainActivity.class);
                ii.putExtra("url", loginUrl);
                ii.putExtra("usrnme", username);
                startActivity(ii);
                finish();
                return json.getString(Message);
            } else {
                return json.getString(Message);
            }
        }
    }
    catch (JSONException e) {
        e.printStackTrace();
    }
    return null;
}
```

Fig. 17: Screenshot from the code executed in the background by the try login class

The code above is the Do in background method. Here is where the actual execution of the main code from the Try Login class is being executed. It will first retrieve the two variables username and password. Then it will add them to an array list of namevaluepairs named parameters. The name value pair is a special combination between key and value. The two variables username and password will be added to the array list parameters. The keys are username and password (the first part of the name value pair) and the values are the variables stored in username and password. The class

will then create a json object. This json object will be equal to the result from the code that has been sent to the makehttprequest part of the jsonParsing Class. This will be explained later in the code.

If the json object returned is not null then the code can continue. The integer success equals the integer retrieved from the json object. If this integer equals "1" then the Main activity will be started through an intent. First the intent has to be created and linked between the login activity and the main activity. With the help of the "add extra" parameters the username and LoginUrl are being transferred from this to the following activity. The code will also return the json.getstring object this will be used in the next part of the code. If the code is impossible to execute a JSONException will be caught. The error log will then show the stacktrace.

```
protected void onPostExecute(String message) {
    progressDialog.dismiss();
    if (message == null) {
        makeText(Login.this, "server not found!", LENGTH_SHORT).show();
    }
    if (message != null) {
        makeText(Login.this, message, LENGTH_SHORT).show();
    }
}
}
```

Fig. 18: Screenshot from the code executed after the try login class

In the last part of the login activity the progress dialog will be dismissed. The user can now no longer see the dialog. If the message object received from the do in background method is null, it means the server has not been encountered. The application will give a notification to the user that the server is not found.

If the message is not null than the application will show the message received. This message will be "logged in successfully!". This is the end of the login activity.

3.4.2 Main activity (library scanner)

```
package be.vives.vincent.libraryscanner;

import android.app.AlertDialog;
import android.app.ProgressDialog;
import android.content.DialogInterface;
import android.content.Intent;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.net.Uri;
import android.os.AsyncTask;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.view.WindowManager;
import android.view.inputmethod.InputMethodManager;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ImageView;
import android.widget.LinearLayout;
import android.widget.TextView;
import android.widget.Toast;

import com.google.zxing.integration.android.IntentIntegrator;
import com.google.zxing.integration.android.IntentResult;

import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.NameValuePair;
import org.apache.http.StatusLine;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.message.BasicNameValuePair;
import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

import java.io.BufferedInputStream;
import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.URL;
import java.net.URLConnection;
import java.util.ArrayList;
import java.util.List;
```

Fig. 19: Screenshot from the imports of main activity

The code displayed before is used to import the different parts of the android library that will be used. There are a lot of standard imports like the buttons, texts... and some more specified ones like the buffered input stream. There are also custom ones like the ZXing integration. These have to be imported to be able to use the ZXing barcode scan functionalities.

```
public class MainActivity extends AppCompatActivity {
    private Button SendButton;
    private Button LinkButton;
    private EditText authorText, titleText, dateText;
    private ImageView coverView;
    private ImageView[] starViews;
    private final JSONParser jsonParser = new JSONParser();
    private LinearLayout starLayout;
    private ProgressDialog progressDialog, progressDialog2, progressDialog3;
    private String scanContent, booktype, loginUrl, username;
    private static final String Message = "message";
    private TextView rating, descriptionText;
```

Fig. 20: Screenshot from the declaration of variables. And Jsonparser class

This is the declaration of the variables used in the main activity. Also the Jsonparser class is declared. This class is used for sending the data to the database and retrieving it. This code will be explained more in the appropriate subchapter. As you can see there are two buttons declared globally, three edit Texts, an image view for showing the cover, and an array of image views for showing the rating. There are also three progress dialogs, four strings and two text views. The main activity will use the App compat Activity. This is a way to have the same exact theming throughout the complete application.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    loginUrl = getIntent().getStringExtra("url");
    username = getIntent().getStringExtra("usrnme");
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    this.getWindow().setSoftInputMode(WindowManager.LayoutParams.SOFT_INPUT_STATE_ALWAYS_HIDDEN);
    Button scanButton = (Button) findViewById(R.id.scan_button);
    Button cancelButton = (Button) findViewById(R.id.CancelButton);
    Button clearButton = (Button) findViewById(R.id.ClearButton);
    SendButton = (Button) findViewById(R.id.SendDataButton);
```

Fig. 21: Screenshot from the on create method

The on create method is very similar to the one used in the login activity. First we will retrieve the values received through the intent. These values will be stored in the variables LoginUrl and username. The layout is linked to the activity_main.xml file. And the keyboard is hidden on the start of the activity. The scan, cancel and clear button are now being linked to the buttons declared in the layout. These three buttons are declared locally. The send button is also linked to the right button declared in the layout.


```

SendButton.setOnClickListener((v) -> {
if (!authorText.getText().toString().equals("") && !titleText.getText().toString().equals("")
    && !dateText.getText().toString().equals("")) {
    final AlertDialog.Builder menuPopup = new AlertDialog.Builder(MainActivity.this);
    final String[] options = {"Teaching", "Research"};
    menuPopup.setTitle("Book usage");
    menuPopup.setItems(options, (dialog, item) -> {
        switch (item) {
            case 0:
                booktype = "0";
                new SendToDatabase().execute();
                break;
            case 1:
                booktype = "1";
                new SendToDatabase().execute();
                break;
        }
    });
    AlertDialog menuDrop = menuPopup.create();
    menuDrop.show();
}
}

```

Fig. 22: Screenshot from the on click listener first part: empty input protector

When the send button is clicked the code will transform the data inputted into the title, author and publication year fields to a string. This string is then compared to make sure it isn't empty. If the strings aren't empty the following code will be executed. On execution of the code there will be a pop up created in the form of an alert dialog. Here the user will have to choose between 2 options: "teaching" or "research". As long as this hasn't been selected the application will not continue. After selecting one of the options the details are now sent to the Send To Database class. This class will be explained later on.

```

if (authorText.getText().toString().equals("") && titleText.getText().toString().equals("")
    && dateText.getText().toString().equals("")) {
    Toast toast = Toast.makeText(getApplicationContext(), "Add the title, the author(s) and the publication year!",
        Toast.LENGTH_SHORT);
    toast.show();
}
if (authorText.getText().toString().equals("") && !titleText.getText().toString().equals("")
    && !dateText.getText().toString().equals("")) {
    Toast toast = Toast.makeText(getApplicationContext(), "Add the author(s)!", Toast.LENGTH_SHORT);
    toast.show();
}
if (!authorText.getText().toString().equals("") && titleText.getText().toString().equals("")
    && !dateText.getText().toString().equals("")) {
    Toast toast = Toast.makeText(getApplicationContext(), "Add the title!", Toast.LENGTH_SHORT);
    toast.show();
}
if (!authorText.getText().toString().equals("") && !titleText.getText().toString().equals("")
    && dateText.getText().toString().equals("")) {
    Toast toast = Toast.makeText(getApplicationContext(), "Add the publication year!", Toast.LENGTH_SHORT);
    toast.show();
}
if (authorText.getText().toString().equals("") && titleText.getText().toString().equals("")
    && !dateText.getText().toString().equals("")) {
    Toast toast = Toast.makeText(getApplicationContext(), "Add the author(s) and title!", Toast.LENGTH_SHORT);
    toast.show();
}
if (authorText.getText().toString().equals("") && !titleText.getText().toString().equals("")
    && dateText.getText().toString().equals("")) {
    Toast toast = Toast.makeText(getApplicationContext(), "Add the author(s) and publication year", Toast.LENGTH_SHORT);
    toast.show();
}
if (!authorText.getText().toString().equals("") && titleText.getText().toString().equals("")
    && dateText.getText().toString().equals("")) {
    Toast toast = Toast.makeText(getApplicationContext(), "Add the title and publication year", Toast.LENGTH_SHORT);
    toast.show();
}
};
}

```

Fig. 23: Screenshot from the click listener second part: warning messages.

The code before is the second part of the on click listener. This is the code I used for the protection of the code. Since the code is not being executed if one or more of the input fields are empty I had to implement a way to warn the user that he needs to fill in the fields. As you can see from the code there are seven different cases. Each case is a different combination of one or more input fields being empty. The user will see a message at the bottom of the screen in the form of a toast explaining which fields need to be filled in for the application to be able to send the data to the database.

```

authorText = (EditText) findViewById(R.id.bookAuthor);
authorText.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        InputMethodManager imm = (InputMethodManager) getSystemService(INPUT_METHOD_SERVICE);
        imm.showSoftInput(authorText, InputMethodManager.SHOW_IMPLICIT);
        getWindow().setSoftInputMode(WindowManager.LayoutParams.SOFT_INPUT_STATE_VISIBLE);
    }
});
titleText = (EditText) findViewById(R.id.bookTitle);
titleText.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        InputMethodManager imm = (InputMethodManager) getSystemService(INPUT_METHOD_SERVICE);
        imm.showSoftInput(titleText, InputMethodManager.SHOW_IMPLICIT);
        getWindow().setSoftInputMode(WindowManager.LayoutParams.SOFT_INPUT_STATE_VISIBLE);
    }
});
descriptionText = (TextView) findViewById(R.id.bookDescription);
dateText = (EditText) findViewById(R.id.book_date);
dateText.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        InputMethodManager imm = (InputMethodManager) getSystemService(INPUT_METHOD_SERVICE);
        imm.showSoftInput(dateText, InputMethodManager.SHOW_IMPLICIT);
        getWindow().setSoftInputMode(WindowManager.LayoutParams.SOFT_INPUT_STATE_VISIBLE);
    }
});

```

Fig. 24: Screenshot from the method to utilise keyboard in the edit texts

The code above is used for making sure the keyboard will open when one of the edit text fields are selected. After clicking on the edit text field the keyboard will pop up. Because of the different lay outs I had to implement this code. The original pop up from the keyboard on the selection of an edit text was not being executed anymore.

```

starLayout = (LinearLayout) findViewById(R.id.starLayout);
rating = (TextView) findViewById(R.id.bookRatingCount);
coverView = (ImageView) findViewById(R.id.thumb);
starViews = new ImageView[5];
for (int s = 0; s < starViews.length; s++) {
    starViews[s] = new ImageView(this);
}

```

Fig. 25: Screenshot from the code creating the layout for the ratings

Here will I edit the linear layout named star layout. With this code there will be an image view created to show the cover, this is named the cover view and an array of image views for displaying the stars after the rating will be retrieved from the Google Books database.

```

LinkButton = (Button) findViewById(R.id.LinkButton);
LinkButton.setVisibility(View.GONE);
LinkButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if (v.getId() == R.id.LinkButton) {
            String tag = (String) v.getTag();
            Intent webView = new Intent(Intent.ACTION_VIEW);
            webView.setData(Uri.parse(tag));
            startActivity(webView);
        }
    }
});

```

Fig. 26: Screenshot from the code for the web view linking Google Books (11)

The link button is declared locally and linked to the link button declared in the XML file. This button is hidden (since it will only appear after the book has been found in the Google Books database) and the on click listener is declared. When the button is clicked, the code will retrieve the URL through the `v.getTag()` command. Then an intent will open a new action view (new screen) with in this screen a web view. The web view will receive the URL through the `setDataUri.parse(tag)`. Because of this the webview will be opened to the right page immediately

```

scanButton.setOnClickListener((v) -> {
    IntentIntegrator integrator = new IntentIntegrator(MainActivity.this);
    integrator.addExtra("SCAN_WIDTH", 640);
    integrator.addExtra("SCAN_HEIGHT", 480);
    integrator.addExtra("PROMPT_MESSAGE", "Scan the ISBN!");
    integrator.initiateScan(IntentIntegrator.PRODUCT_CODE_TYPES);
});

```

Fig. 27: Screenshot from the code starting the barcode scanner

When the scan button is clicked the following code will be executed. A new intent integrator named integrator will be created in the main activity. There will be extra details sent to the scanner. The scan width, height and a message will be passed. Then using the initiate scan command, the barcode scanner will start and show a square from 480*640 dp. The message scan the ISBN will also be displayed.

```

clearButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        titleText.setText("");
        authorText.setText("");
        dateText.setText("");
        starLayout.removeAllViews();
        rating.setText("");
        coverView.setImageBitmap(null);
        descriptionText.setText("");
        LinkButton.setVisibility(View.GONE);
    }
});

```

Fig. 28: Screenshot from the code from the clear button

When the clear button is clicked the code will empty all the edit text's contents to "" or an empty string. This will make sure that the status of the main activity will be exactly the same as when the main activity was first started after the log in. The reason I did this is to make sure you can scan multiple books one after another without having to close the application and log in again. This code helps with the comfort from the user.

```
cancelButton.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        if (!authorText.getText().toString().equals("") && !titleText.getText().toString().equals("")) {
            new CancelBookOrder().execute();
        }
        if (authorText.getText().toString().equals("") && titleText.getText().toString().equals("")) {
            Toast toast = Toast.makeText(getApplicationContext(), "Add the title and the author(s)!", Toast.LENGTH_SHORT);
            toast.show();
        }
        if (authorText.getText().toString().equals("") && !titleText.getText().toString().equals("")) {
            Toast toast = Toast.makeText(getApplicationContext(), "Add the author(s)!", Toast.LENGTH_SHORT);
            toast.show();
        }
        if (!authorText.getText().toString().equals("") && titleText.getText().toString().equals("")) {
            Toast toast = Toast.makeText(getApplicationContext(), "Add the title!", Toast.LENGTH_SHORT);
            toast.show();
        }
    }
});
```

Fig. 29: Screenshot from the cancel button

When clicking the cancel button the code above is executed. The code will check if the title and author input are filled in. It does this by retrieving the input, converting it to a string object and comparing this string to an empty string. There are four cases in total. Three of them are for the different cases of one or more inputs being empty. The user will then receive a notification at the bottom of his screen asking him to fill out the necessary field(s). If all the necessary fields are filled in, the application will execute the cancel book order class. This class will be explained later on.

```
public void onActivityResult(int requestCode, int resultCode, Intent intent) {
    IntentResult scanResult = IntentIntegrator.parseActivityResult(requestCode, resultCode, intent);
    if (scanResult != null) {
        scanContent = scanResult.getContents();
        String scanFormat = scanResult.getFormatName();
        SendButton.setTag(scanContent);
        if (scanContent != null && scanFormat != null && scanFormat.equalsIgnoreCase("EAN_13")) {
            String bookSearchString = "https://www.googleapis.com/books/v1/volumes?" + "q=isbn:" +
                scanContent + "&key=AIzaSyC3VNzsy8LGkmvcvdP2AyKwktovsMNGSMg";
            new GetBookDetails().execute(bookSearchString);
        } else {
            if (scanContent != null) {
                Toast toast = Toast.makeText(getApplicationContext(), "This is not an ISBN code", Toast.LENGTH_SHORT);
                toast.show();
            }
        }
    }
    if (scanContent == null) {
        Toast toast = Toast.makeText(getApplicationContext(), "No ISBN data received", Toast.LENGTH_SHORT);
        toast.show();
    }
}
```

Fig. 30: Screenshot from the onActivityResult class (12)

The on activity result is the code that will be executed after the Scanner sends back the code he has scanned. This class will first check if the result he received is not null. If the scan result is not null, the code will retrieve the content. The local string scanformat will

retrieve the format from the scanned barcode. The ISBN code is a 13 number long EAN code. The code is unique for each book. If the scan content is not null and the scan format is not null either, the code will check if the scan format is an EAN 13 format. If this is the case, the local string booksearchstring gets its value. This is the link to the Google Books API + the query for an ISBN code + the scanned content + the key. This key needs to be retrieved from the developer of the application. Without this key it is impossible to implement the search in the database. The get book details class is now executed and will receive the booksearchstring. If the scan format is not correct the user will receive a notification at the bottom of his screen telling him that the received barcode was not an ISBN. If the scan content was null however the user will receive a notification telling him that there was no ISBN data received.

```
private class GetBookDetails extends AsyncTask<String, Void, String> {

    @Override
    protected void onPreExecute() {
        super.onPreExecute();
        progressDialog = new ProgressDialog(MainActivity.this);
        progressDialog.setMessage("Searching book!");
        progressDialog.setIndeterminate(false);
        progressDialog.setCancelable(true);
        progressDialog.setCanceledOnTouchOutside(false);
        progressDialog.show();
    }
}
```

Fig. 31: Screenshot from the get book details class. Before executing

The on pre execute from the get book details class is performing the same as the pre execute from the login screen. It will create a progress dialog that is not determinable, cannot be cancelled by touching outside of it. The message displayed on it is “searching book!”.

```
@Override
protected String doInBackground(String... bookURLs) {
    StringBuilder bookDetailBuilder = new StringBuilder();
    for (String bookURL : bookURLs) {
        HttpClient bookClient = new DefaultHttpClient();
        try {
            HttpGet bookDetail = new HttpGet(bookURL);
            HttpResponse responseDetail = bookClient.execute(bookDetail);
            StatusLine bookSearchStatus = responseDetail.getStatusLine();

            if (bookSearchStatus.getStatusCode() == 200) {
                HttpEntity bookEntity = responseDetail.getEntity();
                InputStream bookContent = bookEntity.getContent();
                InputStreamReader bookInput = new InputStreamReader(bookContent);
                BufferedReader bookReader = new BufferedReader(bookInput);
                String lineIn;
                while ((lineIn = bookReader.readLine()) != null) {
                    bookDetailBuilder.append(lineIn);
                }
            }
        }
    }
}
```

```

    }
    catch (Exception e) {
        e.printStackTrace();
    }
}
return bookDetailBuilder.toString();
}

```

Fig. 32: Screenshot from the in background method of get book details (13)

The do in background is again where the majority of the code is performed. This is to make sure the heavy and long code is not being executed in the main thread. It uses a local stringbuilder. This is considered a string object but the length can be changed on the fly. The bookUrl is the booksearchstring received from the on activity class. The application will create a new http client. There's also a get method declared named bookdetail. It will integrate the google books URL. This part of the code will search in the Google Books database. This is executed from the httpresponse "responseDetail". From this response detail we will find the status code by using "getStatusLine and save this in a statusline variable. If this booksearchstatus.getStatusCode() equals 200 the details from the book will be retrieved. Here for I will use an httpEntity, an inputstream reader and a buffered reader. The inputstream book content is created by responseDetail.getEntity().getContent(). The inputstreamreader bookinput consists of the inputstreamreader from "bookcontent". I have then created a buffered reader named bookReader. This buffered reader will help with adding the details to the stringbuilder. A local string LineIn will be used for the comparison of the bookReader. As long as the LineIn that is equal to the bookreader.readLine() is not null the line will be added to the string builder. The book details are now all added to the stringBuilder. The stringbuilder will now be returned as a string object. This way the post execute will be able to use it.

```

protected void onPostExecute(String result) {
    progressDialog.dismiss();
    try {
        JSONObject resultObject = new JSONObject(result);
        JSONArray bookArray = resultObject.getJSONArray("items");
        JSONObject bookObject = bookArray.getJSONObject(0);
        JSONObject volumeObject = bookObject.getJSONObject("volumeInfo");
        try {
            titleText.setText("" + volumeObject.getString("title"));
        }
        catch (JSONException jse) {
            titleText.setText("");
            jse.printStackTrace();
        }
        StringBuilder authorBuild = new StringBuilder("");
        try {
            JSONArray authorArray = volumeObject.getJSONArray("authors");
            for (int a = 0; a < authorArray.length(); a++) {
                if (a > 0) authorBuild.append(", ");
                authorBuild.append(authorArray.getString(a));
            }
        }
    }
}

```

```

        authorText.setText("" + authorBuild.toString());
    }
    catch (JSONException jse) {
        authorText.setText("");
        jse.printStackTrace();
    }
    try {
        dateText.setText("" + volumeObject.getString("publishedDate"));
    }
    catch (JSONException jse) {
        dateText.setText("");
        jse.printStackTrace();
    }
}
try {
    descriptionText.setText("" + volumeObject.getString("description"));
}
catch (JSONException jse) {
    descriptionText.setText("");
    jse.printStackTrace();
}
try {
    double decNumberStars = Double.parseDouble(volumeObject.getString("averageRating"));
    int numberStars = (int) decNumberStars;
    starLayout.setTag(numberStars);
    starLayout.removeAllViews();
    for (int s = 0; s < numberStars; s++) {
        starViews[s].setImageResource(R.drawable.star);
        starLayout.addView(starViews[s]);
    }
}
catch (JSONException jse) {
    starLayout.removeAllViews();
    jse.printStackTrace();
}
try {
    LinkButton.setTag(volumeObject.getString("infoLink"));
    LinkButton.setVisibility(View.VISIBLE);
}
catch (JSONException jse) {
    LinkButton.setVisibility(View.GONE);
    jse.printStackTrace();
}
}
try {
    JSONObject imageInfo = volumeObject.getJSONObject("imageLinks");
    new GetBookCover().execute(imageInfo.getString("smallThumbnail"));
}
catch (JSONException jse) {
    coverView.setImageBitmap(null);
    jse.printStackTrace();
}
}
}

```

```

    catch (Exception e) {
        e.printStackTrace();
        Toast toast = Toast.makeText(getApplicationContext(),
            "Book not found! Add the info!", Toast.LENGTH_SHORT);
        toast.show();
        titleText.setText("");
        authorText.setText("");
        dateText.setText("");
        starLayout.removeAllViews();
        rating.setText("");
        coverView.setImageBitmap(null);
    }
}

```

Fig. 33: Screenshot from the post execute of the Get book details class (14)

In this class the book details will be added to the appropriate fields. First the progress dialog is dismissed. The first code being executed is the json object "resultobject" with in here the string received from the in background. Resultobject.getJSONArray("items").getJSONObject(0).getJSONObject("volumeInfo"). All of the retrieved code is then stored in the JsonObject "VolumeObject". The code will now try to retrieve all the right strings, arrays and JsonObject using the Try catch code displayed above. For the title it is a string that is retrieved. For the author(s) however it is an array since there can be more than one author. Since the length is not known beforehand I decided to also implement a stringbuilder for this. The publication year and description are also strings retrieved from the object. Now we have arrived at the average rating count. Here is where the stars will be retrieved and displayed. First I will retrieve the string with in this the average rating and parse this to a double. This double then gets converted to an integer, since I have not included half stars. Then we reach a FOR sequence. This sequence will set the image resource to the star. This star will then be added to the correct starview. The link button will receive a tag containing the link to the Google Books API. The button will also become visible. The link for the book cover is also provided but will be retrieved through another AsyncTask. The "Get Book Cover" class. This class will be explained next.


```

private class GetBookCover extends AsyncTask<String, Void, String> {

    private Bitmap coverImage;

    protected String doInBackground(String... thumbURLs) {
        try {
            URL coverURL = new URL(thumbURLs[0]);
            URLConnection coverConnection = coverURL.openConnection();
            coverConnection.connect();
            InputStream coverInputStream = coverConnection.getInputStream();
            BufferedInputStream coverBuffer = new BufferedInputStream(coverInputStream);
            coverImage = BitmapFactory.decodeStream(coverBuffer);
            coverBuffer.close();
            coverInputStream.close();
        } catch (Exception e) {
            e.printStackTrace();
        }

        return "";
    }

    protected void onPostExecute(String result) { coverView.setImageBitmap(coverImage); }
}

```

Fig. 34: Screenshot from the Get book cover class (15)

The code above is utilised to retrieve the book cover form the URL provided by the post execute from the get book details class. The cover is retrieved through the inputstream as a bitmap. This bitmap has to be decoded from the coverbuffer. The streams are then closed. After the execution of the do in background task, the coverview will be set by the setImageBitmap. Next I will explain the send data to database class.

```

private class SendToDatabase extends AsyncTask<String, Void, String> {

    @Override
    protected void onPreExecute() {
        super.onPreExecute();
        progressDialog2 = new ProgressDialog(MainActivity.this);
        progressDialog2.setMessage("Uploading Book!");
        progressDialog2.setIndeterminate(false);
        progressDialog2.setCancelable(true);
        progressDialog2.setCanceledOnTouchOutside(false);
        progressDialog2.show();
    }
}

```

Fig. 35: Screenshot from the pre execute Send to database class

The pre execute is the same as the previous classes. It will implement a progress dialog with the text “uploading book!”. The rest of the attributes of the progress dialog are the same as the previous dialogs. It is only cancellable by pressing the back button, there’s no progress bar and touching outside of the dialog does not cancel it.

```

@Override
protected String doInBackground(String... args) {
    String authorData = "" + authorText.getText().toString();
    String titleData = "" + titleText.getText().toString();
    String yearData = "" + dateText.getText().toString().split("\\-")[0];
    String isbn = "" + scanContent;
    loginUrl = "http://" + loginUrl.split("/")[2] + "/Library/addtodatabase.php";
    try {
        List<NameValuePair> parameters = new ArrayList<>();
        parameters.add(new BasicNameValuePair("author", authorData));
        parameters.add(new BasicNameValuePair("title", titleData));
        parameters.add(new BasicNameValuePair("year", yearData));
        parameters.add(new BasicNameValuePair("isbn", isbn));
        parameters.add(new BasicNameValuePair("usrnme", username));
        parameters.add(new BasicNameValuePair("docencia", booktype));

        JSONObject json = jsonParser.makeHttpRequest(loginUrl, parameters);
        return json.getString(Message);
    }
    catch (JSONException e) {
        e.printStackTrace();
    }
    return null;
}

```

Fig. 36: Screenshot from the Do in background method from the send to database class

This method will retrieve the text inputted into the title, author and publication year inputs and convert them to strings. The LoginUrl will extract just the IP address of the server and add the right phpscript. A new array list will be created to store the name value pairs. This is the same way as implemented in the login activity. This array list will be sent to the make http request method from the Jsonparser class. It will retrieve a json object from this class. It will retrieve the string “message” attached to the json object. This will be returned to be used in the post execute method.

```

protected void onPostExecute(String Message) {
    if (Message.equals("1")) {
        progressDialog2.dismiss();
        Toast toast = Toast.makeText(getApplicationContext(), "Data uploaded to server!", Toast.LENGTH_SHORT);
        toast.show();
    }
    if (Message.equals("0")) {
        progressDialog2.dismiss();
        Toast toast = Toast.makeText(getApplicationContext(), "Book already in Database!", Toast.LENGTH_SHORT);
        toast.show();
    }
}

```

Fig. 37: Screenshot from the post execute send data to database class

During the post execute the received string “message” will be checked. If this message equals “1” then a notification will appear at the bottom of the screen telling the user that the data has been uploaded to the server successfully. If the message equals “0” the user will receive a notification at the bottom of the screen that the book is already in the database.

```

private class CancelBookOrder extends AsyncTask<String, Void, String> {

    @Override
    protected void onPreExecute() {
        super.onPreExecute();
        progressDialog3 = new ProgressDialog(MainActivity.this);
        progressDialog3.setMessage("Searching and cancelling book!");
        progressDialog3.setIndeterminate(false);
        progressDialog3.setCancelable(true);
        progressDialog3.setCanceledOnTouchOutside(false);
        progressDialog3.show();
    }
}

```

Fig. 38: Screenshot from the pre execute from the cancel book order class

This pre execute is the same as the pre execute from the Send to database class. Therefore I will not explain this part.

```

@Override
protected String doInBackground(String... args) {
    String authorData = "" + authorText.getText().toString();
    String titleData = "" + titleText.getText().toString();
    loginUrl = "http://" + loginUrl.split("/")[2] + "/library/cancelorder.php";
    try {
        List<NameValuePair> parameters = new ArrayList<>();
        parameters.add(new BasicNameValuePair("author", authorData));
        parameters.add(new BasicNameValuePair("title", titleData));
        parameters.add(new BasicNameValuePair("usrnme", username));

        JSONObject json = jsonParser.makeHttpRequest(loginUrl, parameters);
        return json.getString("Message");
    }
    catch (JSONException e) {
        e.printStackTrace();
    }
    return null;
}
}

```

Fig. 39: Screenshot from the background code from cancel book order

The do in background performs the same task as the one from the send to database class therefore I will not explain this part.

```

protected void onPostExecute(String Message) {
    if (Message.equals("1")) {
        progressDialog3.dismiss();
        Toast toast = Toast.makeText(getApplicationContext(), "Book request has been cancelled", Toast.LENGTH_SHORT);
        toast.show();
    }
    if (Message.equals("0")) {
        progressDialog3.dismiss();
        Toast toast = Toast.makeText(getApplicationContext(), "This book hasn't been requested by your account",
            Toast.LENGTH_SHORT);
        toast.show();
    }
}
}
}

```

Fig. 40: Screenshot from the post execution from the cancel book order class

In the post execute the message will also be checked. If it equals "1" the user will receive a notification that the book request has been successfully cancelled. If the message equals "0" the user will receive a notification that he did not request the book.

3.4.3 Json parser

```
import android.util.Log;

import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.NameValuePair;
import org.apache.http.client.entity.UrlEncodedFormEntity;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.impl.client.DefaultHttpClient;
import org.json.JSONException;
import org.json.JSONObject;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.List;
```

Fig. 41: Screenshot from the imports used by the Json Parser class

The imports used by the Json parser are declared in this part. Since the json parser is used to connect to the server and send data to the database it will implement a lot of apache imports. These imports are the ones used for making the connection and maintaining it. The last part of the imports are used for retrieving the information from the database and sending the responses back to the appropriate activities that have called this class.

```
public JSONObject makeHttpRequest(String url, List<NameValuePair> parameters) {
    try {
        DefaultHttpClient ConnectionClient = new DefaultHttpClient();
        HttpPost post = new HttpPost(url);
        post.setEntity(new UrlEncodedFormEntity(parameters));
        HttpResponse ConnectionResponse = ConnectionClient.execute(post);
        HttpEntity ConnectionEntity = ConnectionResponse.getEntity();
        is = ConnectionEntity.getContent();
    }
    catch (IOException e) {
        e.printStackTrace();
    }

    try {
        BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(is, "iso-8859-1"), 8);
        StringBuilder stringBuilder = new StringBuilder();
        String stringHelper = null;
        while ((stringHelper = bufferedReader.readLine()) != null) {
            stringBuilder.append(stringHelper).append("\n");
        }
        is.close();
        json = stringBuilder.toString();
    }
}
```

```

        catch (JSONException e) {
            e.printStackTrace();
        }

        return jsonObject;
    }
}

catch (Exception e) {
    e.printStackTrace();
}

try {
    jsonObject = new JSONObject(json);
}

```

Fig. 42 : Screenshot from the code used to send added parameters to the database(16)

The json parser class is used to make a connection to the database using the http post protocol. This is to encode the details sent. A user capturing the details would not be able to retrieve the details. The json parser class receives the Url and the array list from the activity calling it. The http Entity is again achieved by a series of actions. First it will be set by encoding the parameters and this will be sent to the database. The response will be handled the following way: `Connectionclient.execute(post).getEntity()`. An inputstream will then be used to capture the response. It will get the content from the connection entity.

After having the inputstream a buffered reader will read this inputstream. For this a string named string helper and a stringbuilder named stringbuilder will help it. The action will be almost the same as when recovering the book details. The stringhelper will be equal to the `bufferedReader.readLine()`. As long as this isn't null the stringbuilder will append the string read into the stringhelper. When the stringhelper is finally null, the inputstream will close and the json object will receive the value of the stringbuilder converted to a string. This will be sent back to the activity calling the json parser class.

4 User and system requirements

During this chapter I will explain the necessary factors for both users and smartphones to be able to use the Library scanner application. I will first cover the requirements for users and then for the smartphones.

4.1 User requirements

The application was designed to be used in a professional way (by a company, university, high school...). This also means that the company or school has to have a database of all the users they want using the application. In this database they have to manually enter the users. It would be illogical to create a way to register users from the application, since only employees or authorized persons can make the book requests. The protection is hereby improved.

4.1.1 Login credentials

The server manager will add the employees who are approved to use the application into the “personal” table. Here they will be assigned a username and password along with additional information that can be provided. The user will need the username and password for logging in to the server. Without these credentials he will not be able to initialise the application.

4.1.2 Internet connection

To use the application the user needs to be able to connect to the Internet. This can be done either by mobile data connection or a Wi-Fi connection. Because of this the application can be used almost everywhere in the world. The mobile data connections have significantly improved over the last decade and nowadays you can connect almost everywhere to a (free) Wi-Fi network.

4.2 System requirements

The following requirements are mandatory for a good experience of the application. If these requirements are not met the application might feel a bit slow and tasks will take very long to complete. The application has also been tested on devices running close to the minimum specifications.

4.2.1 Minimum required specifications

Keep in mind that these are the absolute minimum requirements for having a pleasant experience with the application.

The minimum processor speed should be at least 1 GHz. Most of the smartphones that have been produced since 2013 will have at least a Dual core 1 GHz processor. This has become the base for a smooth Android experience.

The minimum graphics processing unit should be an Adreno 200 integrated GPU. This GPU will be able to render the application. The unit is clocked at 200 MHz

The minimum RAM requirement is 512MB. Since the introduction of the HTC Desire back in February 2010, 512MB of RAM was only for the top-notch phones. Since 2013 512MB has become the Android based smartphone's base.

The smartphone needs to be able to connect to a mobile data or Wi-Fi network. A 3G connection or higher is recommended for mobile data. For Wi-Fi the connection speed needs to be at least 54Mbit/sec. This is a G connection. All smartphones since 2013 include a chip that is capable of connecting to B/G/N networks.

The smartphone needs to have a camera. The minimum resolution of this camera needs to be 3 Megapixel. This will make sure the barcode is easily focussed. The better the quality of the camera, the faster the application will detect the barcode.

The smartphone needs to run Android 4.0 (Ice Cream Sandwich) or a custom Rom that is based on Android 4.0 or higher. This can be, for example, Cyanogenmod CM9. This Custom Rom is based on the Android 4.0.4 Base.

4.2.2 Recommended specifications

The following specifications are the ones from the phone that was used to test the application. This was a high-end phone so there was no lag on the application whatsoever.

The recommended processor is a quad core processor clocked at 2.5GHz type snapdragon.

The recommended GPU is an Adreno 330 clocked at 578MHz.

The recommended RAM is 3GB.

The recommended mobile data connection speed is 4G (LTE).

The recommended camera resolution is 13 Megapixel.

The recommended android version for this application is Android 5.0 (Lollipop). The smartphone used for testing was running Cyanogenmod CM12, which is a custom Rom based on Android 5.0.1.

These are the specifications from the smartphone used for testing. With these specifications there were no speed issues, rendering issues... Therefore I recommend these specifications for the application usage. The application has been extensively tested with these specifications.

The application has also been tested on a lower-end smart phone. It worked well with only a little lag from time to time, mainly when opening a new intent.

4.2.3 Server requirements

The server does not have any specified hardware requirements. As long as you can run a web server capable of implementing a MySQL database.

#	Naam	Type	Collatie	Attributen	Leeg	Standaardwaarde	Extra
1	id 🔑	int(11)			Nee	<i>Geen</i>	AUTO_INCREMENT
2	username	char(20)	utf8_general_ci		Ja	<i>NULL</i>	
3	fecha_pet	datetime			Ja	<i>NULL</i>	
4	docencia	int(1)			Ja	<i>NULL</i>	
5	autor	varchar(100)	utf8_general_ci		Ja	<i>NULL</i>	
6	titulo	varchar(100)	utf8_general_ci		Ja	<i>NULL</i>	
7	anno	int(11)			Ja	<i>NULL</i>	
8	isbn	varchar(50)	utf8_general_ci		Ja	<i>NULL</i>	
9	estado	int(11)			Ja	<i>NULL</i>	

Fig. 43: Screenshot from the fields of the libros_total table

In the picture above you can see the fields necessary in the libros_total table. In the following screenshot you will see the fields necessary in the personel table.

#	Naam	Type	Collatie	Attributen	Leeg	Standaardwaarde	Extra
1	ID	int(11)			Ja	<i>NULL</i>	
2	Alias	varchar(30)	utf8_general_ci		Ja	<i>NULL</i>	
3	username	char(20)	utf8_general_ci		Ja	<i>NULL</i>	
4	clave	varchar(50)	utf8_general_ci		Ja	<i>NULL</i>	
5	Nombre	varchar(50)	utf8_general_ci		Ja	<i>NULL</i>	
6	Apellidos	varchar(50)	utf8_general_ci		Ja	<i>NULL</i>	
7	ip	varchar(50)	utf8_general_ci		Ja	<i>NULL</i>	
8	hostname	varchar(50)	utf8_general_ci		Ja	<i>NULL</i>	
9	CuentaActiva	bit(1)			Ja	<i>NULL</i>	
10	Campus	varchar(255)	utf8_general_ci		Ja	<i>NULL</i>	

Fig. 44.: Screenshot from the fields in the personel table

5 Planning

For the execution of this final project and the writing of the thesis I had a timespan of four months. I started working on the project in February and had to present my work the 27th of May. During these four months I had to prepare and do research, develop the application and prepare the defence.

5.1 February

During this month I have been mainly busy with research. I was looking through online websites to refresh my base of android programming. I started by trying some basic programs again before starting with this project. I started creating a basic design featuring just the scan button and 3 input fields.

Afterwards I was trying to create a barcode scanner but an article caught my eye (XZing). (3) It was a way to implement an existing barcode scanner into the application. After reading this article I tried implementing it into my application. By the end of the month I was able to scan an ISBN barcode and retrieve the barcode itself to the application.

5.2 March

I started off March by trying to make the application connect with the Google Books database. I found a lot of documentation on the Google Books API page. (4) After finding the way to connect to the Google Books API I had to alter the application to retrieve the information from the Json string received.

I had some trouble to find the proper way to display the received information before giving the user the possibility to send it to the database. When I got more comfortable using the Google books API I also integrated a link button that opened a web view with a link to the Google Books page of the book.

By the end of the month I was also asked to implement a log in for the users. I started working on this in a separate project to try it out. (5) I tried to create a login system that was both simple and fast. Finally I managed to get the log in to work in the separate project. After completing this it was now time for placing the log in into the application. After integrating the log in I was confronted with multiple system errors. It took some time to sort through all these errors and resolve them.

5.3 April

After resolving all the errors that occurred with the integration of the log in screen I was now able to begin working on the connection from the application to the database.

First I had to create a local server. I tried placing the database on a webserver but quickly abandoned the idea after having problems with three online servers. Every time I tried uploading the database it stopped after a certain number of records, which caused the database to be incomplete. Because of this it was also impossible to use this solutions.

I installed Xampp 5.6.8 on my laptop and configured it to hold a static IP address. Now that I had the server running and had the ability to manage the database using the integrated PHPMyAdmin I was able to start working on the PHP scripts. I had already written the checklogin.php script while working on the login. Based on this file I started writing the addtodatabase.php script. The connecting code was exactly the same so this saved me some time. (6)

After creating and testing this script I started working on a script to cancel the books. First I implemented the cancel button into the application. It connects to the server the same way the send data button does. It also provides the variables the same way. I decided that for the storage and cancelling of books it would be advantageous to also be able to search for the user posting it.

I had to find a way to pass data from one intent to another.(7) Because of this documentation that was very easy to integrate into the application. I made sure the username is getting added as an extra from the login activity to the library scanner. The username is then getting stored in a variable.

After a meeting with the responsible professor I was asked to implement a way to change the server address without having to rebuild the whole application. I added this on the mean screen in the form of another input with a button to confirm the change. He also asked me to implement a way to oblige the user to select the reason he is going to use the book after clicking the send data button. I decided to implement an alert dialog. (8)This dialog gave the user 2 options to choose from but the obligation to select one.

After implementing this I started resolving some of the errors and warnings I was getting. I let the code inspector from android studio run through my code to find errors against the android convention. I got a warning that I had put a time consuming task in

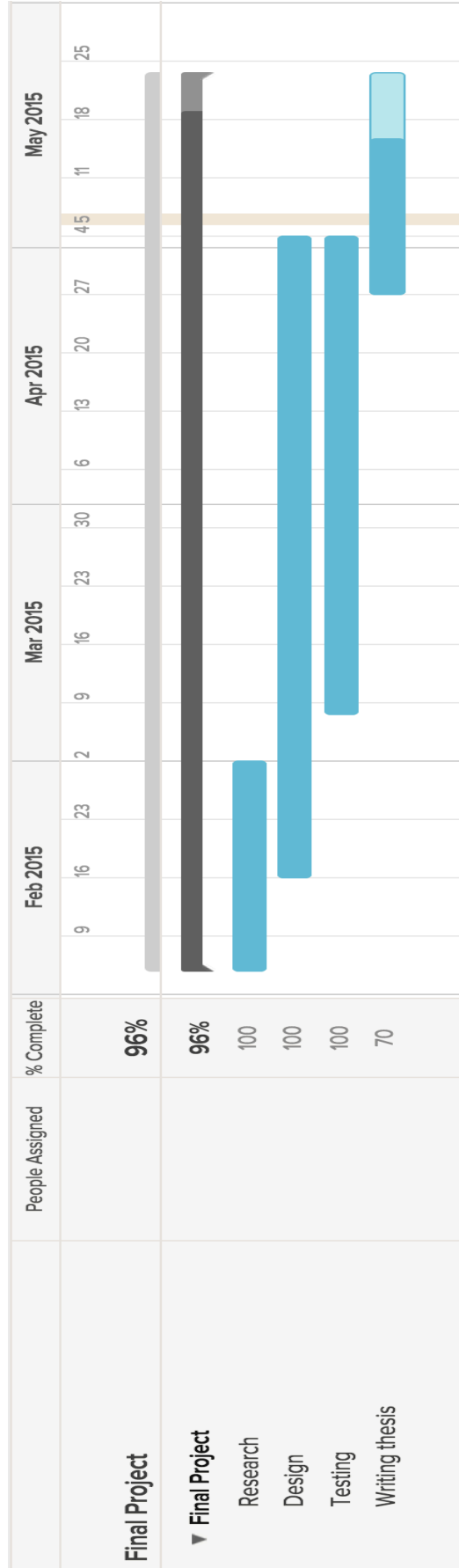
the main thread and this could cause problems. I then searched for a way to be able to perform the tasks in the background.(9)

The implementation of the asyncactivity was rather fast. Unfortunately I encountered more errors. It took me a while to resolve these but by the end of the month I was ready with the implementation of this activity. I made an asyncactivity for retrieving the book details, sending the data, logging in and cancelling the book.

5.4 May

In May I continued the work from the past months. I decided to add in a progress dialog when the Asyncactivities are working. I first encountered problems with the dismiss time from the dialog. It kept on running. I added in the on post execute method. After placing the dismiss inside of this method it worked like a charm.

During this month I also started writing this thesis and preparing my defence. I started creating the PowerPoint with which I will be doing my defence and writing the text I will use on my defence. I also started preparing for possible questions asked to me during my defence.



6 Software tests performed

To test the application I created, I had to set up a server. I decided to use Xampp version 5.6.8. On this server there was a database manager included. I used the MyPHPAdmin to manage the database. I uploaded the database that was provided to me and placed the php scripts in the appropriate place. I gave the computer a fixed IP address, that way I was sure where the application had to connect to.

6.1 Test of the book retrieval

First I tested the book detail retrieval. For this test I didn't need the server running. The only thing needed was an Internet connection. After implementing the login into the code the server had to be running as well for the test. I tested this with five different books by scanning the ISBN code and looking of the book was found in the Google Books database. From the five different books, the application was able to retrieve the details from four. This is an 80% success rate. The book that was not found was a translation. When I searched the book manually I noticed that the ISBN code was different from the one on the book cover itself.

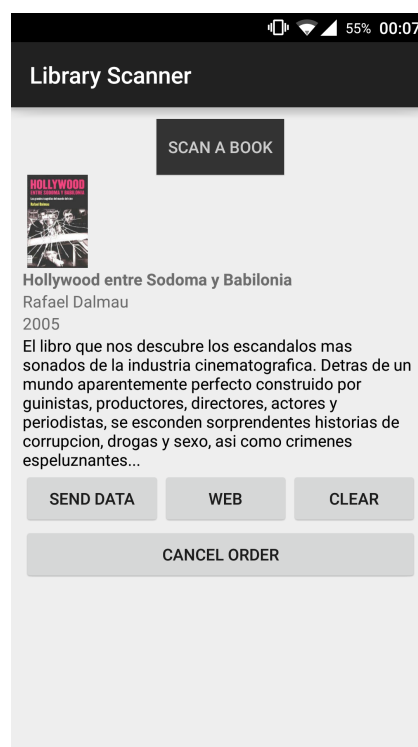
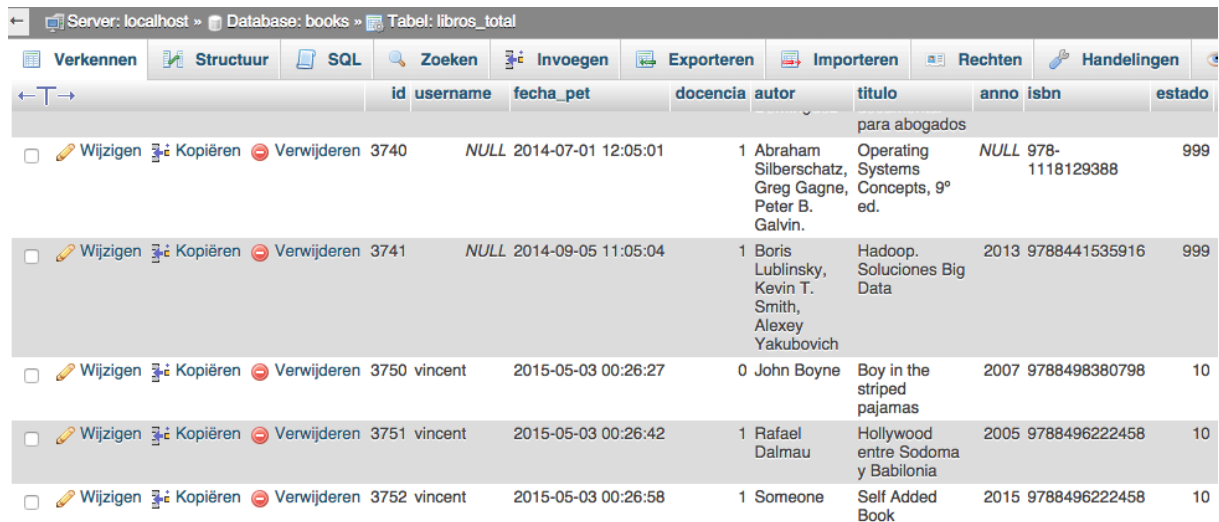


Fig. 45: Example of retrieved book information

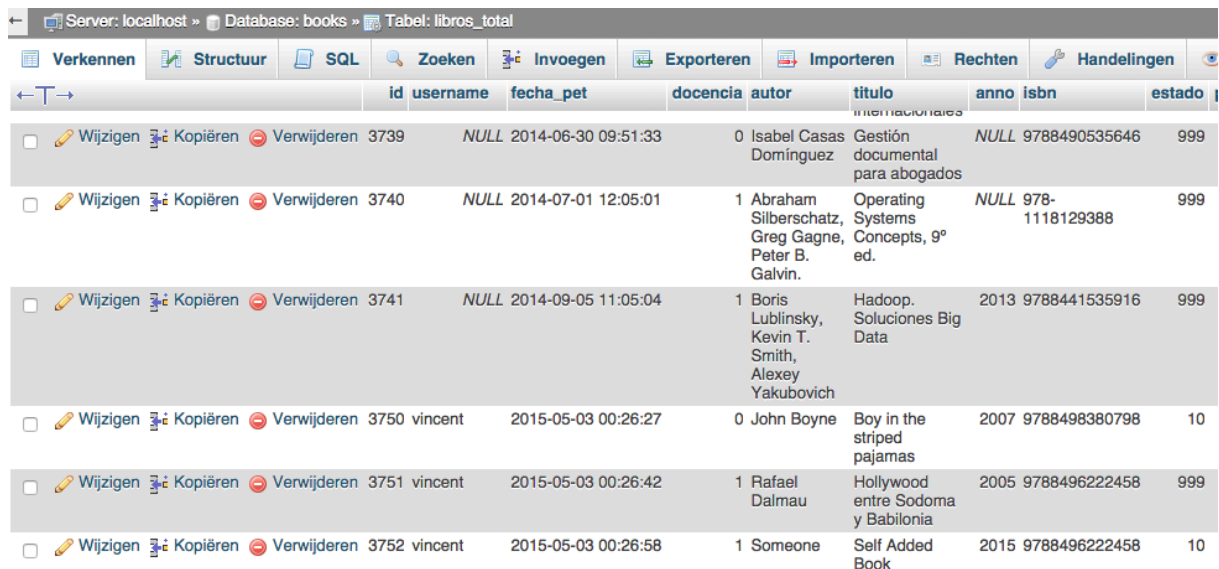
6.2 Test of the connection and transmission of data to the server

Like I said I used the PHPMyAdmin for the management of the database. I uploaded the details from two of the scanned books and added one manually. Then I also tested the cancel method. Below is a screenshot from the server with data that has been added and later cancelled.



	id	username	fecha_pet	docencia	autor	titulo	anno	isbn	estado
<input type="checkbox"/>	3740	NULL	2014-07-01 12:05:01	1	Abraham Silberschatz, Greg Gagne, Peter B. Galvin.	Operating Systems Concepts, 9 ^o ed.	NULL	978-1118129388	999
<input type="checkbox"/>	3741	NULL	2014-09-05 11:05:04	1	Boris Lublinsky, Kevin T. Smith, Alexey Yakubovich	Hadoop. Soluciones Big Data	2013	9788441535916	999
<input type="checkbox"/>	3750	vincent	2015-05-03 00:26:27	0	John Boyne	Boy in the striped pajamas	2007	9788498380798	10
<input type="checkbox"/>	3751	vincent	2015-05-03 00:26:42	1	Rafael Dalmau	Hollywood entre Sodoma y Babilonia	2005	9788496222458	10
<input type="checkbox"/>	3752	vincent	2015-05-03 00:26:58	1	Someone	Self Added Book	2015	9788496222458	10

Fig. 46: Screenshot from the database with 3 books entered (2 automatic one manually)



	id	username	fecha_pet	docencia	autor	titulo	anno	isbn	estado
<input type="checkbox"/>	3739	NULL	2014-06-30 09:51:33	0	Isabel Casas Dominguez	Gestión documental para abogados internacionales	NULL	9788490535646	999
<input type="checkbox"/>	3740	NULL	2014-07-01 12:05:01	1	Abraham Silberschatz, Greg Gagne, Peter B. Galvin.	Operating Systems Concepts, 9 ^o ed.	NULL	978-1118129388	999
<input type="checkbox"/>	3741	NULL	2014-09-05 11:05:04	1	Boris Lublinsky, Kevin T. Smith, Alexey Yakubovich	Hadoop. Soluciones Big Data	2013	9788441535916	999
<input type="checkbox"/>	3750	vincent	2015-05-03 00:26:27	0	John Boyne	Boy in the striped pajamas	2007	9788498380798	10
<input type="checkbox"/>	3751	vincent	2015-05-03 00:26:42	1	Rafael Dalmau	Hollywood entre Sodoma y Babilonia	2005	9788496222458	999
<input type="checkbox"/>	3752	vincent	2015-05-03 00:26:58	1	Someone	Self Added Book	2015	9788496222458	10

Fig. 47: Screenshot of the server with the middle book cancelled. (done by rescanning the book and sending cancel request)

The tests above show that the application works. After scanning a book I have also tested the link button. For an illustration of this test you can look in the user manual.

7 User manual

During this chapter I will give detailed instructions on how to use the application and all of its functions. During each step I will explain all the different possibilities of the application.

7.1 Open the application launcher and select the Library Scanner app

On the smartphone select the application launcher. Search for the library scanner application and click on it. The application will now open and show the login screen. Here you can see three input fields: “username”, “password” and “IP address”. You can also see two buttons: “Log in” and “Change server IP”.

7.2 Login using the credentials given to you by the system administrator

On the login screen enter the right username (given by the system administrator) and matching password. Click on the login button. The application will then try and make a connection with the server declared at the bottom of the page. There are multiple protections build into this app.

7.2.1 Login server not found

After pressing the login button the application will try and make a connection with the server declared at the bottom. However when this server is not encountered or a connection cannot be made a warning message will appear at the bottom of the app. The warning message will tell you that the server has not been found.

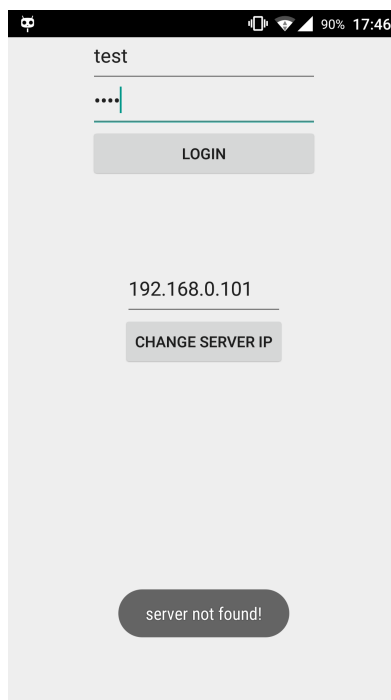


Fig. 48: Screenshot server not found

7.2.2 Change the log in server

When the login server cannot be found, you have to edit the server IP address in the app. In the input field at the middle of the screen you can see an IP address. Change this IP address to the server's IP address. Then click on the change server IP button to change it. After clicking the button a notification will appear telling you that the server's IP has been changed. The input field will make the app use the following IP address: "http://" + the text you've entered + "/library/checklogin.php".

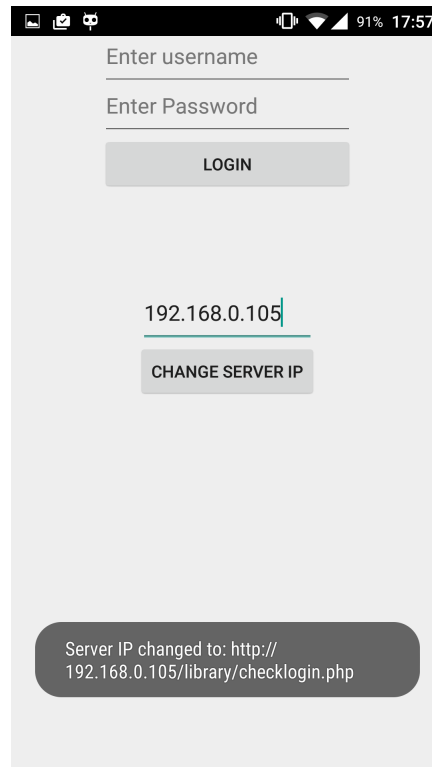


Fig. 49: Screenshot changing IP address

7.2.3 Logging in with username, password or both blank

When trying to login with one or both of the inputs empty a warning message will appear at the bottom of the app. This message will tell you to fill in or the username field, the password field or both. After filling in the missing credential(s) you can log in normally.

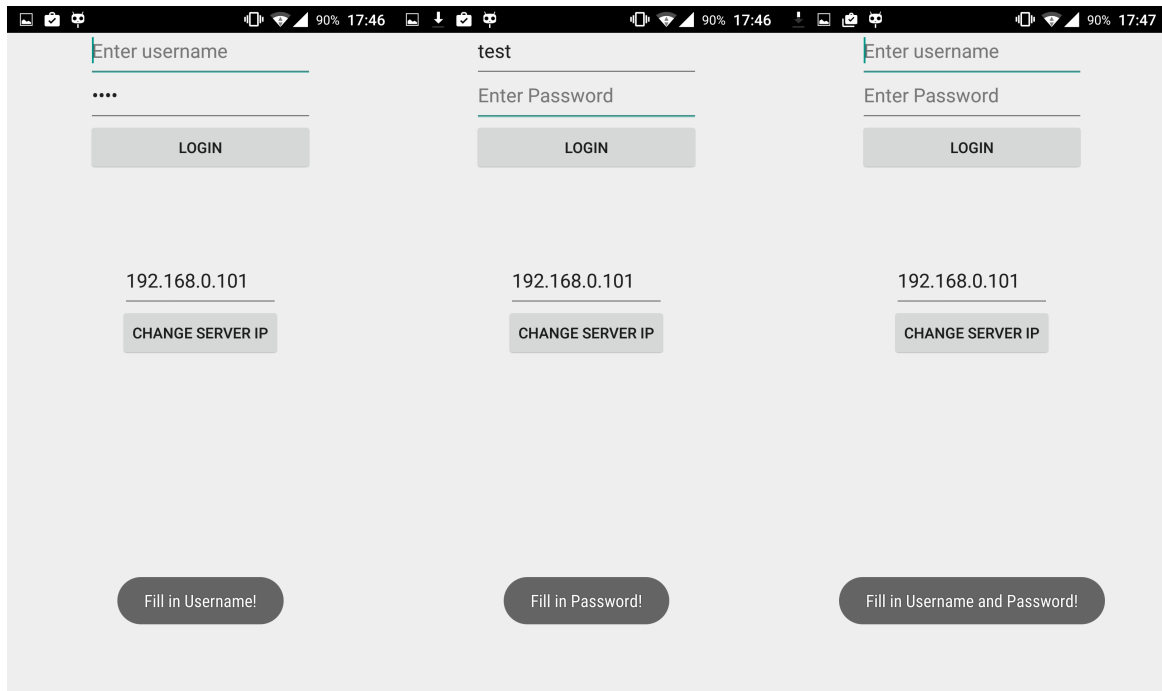


Fig. 50, Fig. 51 and Fig. 52: Screenshots from the Username and Password protection

7.2.4 Logging in with wrong credentials

The application will check the inputted username and password with the usernames and matching passwords in the database. When you've entered the wrong credentials a message will appear at the bottom of the app. This warning message will tell you the username or password is invalid. After correcting the credentials, the log in will proceed normally.

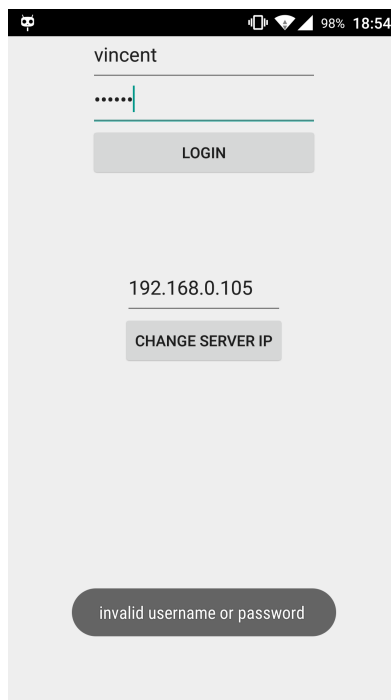


Fig. 53: Screenshot invalid Username or Password

After logging in with the right credentials the application will now show the main page.

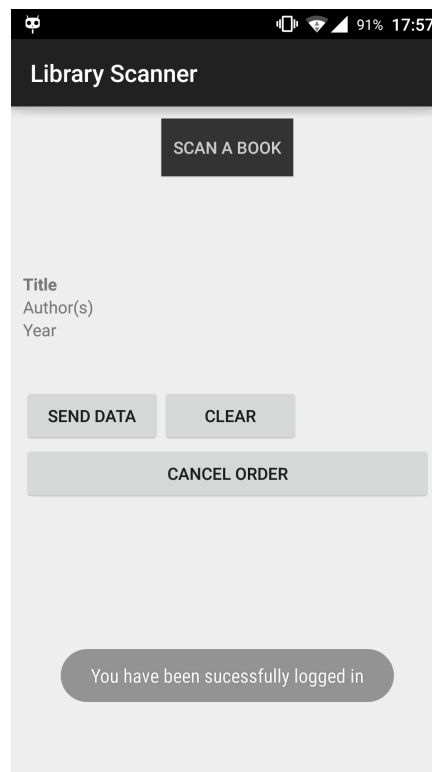


Fig. 54: Screenshot successful log in

7.3 Main application page

Here you can see 3 input fields: "Title", "Author(s)" and "Year". There are also four buttons visible: "Scan Book", "Send Data", "Clear" and "Cancel Order" and one hidden button: "link".

7.3.1 Retrieve book information online

After pressing the scan button the application searches for the application "Barcode Scanner" by ZXing. If this application is encountered on the phone the barcode scanner will launch. If the application is not encountered the library scanner application will open a dialog to inform you the barcode scanner has to be downloaded. From this dialog you will be linked to the Google Play Store page of the Barcode Scanner app. On this page click install. After installation of the barcode scanner application you can return to the book app using the recent app functionality of the smartphone. This can be done by long pressing (2 seconds) the menu button. The recent applications will pop up. Select the library scanner from this list and click the scan book button again.

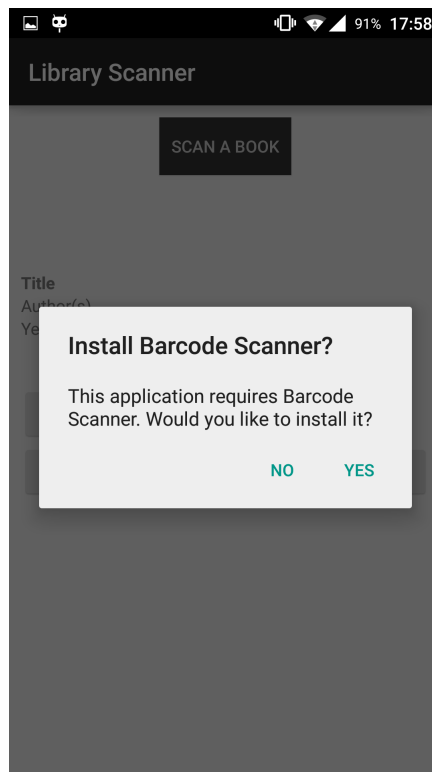


Fig. 55: Screenshot install dialog when application is not found

After the launch of the barcode scanner search for the ISBN code on the book. This barcode can usually be found on the back of the book. Hold the barcode in the designated square of the barcode scanner and wait for the camera to focus. After focussing the application will make a sound and show in the bottom it had encountered the barcode.

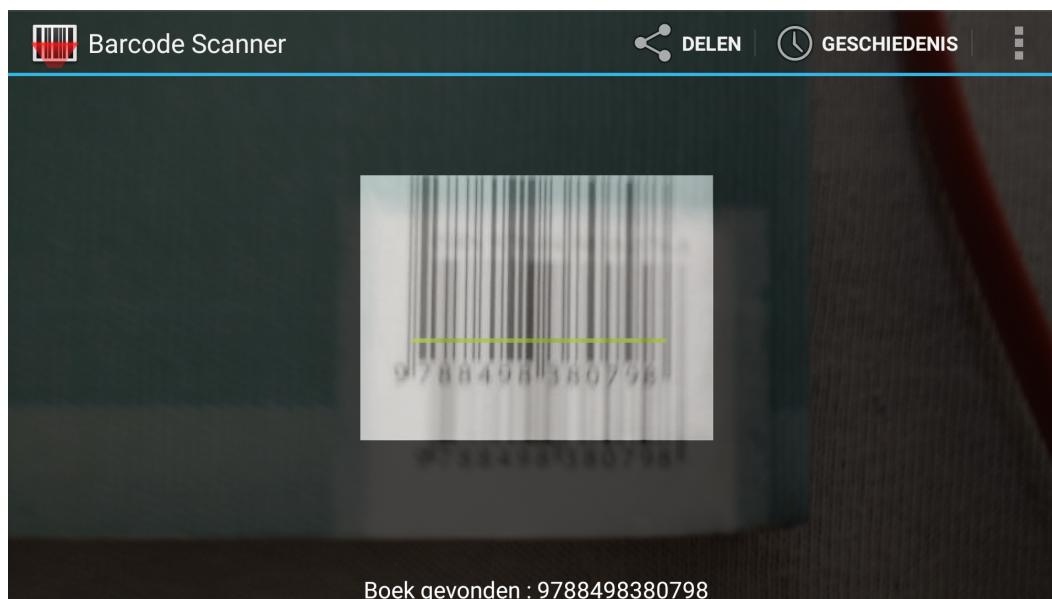


Fig. 56: Screenshot barcode scanner

After encountering the barcode you will be redirected to the main page of the library scanner application. Wait for some seconds while the application searches on Google books for the information. There are 2 possible outcomes. You receive a message at the bottom of the application mentioning: “Book not found! Please add the information!” or the book information will be displayed. You will be able to see a thumbnail picture of the book cover, the title, the author(s), the publication year, the book description and the rating of the book (when available). If the book is found online the hidden button “link” will also become visible.

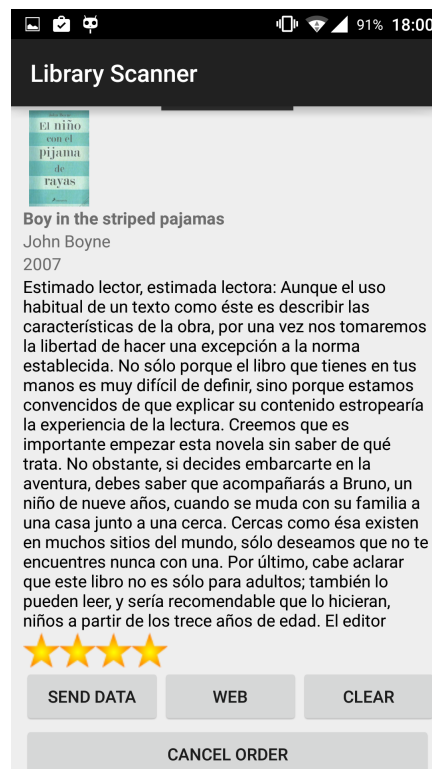


Fig. 57: Screenshot retrieved book details

7.3.2 Enter the data manually

When the book information can't be found online you have to enter it manually. Enter the book title, author and publication year in the corresponding fields. The book title and author fields automatically write in camel case. For sending the book request to the database all three fields must be filled in. For sending a book cancel request only title and author need to be filled in.

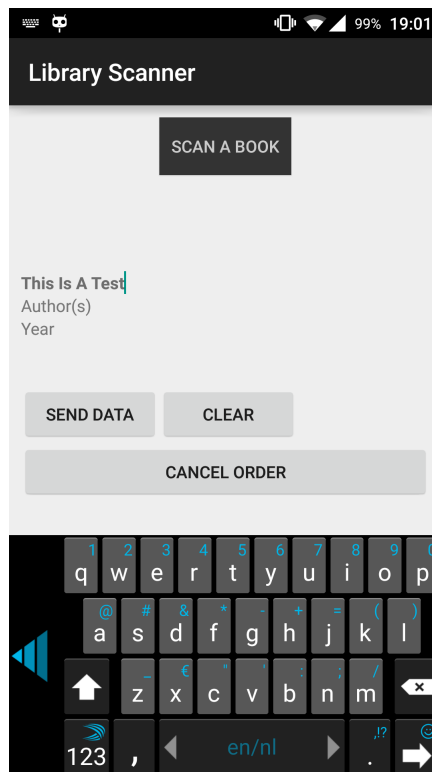


Fig. 58: Screenshot from data being entered manually

7.3.3 Send data to database

When the send data button is pressed the application will check if the title, author(s) and publication year inputs are filled in.

If all fields are empty the application will show a warning message at the bottom of the screen mentioning: “Add the title, the author(s) and the publication year!”.

If the title and author fields are empty the application will show a warning message at the bottom of the screen mentioning: “Add the title and the author(s)!”.

If the author and publication year fields are empty the application will show a warning message at the bottom of the screen mentioning: “Add the author(s) and the publication year!”.

If the title and publication year fields are empty the application will show a warning message at the bottom of the screen mentioning: “Add the title and the publication year!”.

If the title field is empty the application will show a warning message at the bottom of the screen mentioning: “Add the title!”.

If the author field is empty the application will show a warning message at the bottom of the screen mentioning: “Add the author(s)!”.

If the publication year field is empty the application will show a warning message at the bottom of the screen mentioning: “Add the publication year!”.

If all fields are filled in correctly the application will open a dialog screen asking you the purpose of the book. After selecting the purpose the application will contact the server and send the book details to it. The book details are now stored inside the server. The responsible person can now execute the book request.

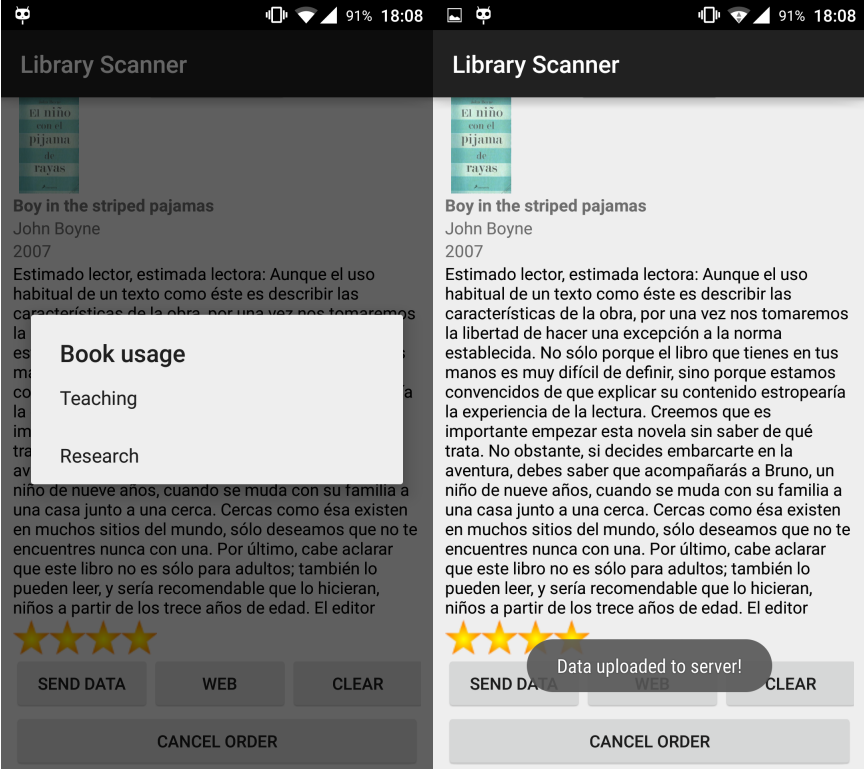


Fig. 59: Screenshot dialog screen with options

Fig. 60: Screenshot Data uploaded to server

7.3.4 Link to the found book

After clicking on the link button a new screen will be opened. This is a web view that will directly link to the book's page on Google Books. On this page you can find some extra details. The page shows extra details about the author(s), the page count... and in the left corner links to online and physical stores where you can purchase the book. By clicking one of the links on the left you can find the price and availability of the book.

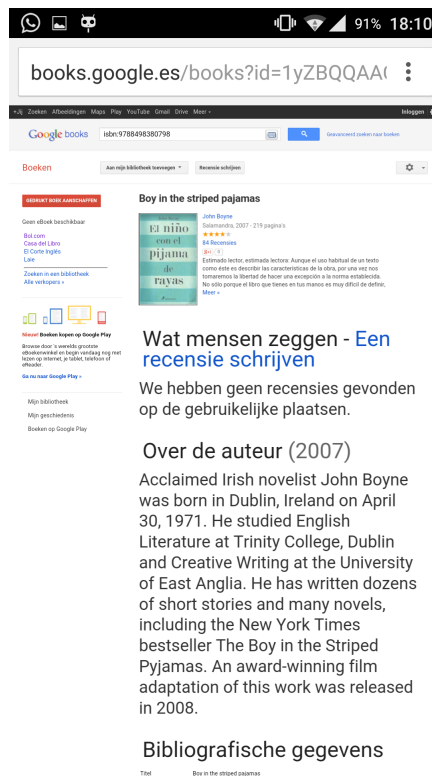


Fig. 61: Screenshot link to book through webview

7.3.5 Clear the retrieved or manually entered details

If you want to clear all the information that is shown on the screen (including the cover thumbnail, the title, the author(s), the publication year, the rating) press the clear button. It will reset all the input fields back to their original state. The ratings will also be removed and the link button will go back to its invisible state.

7.3.6 Cancel one of the book(s) you've requested

If you want to cancel one (of the) book(s) you have requested, this can be done fairly easily. If you have the book nearby you can scan in the ISBN barcode again and retrieve the book details. After the book details have been retrieved again simply click the cancel order button. If you don't have the book nearby you have to fill in the title and the author(s).

There is a protection when clicking the cancel button. After clicking the cancel button the application first checks if the correct fields are filled in.

If the title field is empty the application will show a warning message at the bottom of the screen mentioning: "Add the title!".

If the author field is empty the application will show a warning message at the bottom of the screen mentioning: "Add the author(s)!".

If all the fields are correctly filled in, the application makes a connection with the server and searches for the book and checks if the user who is asking to cancel it has requested the book. If the book is encountered and the username is from the person who wants to

cancel it, the table is altered and the status is changed to "cancelled" (status 999). You get a notification that the book request has been successfully cancelled. If the book is not encountered or the username is not from the person who is trying to cancel it, you will get a notification at the bottom of the screen saying the book hasn't been requested by this account.

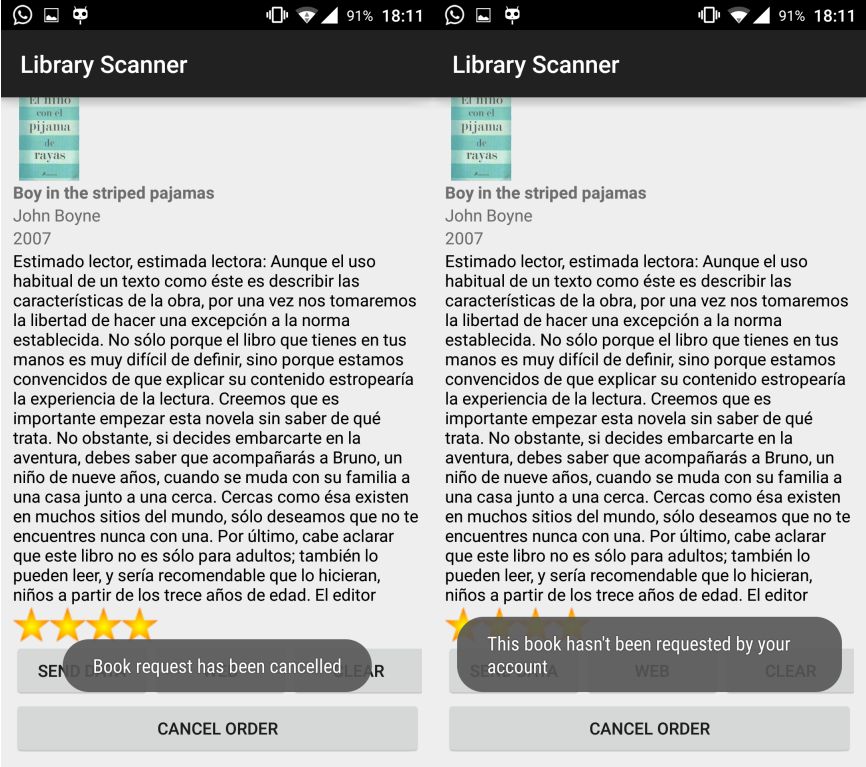


Fig. 62: Screenshot book request cancelled successfully

Fig. 63: Screenshot book not found or not the right user to cancel

7.4 Closing the application

When you're finished with the application you can simply close it by pressing on the back button until you are back in the application launcher. The other possibility is to press on the home button to go back to your home screen. When you start the application again you will now have to log in again.

8 Installation manual

In this chapter I will explain how to install the application on the smartphone. It will be in a step-by-step form. This progress is very easy since the smartphone does most of the work. When the application is downloaded directly to the smartphone from the Internet you may go directly to step 8.3.

8.1 Download the LibraryScanner.apk to your computer

To download the libraryscanner.apk you need to be able to access the private server on which it is stored. You can also have the libraryscanner.apk sent to you by someone who already downloaded it.

8.2 Transfer the LibraryScanner.apk to your mobile phone

This step will be split for a Windows and a Mac computer. First connect the mobile phone through USB.

8.2.1 Windows computer

On a windows computer this can be done by first locating the LibraryScanner.apk package on the computer. Afterwards you can simply drag the installation package to the mobile phone, which will be used as an external memory stick. Make sure you have placed it somewhere that is easily locatable on the smartphone's internal memory.

8.2.2 Mac computer

On a mac running OSX this progress is a little bit different. First you will have to download an application called "Android file transfer". This can be downloaded from the official Android website from google. After downloading and installing this application connect your phone to the mac. Drag the installation package to the internal memory of the phone and place it somewhere you can find it easily

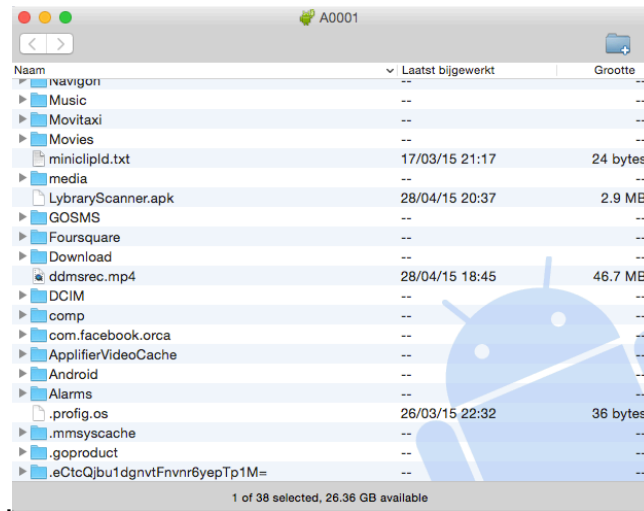


Fig. 64: Android File Transfer

8.3 Enable installation from unknown sources

This step will be split up for Android 5.0 and newer versions and Android 4.X versions.

8.3.1 Android 5.0 (Lollipop) and newer versions

Go to the settings menu from the smartphone. This can be done either by dragging down on the notification bar twice (on Android 5.0 Lollipop and newer versions) then pushing the gear wheel.

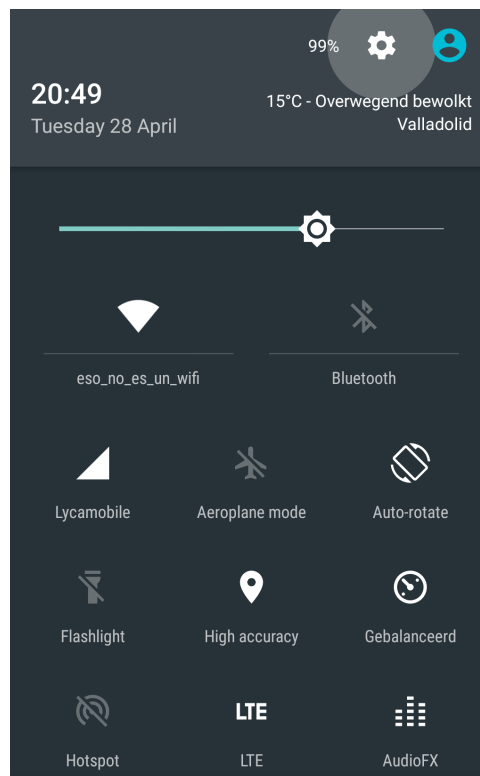


Fig. 65: Selection from the settings menu on Android 5.0.2

8.3.2 Android 4.X versions (Ice Cream Sandwich, Jelly Bean and Kit Kat)

On the older versions of android this function was not yet implemented. To go to the settings simply open up your application launcher and find the application named settings.

Now that we have entered the settings menu search for the “security” tab. Under device administration you should see the option “unknown sources – Allow installation of apps from sources other than the Google Play Store”. Activate this option either by checking the checkbox or by toggling the switch.

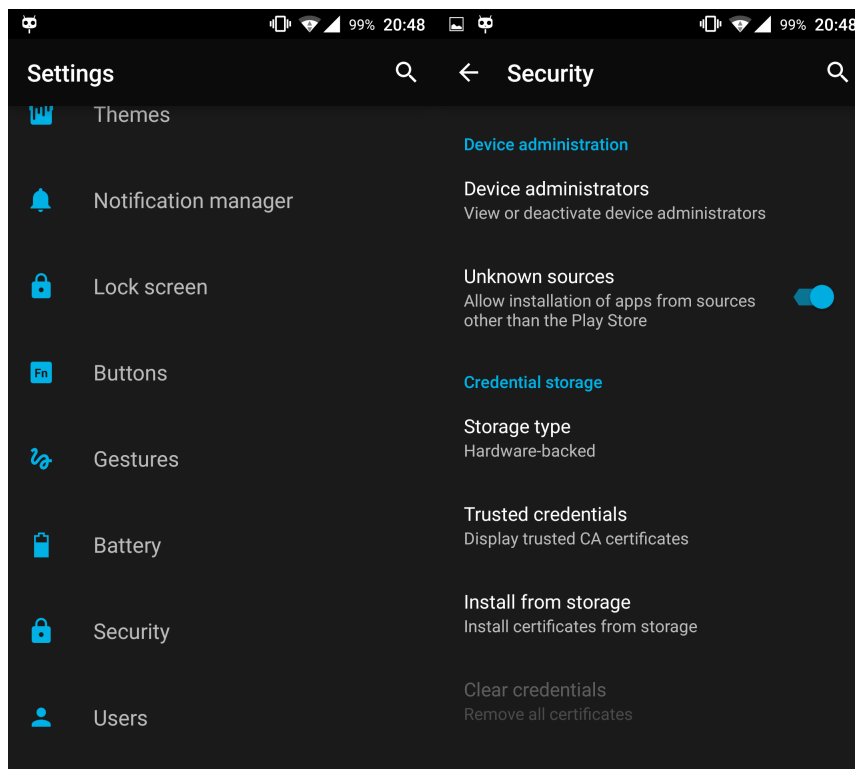


Fig. 66: Location of the security tab on Android 5.0.2

Fig. 67: Unknown sources switch on Android 5.0.2

8.4 Locate the installation package on the phone and execute it

Using a file explorer locate the installation package (LibraryScanner.apk) on the internal memory of the mobile phone. This is an application standard in Cyanogenmod, however when running a stock Android version you will have to download one from the Google Play Store (for example file explorer) in order to be able to search for the application. There are a lot of applications for exploring the internal memory. Choose the application that suits you best.

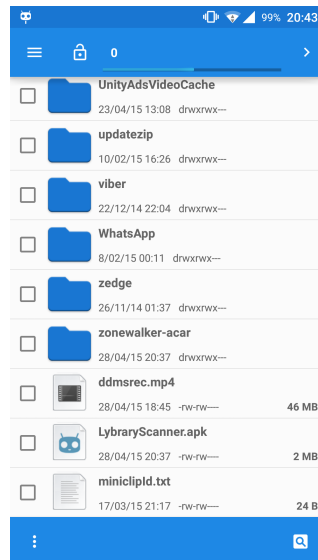


Fig. 68: File explorer on android 5.0.2

After locating the installation package on the phone click on it. This will open an installation dialog asking you to accept the permissions used by this application. Click on accept and wait for the application to be installed.

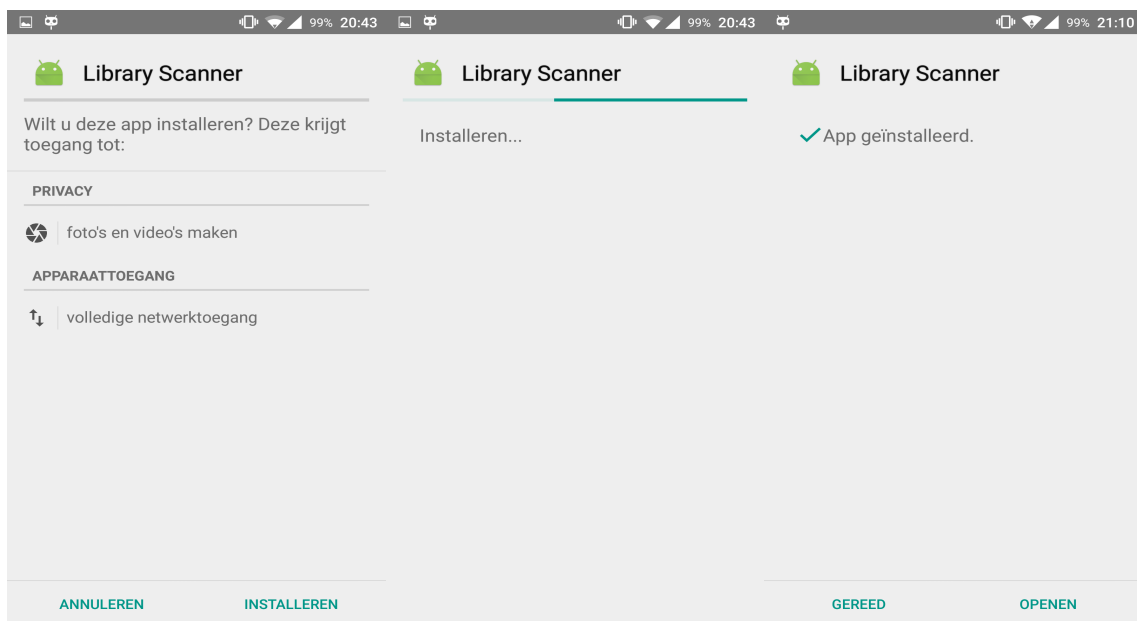


Fig. 69: Permission screen before installation

Fig. 70: Installation

Fig. 71: Installation completed

The application is now installed on your smartphone and ready to use.

9 Conclusion

This application can become a very handy tool for the people using it. It can save valuable time and therefor also money.

The application itself has completed the objectives that were asked. It implements a log in system so that only authorised users can use it. It implements the barcode scanner, which is a huge time saving factor. The data can be entered manually as well. This comes in handy when the book that the user wants to request is not found in the Google Books database.

There are a couple of expansions that could be possible. I did not include these in the final project since I didn't have the time to get them fully working. For example the cancel button could link to a list retrieved from the database showing the current requested books by the user. There could be a way for the user to enter the ISBN code manually or even implement a way to search on the book title.

I found it very interesting to work on this application since I had to use my basic knowledge from the Android coding language and improve it by doing online research. The Android language is very well documented online. Also the use of PHP and MySQL was interesting. I learned how to combine all three of these languages to perform some more complicated tasks.

Another adaptation that can be done is placing the database on a (paid) webserver that can support such a large database. This will greatly improve the functionality. Also the creation of a webpage for the person in charge of managing the book purchase requests. These are all possible improvements that can be added to this project. I am sure this application can become useful in a productive environment.

10 List of figures

Fig. 1: Graph from the market share (1)	3
Fig. 2: Graph from the Android version spread as of 04/2015 (2)	5
Fig. 3: Screenshot from checklogin.php	9
Fig. 4: Screenshot from the addtodatabase.php script.....	11
Fig. 5: Screenshot from the cancelorder.php script.....	12
Fig. 6: Screenshot from the login.xml file	17
Fig. 7: Screenshot from the layout on mobile phone.....	18
Fig. 8: Screenshot from the first part of the XML file	19
Fig. 9: Screenshot form the second part of the XML file	21
Fig. 10: Screenshot from the third part of the XML	23
Fig. 11: Screenshot from the main activity implementing the XML.....	23
Fig. 12: Screenshot from the imports used by the login	24
Fig. 13: Screenshot from the declaration of the variables used by the login.....	25
Fig. 14: Screenshot from the declarations and click listeners.....	25
Fig. 15: Screenshot from the code used by the login button and the IP change button....	26
Fig. 16: Screenshot from the code before execution of the try login class.....	26
Fig. 17: Screenshot from the code executed in the background by the try login class.....	27
Fig. 18: Screenshot from the code executed after the try login class	28
Fig. 19: Screenshot from the imports of main activity	29
Fig. 20: Screenshot from the declaration of variables. And jsonparser class	30
Fig. 21: Screenshot from the on create method.....	30
Fig. 22: Screenshot from the on click listener first part: empty input protector	31
Fig. 23: Screenshot from the click listener second part: warning messages.	31
Fig. 24: Screenshot from the method to utilise keyboard in the edit texts.....	32
Fig. 25: Screenshot from the code creating the layout for the ratings.....	32
Fig. 26: Screenshot from the code for the web view linking Google Books (11)	33
Fig. 27: Screenshot from the code starting the barcode scanner.....	33
Fig. 28: Screenshot from the code from the clear button	33
Fig. 29: Screenshot from the cancel button	34
Fig. 30: Screenshot from the onActivity class (12)	34
Fig. 31: Screenshot from the get book details class. Before executing	35
Fig. 32: Screenshot from the in background method of get book details (13).....	36
Fig. 33: Screenshot from the post execute of the Get book details class (14)	38
Fig. 34: Screenshot from the Get book cover class (15).....	39
Fig. 35: Screenshot from the pre execute Send to database class.....	39
Fig. 36: Screenshot from the Do in background method from the send to database class	40
Fig. 37: Screenshot from the post execute send data to database class.....	40
Fig. 38: Screenshot from the pre execute from the cancel book order class	41
Fig. 39: Screenshot from the background code from cancel book order.....	41
Fig. 40: Screenshot from the post execution from the cancel book order class	41
Fig. 41: Screenshot from the imports used by the Json Parser class.....	42
Fig. 42 : Screenshot from the code used to send added parameters to the database(16)43	
Fig. 43: Screenshot from the fields of the libros_total table.....	47
Fig. 44.: Screenshot from the fields in the personel table	48
Fig. 45: Example of retrieved book information	53

Fig. 46: Screenshot from the database with 3 books entered (2 automatic one manually)	54
Fig. 47: Screenshot of the server with the middle book cancelled. (done by rescanning the book and sending cancel request)	54
Fig. 48: Screenshot server not found	55
Fig. 49: Screenshot changing IP address	56
Fig. 50, Fig. 51 and Fig. 52: Screenshots from the Username and Password protection..	57
Fig. 53: Screenshot invalid Username or Password	57
Fig. 54: Screenshot successful log in	58
Fig. 55: Screenshot install dialog when application is not found.....	59
Fig. 56: Screenshot barcode scanner.....	59
Fig. 57: Screenshot retrieved book details	60
Fig. 58: Screenshot from data being entered manually	61
Fig. 59: Screenshot dialog screen with options	62
Fig. 60: Screenshot Data uploaded to server	62
Fig. 61: Screenshot link to book through webview.....	63
Fig. 62: Screenshot book request cancelled successfully	64
Fig. 63: Screenshot book not found or not the right user to cancel.....	64
Fig. 64: Android File Transfer	66
Fig. 65: Selection from the settings menu on Android 5.0.2	66
Fig. 66: Location of the security tab on Android 5.0.2	67
Fig. 67: Unknown sources switch on Android 5.0.2	67
Fig. 68: File explorer on android 5.0.2.....	68
Fig. 69: Permission screen before installation.....	68
Fig. 70: Installation.....	68
Fig. 71: Installation completed	68

11 List of tables

Table 1: Sequence diagram for login.....	10
Table 2: Sequence diagram from the connection and retrieving of book details	11
Table 3: Flowchart from the complete application.....	14

12 Bibliography

- (1)International Data Corporation. (sd). *Smartphone OS Market Share*. Opgeroepen op 04 30, 2015, van Website of IDC: <http://www.idc.com/prodserv/smartphone-os-market-share.jsp>
- Wikipedia. (sd). *Android Operating System*. Opgeroepen op 04 30, 2015, van Wikipedia: http://en.wikipedia.org/wiki/Android_%28operating_system%29
- XZing. (sd). *integrate ZXing reader to android application*. Opgeroepen op 04 29 15, 30, van github: <https://github.com/zxing/zxing/wiki/Scanning-Via-Intent>
- (4)Google Books API. (sd) *information for connecting to Google Books*. Last checked on 04 30, 2015, from Google: http://developers.google.com/books/docs/v1/getting_started
- (5)Android Log in. (sd) *Create Login screen*. Last checked on 04 30 ,2015, from androidhive: <http://www.androidhive.info/2012/01/android-login-and-registration-with-php-mysql-and-sqlite/>
- (6)PHP add to database. (sd). *Add data to database using php*. Last checked on 04 30, 2015, from stackoverflow <http://stackoverflow.com/questions/4910415/android-sending-data-to-be-stored-in-mysql>
- (7)Send data as an extra. (sd) *add data as an extra*. Last checked on 04 30 ,2015, from stackoverflow: <http://stackoverflow.com/questions/6707900/pass-a-string-from-one-activity-to-another-activity-in-android>
- (8)Add alertdialog and progressdialog. (sd) *add the alertdialog and progressdialog*. Last checked on 04 30, 2015, from cssinnovations: <http://cssinnovations.blogspot.com.es/2012/07/selection-list-in-alert-dialog-box.html>
- (9)Perform a thread in the background. (sd) *perform thread in background*. Last checked on 04 30 ,2015, from androidsearch: <https://androidresearch.wordpress.com/2012/03/17/understanding-async-task-once-and-forever/>
- (10) Create a input method. (sd) *Creating an input method for enter button*. Last checked on 04 30,2015 from Google: <http://developer.android.com/guide/topics/text/creating-input-method.html>
- (11) Webview. (sd) *how to implement the web view*. Last checked on 04 30, 2015 from Google android developer: <http://developer.android.com/reference/android/webkit/WebView.html>

(12) on action result. (sd) *how to retrieve a result to activity* Last checked on 04 30, 2015 from Google android Developer: <http://developer.android.com/training/basics/intents/result.html>

(13) implement HTTPget and post.(sd) *how to implement http post and get*. Last checked on 04 30, 2015 from Google android Developer: <http://developer.android.com/reference/org/apache/http/client/methods/HttpGet.html>

(14) json object information. (sd) *information on using json object*. Last checked on 04 30, 2015 from Google android Developer: <http://developer.android.com/reference/org/json/JSONObject.html>

(15) Bitmap usage. (sd) *information of using the bitmap for coverthumbnail retrieval*. Last checked on 04 30, 2015 from Google android Developer: <http://developer.android.com/reference/android/graphics/Bitmap.html>

(16) Usage of the Json parser.(sd) *How to use the jsonparser in android*. Last checked on 04 30, 2015 from Google android Developer: <http://stackoverflow.com/questions/9605913/how-to-parse-json-in-android>