Emer Hennebry

14310781

MyReceiptsApp

Technical Guide

20/05/2018

Project Supervisor Name: Jane Kernan

Table of Contents

Introduction

Overview

My final year project is an Android application which gathers in-store receipts from Near Field Communication (NFC). The apps aim is to provide a simple way to organise and budget purchases. The app achieves this with six different sections to it; adding receipts through NFC, returning a product through NFC, viewing receipts, receipts sorted into different folders, recommended shops and an overall calculation section.

Motivations

It can be very easy to misplace paper receipts. This app intends to eliminate that and provide an efficient way to store receipts which will also be a proof of purchase. Customers can't receive receipts in shops through NFC yet, so I have been creating prototype receipts on NFC tags. My app is proof to shops that it can be done. I believe many shops would be willing to implement a system which provides receipts with NFC through this app. This will be efficient for retailers and, through the "recommended shop" section of the app, will give them advertising to their targeted customers. It will be more efficient for customers by making it easier for them to store their proof of purchase from their shop. From the retailers perspective, it will be a more efficient point of sale process with less paper. It will create the digital connection that is needed for the large number of customers that still shop in-store.

Research

Near Field Communication(NFC)

NFC can send and receive data over radio waves within a short-range, normally within 4cm. It is low-power communication between compatible devices. It allows a simple and safe two-way interaction.

NFC technology has become a reality for many companies and users. You would except to have NFC within most Android smartphones made in the past few years and Apple are now starting to incorporate NFC into newer versions of the iPhone. This branch of technology is changing rapidly, and more apps are being developed with NFC. However, development with NFC is still new and growing and I found throughout the project I had to do extensive research when implementing NFC.

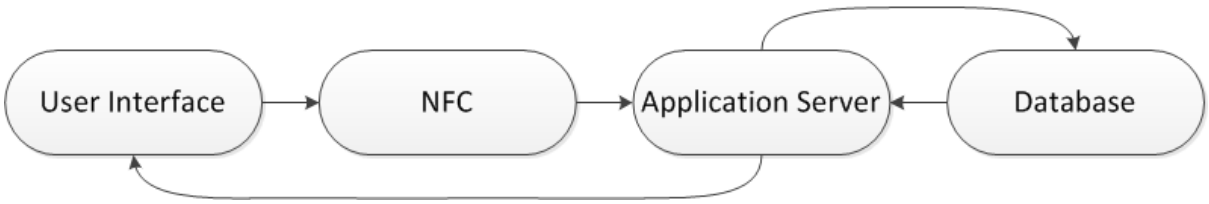
It can be easy to assume that NFC is an extension of Wi-Fi or Bluetooth technology. All three systems support wireless communication and data transfer. However, NFC operates using electromagnetic fields whereas Wi-Fi and Bluetooth use radio transmissions.

When NFC is enabled on your device it is generating an unharmful electromagnetic field, this field is created by an antenna using magnetic induction. The active device (the initiator) starts the

communication when the NFC tag (the passive device) is placed near it. Then, the tag modulates this energy to send data back to the active device.

Design

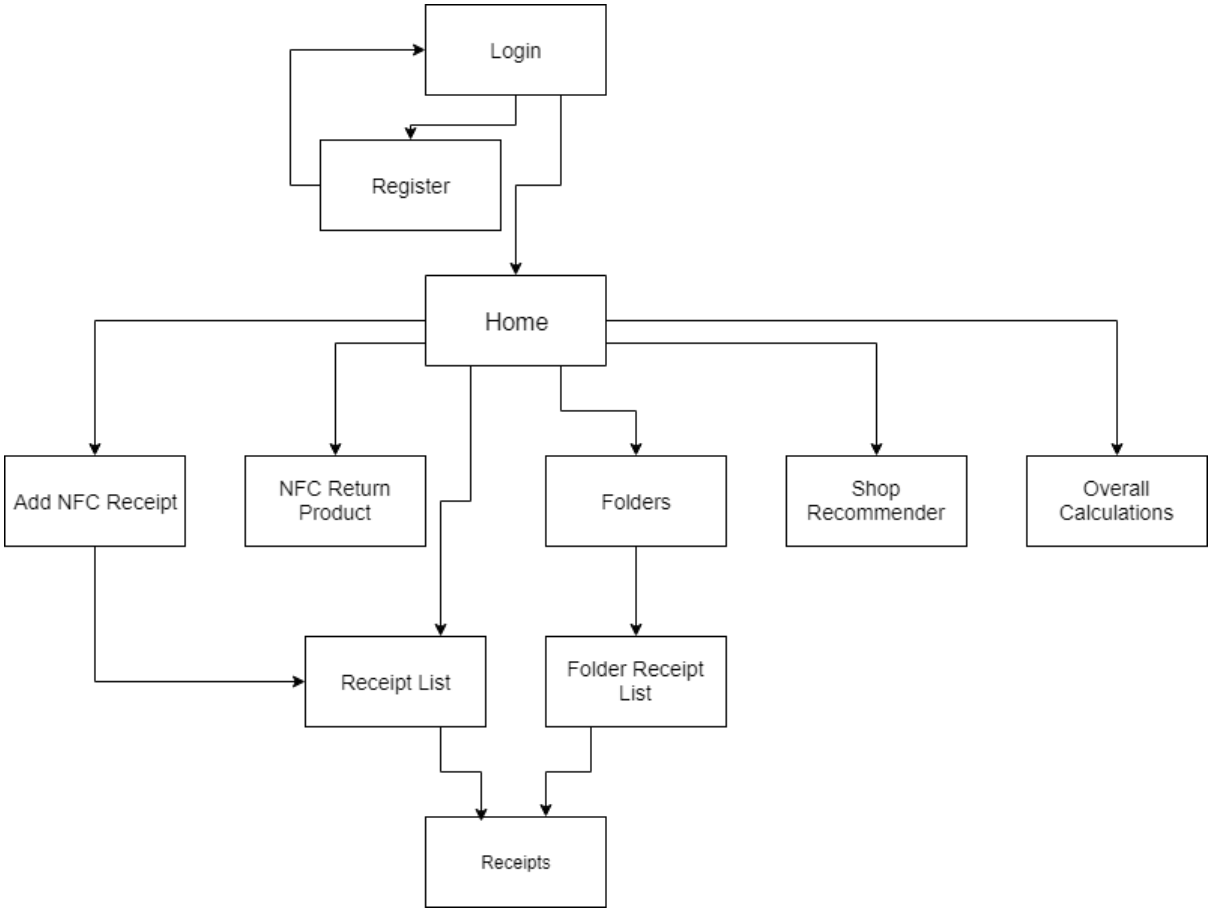
System Architecture



The diagram above illustrates the system architecture. This diagram is after the user was signed in and they're using the application. In the diagram there are four aspects user interface, NFC, application server and database.

Application Diagram

High-level Design

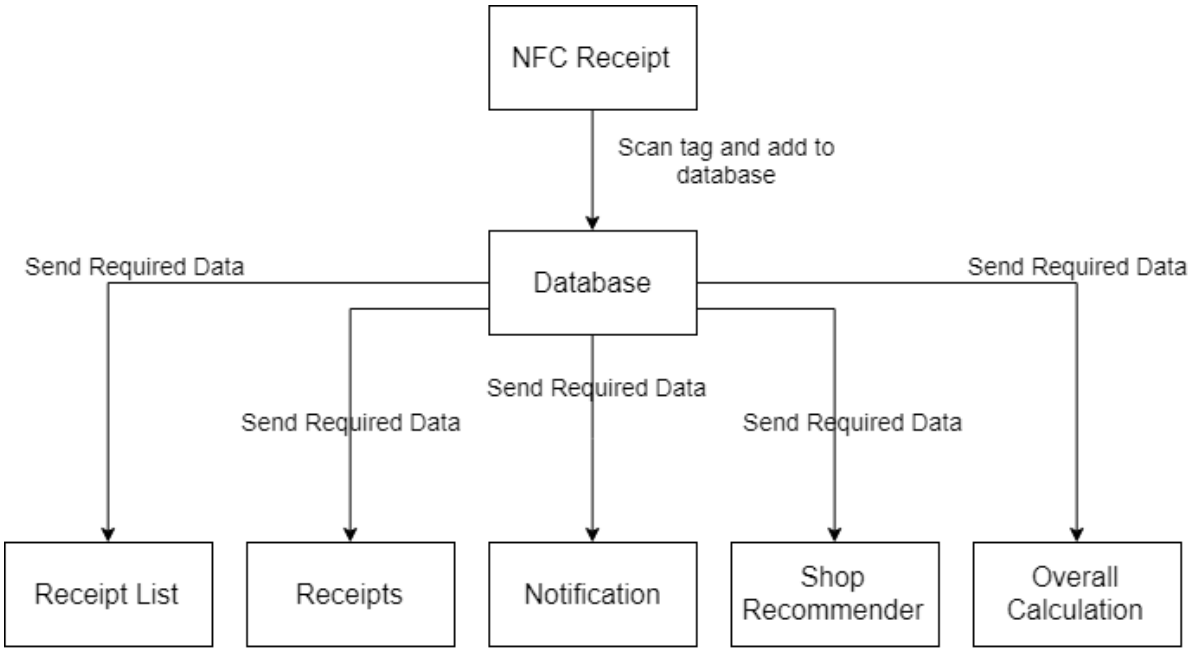


This is a breakdown of the whole application and you can see which section link to each section.

The app starts on the login in page and from there it either goes to the register page or the home page. From the home page the rest of the app accessible.

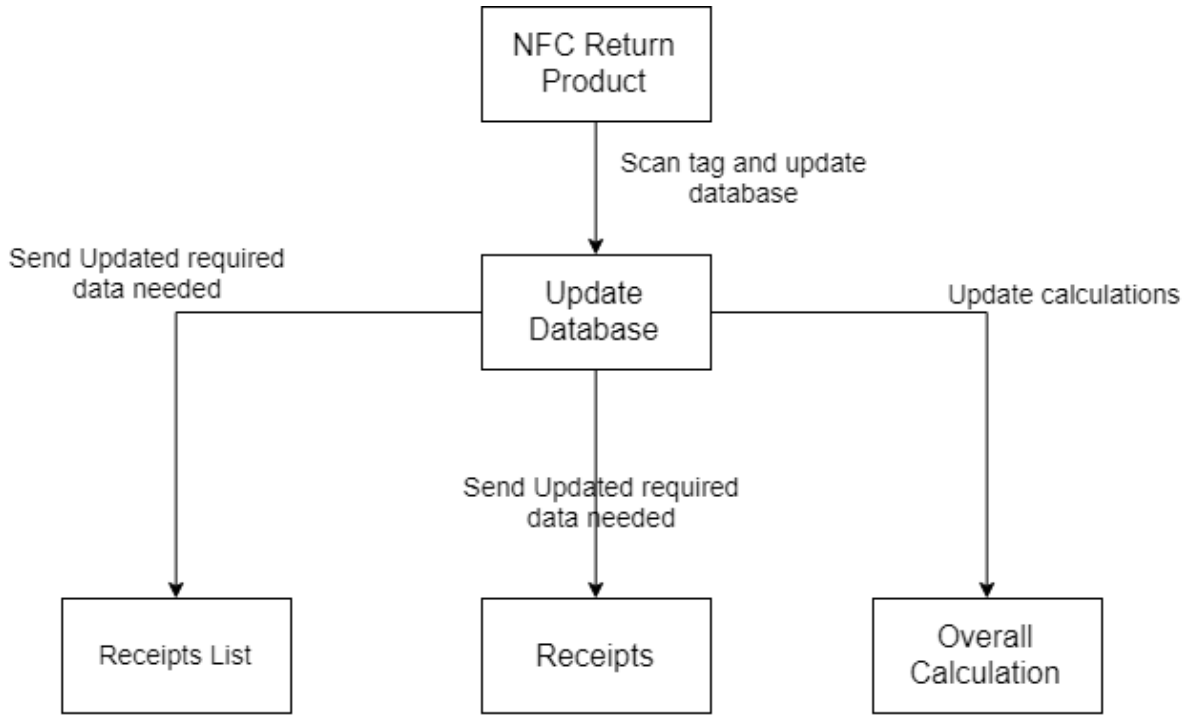
Data Flow Diagrams

NFC Receipts



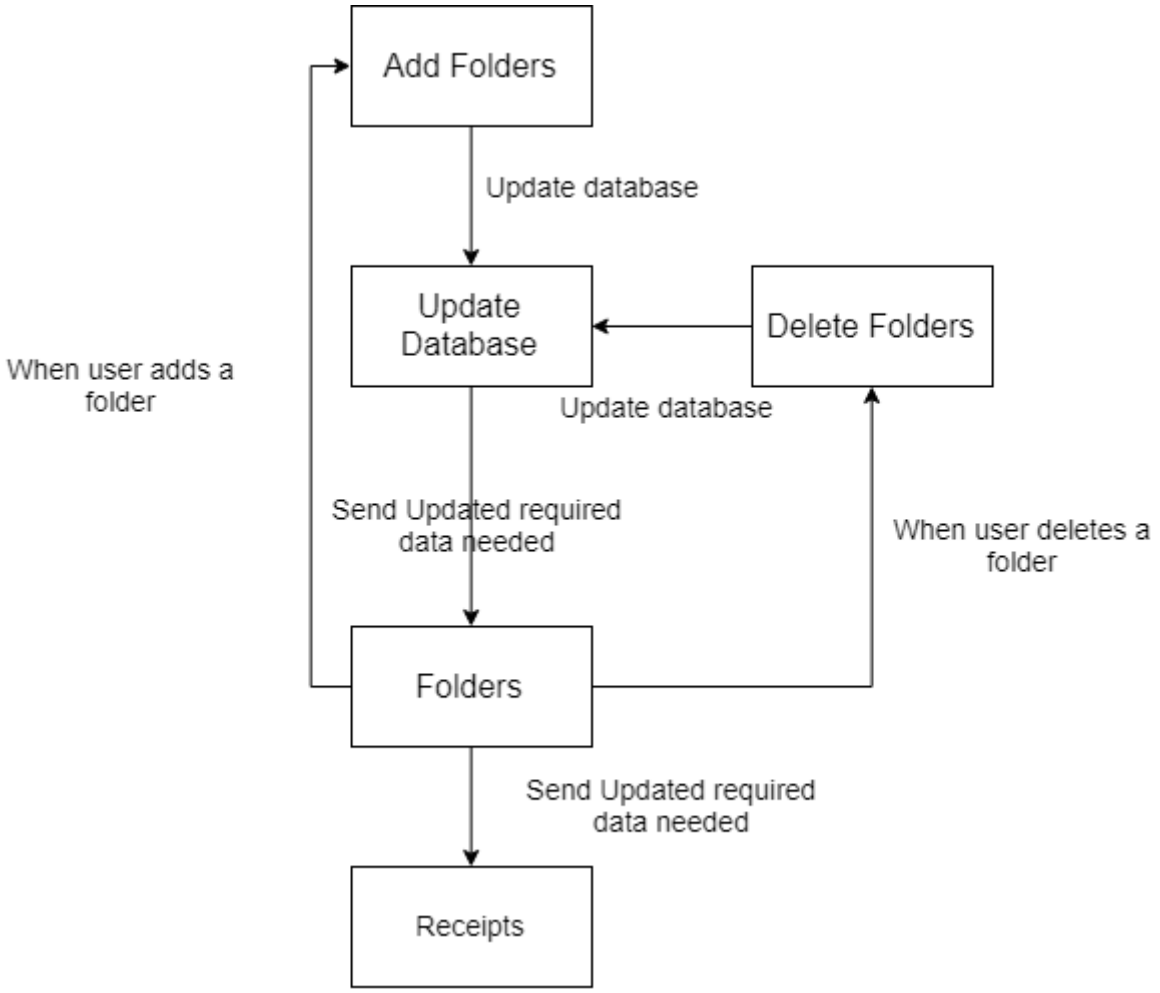
The diagram above is the data flow diagram for when an NFC Receipt is tapped at the back of the phone. The information is saved into the database and then the features use the information required from the database to perform their tasks.

NFC Returns



Here is the data flow diagram for when a product is being returned through NFC. The database is updated and then the features of the app our updated with the correct information.

Folders



Folders can be added and deleted to the database. The database needs be updated when the user adds or deletes a folder.

Data Dictionaries

The formats of the NFC tags are important. If the NFC tag are formatted incorrectly the receipt will not be sent to the database. The NFC tags need to be formatted in NDFE editor like so:

Receipts Tag

Receipt id	String
Shop Name	String
Total Price	Int or Double
Product Quantity	Int
Extras Quantity	Int
Product(s) Name	String
Price(s)	Int or Double
Extra(s) Name	String
Expiry Date(s)	String

Returns Tag

Receipt id	String
Product Name	String
Product Price	Int or Double

Implementation

Project Technology Used

- Android Studio is the integrated development environment for the app
- Eclipse is used for the NFC Eclipse plugin which provides the NFC data exchange format (NDEF) editor to develop the NFC tags
- Java is the primary language
- XML is used to implement the design and layout of the app
- SQLite Database is used to store all data contained within the app

Java is my primary language because it has a big open source support, with many libraries and tools available. It is also the native language for Android SDK, which provides the API libraries and tools necessary to build, test, and debug apps with Android.

NFC

The NFC tag is read on NFCReader.java and ReturnActivity.java activity classes. NFCReader.java is the activity class that reads the receipt information and sends it to the SQLite database. The ReturnActivity.java activity class reads the product information from the tag and finds the product in the database and marks it as returned.

Before I could get started on the NFC activity pages I got access to the NFC hardware, by applying for permission in the manifest. It was also important to select a minimum SDK version of at least level 10 in the app gradle file, because NFC is only supported after Android 2.3.3.

```
<!-- The NFC permission -->
<uses-permission android:name="android.permission.NFC" />
```

The first step I needed to take to implement NFC in an activity class was to import the required packages. I imported the following packages:

- import android.nfc.tech.NfcF;
- import android.nfc.NfcAdapter;
- import android.nfc.Tag;

Before the NFC tag is tapped against the device for an NFC action, on an activity page it will inform the user if their device supports NFC or if the user has the NFC switched off on their device.

```
// Initialize NFC
nfcAdapter = NfcAdapter.getDefaultAdapter(this);

if(nfcAdapter == null) {
    Toast.makeText( context: this,  text: "Device doesn't support NFC", Toast.LENGTH_LONG).show();
    // End running if device doesn't have NFC
    finish();
    return;
}
if(!nfcAdapter.isEnabled()) {
    // if the NFC is turned off on the device, a Toast (a little pop up) appears
    Toast.makeText( context: this,  text: "NFC is disabled, go to phone setting to enable", Toast.LENGTH_LONG).show();
}
```

The code above initializes the NFC adapter and if there is no NFC adapter found a toast appears displaying “Device doesn’t support NFC”. A toast is a little pop up that appears on the bottom of the

screen. If the device does support NFC but it is disabled on the phone another toast appears displaying “NFC is disabled, go to phone setting to enable”.

How NFC tags are dispatched to applications is a complex concept. The Tag Dispatch System is done by creating an intent that encapsulates the NFC tag and its identifying information. If the user scans the tag outside the activity page and if more than one application can handle the intent, the Activity Chooser is presented so the user can select the activity they want the device to go to. There are three different filters for tags. The list is sorted from the highest to the lowest priority.

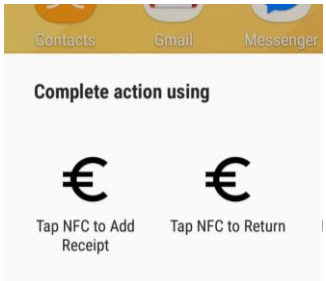
1. ACTION_NDEF_DISCOVERED
2. ACTION_TECH_DISCOVERED
3. ACTION_TAG_DISCOVERED

ACTION_NDEF_DISCOVERED is used to start an activity when a tag contains NDEF payload. ACTION_TECH_DISCOVERED is used if no activities registered to handle the ACTION_NDEF_DISCOVERED intent. ACTION_TAG_DISCOVERED is started if no activities handle the other two intents.

In the manifest for each of the NFC activities pages I have created the ACTION_NDEF_DISCOVERED intent. This intent gives a better user experience, as this intent allows the user to start their application at a more appropriate time than the other two intents. Here is where the ReturnActivity.java activity is defined in the manifest with the ACTION_NDEF_DISCOVERED intent. The NFCReader.java activity is also the same.

```
<activity android:name=".ReturnActivity"
    android:label="Tap NFC to Return">

    <intent-filter>
        <action android:name="android.nfc.action.NDEF_DISCOVERED" />

        <category android:name="android.intent.category.DEFAULT" />

        <data android:mimeType="text/plain" />
    </intent-filter>

</activity>
```

In the NFCReader.java and ReturnActivity.java classes I then set special flags to control how the intents are handled. You start the activity java class that you have selected when the tag is tapped outside the activity in the application. For this app you can either choose to add a receipt or return a product. However, the user most login first.



I also then enabled foreground dispatch to the given Activity. This will give priority to the foreground activity when dispatching a discovered tag to an application. So, in other words if you're on the NFCReader activity it will take that action instead of giving you the option of which activity you want, that is, instead of the options set out in the picture above. Also, if you're in the ReturnActivity class, it will take the same action in that class.

The code below is the snippet of code to read the data off the NFC tag.

```
// Initialise StringBuilder
StringBuilder tagText = new StringBuilder();

// Get array of data to parse
Parcelable[] data = intent.getParcelableArrayExtra(NfcAdapter.EXTRA_NDEF_MESSAGES);

// Check if we have data payload, if not it is an empty but formatted tag
if (data != null) {
    try {
        // Loop through data and get stored records
        for (Parcelable aData : data) {
            NdefRecord[] NDEFRecords = ((NdefMessage) aData).getRecords();

            // Loop through record and print data
            for (NdefRecord NDEFRecord : NDEFRecords) {
                if (NDEFRecord.getTnf() == NdefRecord.TNF_WELL_KNOWN && Arrays.equals(NDEFRecord.getType(),
                        NdefRecord.RTD_TEXT)) {
                    byte[] tagPayload = NDEFRecord.getPayload();

                    tagText.append(new String(tagPayload, offset: (tagPayload[0] & 0077) + 1,
                            length: tagPayload.length - (tagPayload[0] & 0077) - 1,
                            ((tagPayload[0] & 0200) == 0) ? "UTF-8" : "UTF-16")).append(";");
                }
            }
        }
    } catch (UnsupportedEncodingException e) {
        // catch malformed tag
        throw new IllegalArgumentException(e);
    }
```

If the tag is malformed this method will throw an exception error and if the tag is empty but is a formatted tag the activity will not continue. Instead of using for loops I used foreach iteration syntax because for loops which iterate over collections or arrays can be replaced with the foreach iteration syntax in higher java. The program loops through the data on the NFC tag.

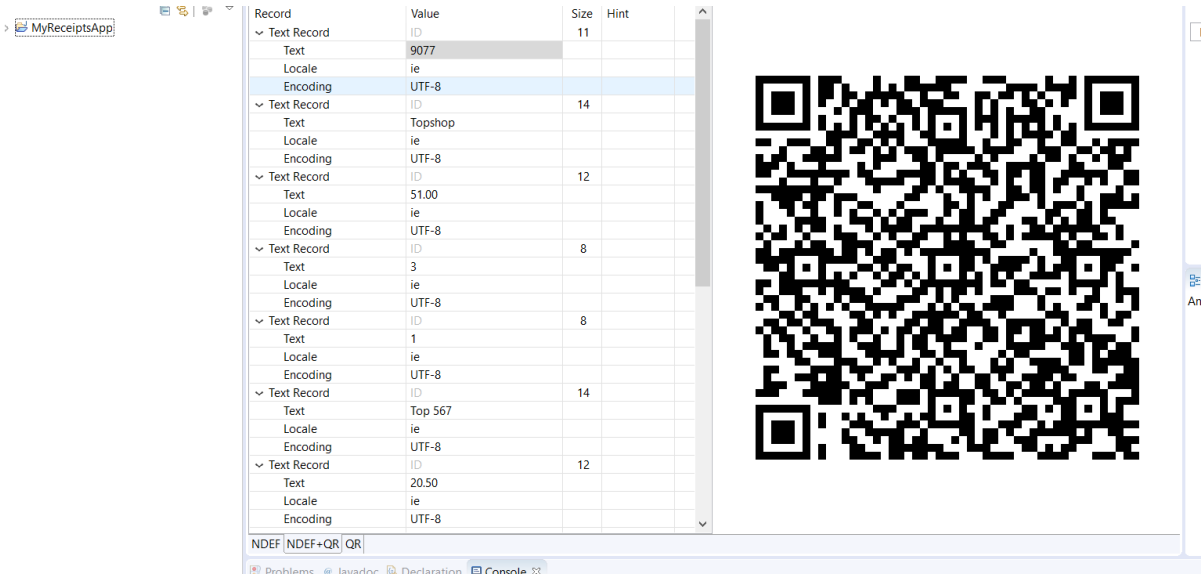
The bitwise operators 0077 gets the length of the language code from the tag and the bitwise operator 0200 gets the bit identifying whether it is stored as UTF-8 or UTF-16. Once the language is known the information can be extracted from the tag and will start each entry in plain English. The binary data from the NFC tag is now put into a string form. Each input that is placed in the string is separated with a semicolon, which makes it easier to extract information from the string and store in the database.

NFC Tags

The data on tags can be encoded in a variety of formats. However, there is one format that is widely used called NFC Data Exchange Format (NDEF). Formatting a tag in NDEF format allows an easy exchange of the tag data with systems that leverage the format.

To develop the NFC tags with the receipt information I used the NFC Eclipse plugin which provides the NDEF editor to program the NFC tags.

I create the text records I need in the NDEF editor and then a QR code is created with these records. Here is what the NDEF edit and QR code look like on Eclipse:



I then use the NFC Developer App [1] on my phone to scan the NDEF editor QR code using the camera on my phone, I then tap the NFC tag to write the receipt information on the tag.

Shop Recommender

The shop recommender section is to get customers to explore different shops similar to shops that they buy in. This section is a benefit to shops as they can promote themselves to the targeted customers.

Due to result conversation about data protection regulations, for the shop recommender system I don't use any personal information to generate recommended shops. The information I employ to show shops and deals that the user would like is the last shop that the user bought in. If a certain shop is the last shop you shopped in, then a list of pictures from different shops, associated with the last shop you bought in, and a list will appear with the use of a listView in the Shops section.

SQLite Database

The database is a key part of this project. SQLite is used because it is an embedded database. That means that this database is saved on your phone, so no third party has your personal information or receipt information. Also, using SQLite means information can be added without internet connection, which is an important consideration as not everyone has mobile data or WiFi while shopping.

I have six tables in my database

- **users:** Contains five columns;
  - userid (Primary Key), name, username, useremail, userpassword
- **receipts:** Contains seven columns;
  - receiptid (Primary Key), username, shopname, price, receiptDate, receiptTime, productsQuality, extrasQualities
- **products:** Contains four columns;
  - id (Primary Key), receiptid, productName, productPrice
- **extras:** Contains four columns;
  - id (Primary Key), receiptid, extraName, expiryDate
- **folders:** Contains three columns;
  - folderid (Primary Key), receiptid, folders
- **categories:** Contains three columns;

- categoryid(Primary Key), usernameFolder, category

These can be seen in the databaseDetails section in my code in MyDBHandler.java, which extends the SQLiteOpenHelper class. Any column that includes the price is defined as a real, this is so that it can accept double numbers and so that the database can get the correct sum from the receipts.

The “receiptid” column in products, extras and folders are Foreign Keys, so that I can join their tables with the “receipts” table, which has the primary key of column “receiptid”. I can join the tables together using the primary key from the “receipts” table and the foreign key in other tables mentioned. I then use a cursor to go through the database to get the information I need and display it.

For example, the code below is from the displayReceipts.java class, which displays the information for the list of all your receipts. What this program demonstrates is that in the if statement it looks for the user by username and then goes through the rows from the “receipts” table getting the columns required and adding it to the listView that has been created. The prices from the database are rounded to two decimals.

```
Intent intent = getIntent();
passedVar = intent.getStringExtra( name: "username");

listview = (ListView) findViewById(R.id.listViewReceipts);
listAdapter = new ReceiptDisplayListAdapter(getApplicationContext(), R.layout.row_layout);
listview.setAdapter(listAdapter);
myDBHandler = new MyDBHandler(getApplicationContext(), name: null, factory: null, dataBaseVersion);
cursor = myDBHandler.getListContents(passedVar);

if(cursor.moveToFirst()) {
    do {
        String username = cursor.getString(cursor.getColumnIndexOrThrow( s: "usersname"));
        if(username.equals(passedVar)) {
            ID_ = cursor.getString(cursor.getColumnIndexOrThrow( s: "receiptid"));
            String shop;
            double price;
            String date;
            shop = cursor.getString(cursor.getColumnIndex( s: "shopname"));
            date = cursor.getString(cursor.getColumnIndex( s: "receiptDate"));
            price = cursor.getDouble(cursor.getColumnIndex( s: "price"));
            double priceRounded = (Math.round(price * 100.0) / 100.0);
            receiptList = new Product(ID_, shop, priceRounded, date);
            listAdapter.add(receiptList);
        }
    } while(cursor.moveToNext());
```

Here is the method in the MyDBHandler.java class that is called in the above program. It gives access to the needed data by the query which selects every column from the “receipts” table only where the string username entered in matches the username in the table and it orders by the column “shopname”.

```
//Used to get information from the receipts table
public Cursor getListContents(String username) {
    SQLiteDatabase db = this.getWritableDatabase();
    Cursor data = db.rawQuery( sql: "SELECT * FROM " + TABLE_RECEIPTS + " WHERE " + COLUMN_USERSNAME + " = ?"
                    + " ORDER BY " + COLUMN_SHOPNAME, new String[] {username});
    return data;
}
```

There's a folders table and a categories table because the categories table is the list of folders available to select from along with the user who created them, and the categories table is the list of "receiptids" and the folder names that they are placed in with. Receipts can be in multiple folders and if a user deletes a folder it is deleted from the categories section, so users can add the folder back and the same receipts will be in it.

ListViews and RecycleViews

Listviews and RecycleViews are widgets that are used to display information as a scorable list. ListViews and RecycleViews both have their pros and cons, but their functionality is the same. Both required an Adapter class to display in a selected format. ListViews extends the arrayAdapter class and the RecycleView extends RecyclerView.Adapter<FolderAdapter.FolderViewHolder>.

Problems and Resolutions

NFC

This NFC technology has been around for a few years, however development with NFC is still new and growing, and I found throughout the project I had to do extensive research when implementing NFC particularly at the start of my project.

I wanted to use NFC tags as a prototype for use by shops because they can store NDEF text records, which could represent the strings contained within the receipt. I struggled at the start on how to put information onto the tag. I tried first with Json files, but I needed a way to connect with the tag. After more research, I found the NFC plugin on Eclipse. I though I would have to move my project from Android Studio to Eclipse, but then I found that I could use Eclipse to send information to the tags. The NFC plugin lets me create text records in NDEF format and it then generates a QR code which I scan using the NFC Developer App [1] using the camera on my phone, I then tap the NFC tag to write the receipt information on the tag.

I think the most difficult part of my project was figuring out how to get this NFC information that I implemented onto the NFC tag on to the app, sort it and send the information to a database. Getting the receipt information from NDEF format onto the app as readable text is explained in the implementation section of my app. It required extensive research and many trial and errors.

Spinner

A spinner is a dropdown menu that I created in the receipts, so users can easily select which folder they want that receipt to be in. I had problems setting up the spinner because the spinner gets the information on which folders to display from the database, however, the hint "Select Folder" needs to be in that list for every user. How I fixed this was when the folders section or the receipt pages are opened and there is no folder for that user "Select Folder" is added to the list of folders. This will then be at the top of every list of folders. The user can't select the "Select Folder" as folder and it simply just there to help the user know that they need to select the folder on the receipt.

Results

Please see the testing documentation named as "TestingManualFinal" in the testing section on gitlab.

Future Work & Possible Improvements

Deploy to Shops

As shops don't send receipts through NFC yet, I will need to advertise the benefits to shops of sending NFC receipts. I will then need to use an Arduino, which will work in a similar way as my NFC tags, that is, be connected to the cash register. As more shops deploy the system the more that can be created

with the app such as, the discounts that shop has, include barcodes that shops generate on their paper receipts onto the receipts on the app and the shop recommender section can expand.

Shopping Online

With this app I am looking at the customers that shop in-store. I hope to extend my app to online shopping as well, by using QR codes. I hope to do this by giving the user a choice of adding the receipt by either NFC or by using their phone camera to scan a QR code. This would be expanding the app to a customer's full expenditure.

Security

One of the reasons why I choose to send my receipts trough NFC is because NFC communications are harder to eavesdrop than other technologies with bigger wireless ranges. Having the password to the app makes sure that it is the user that is receiving and viewing the receipts. The NFC technology in mobile devices can already be used for secure applications, however, I would like to improve the security even more by adding encryption and providing a wider secure channel to send the data, using standard key protocols such as, Diffie-Hellman key exchange or by a unique digital signature, which are methods for establishing a shared secret key over an insecure communication channel, which would prevent the unlikely event of a man in the middle attack. I would like to develop a secure architecture and look in to this more to provide the highest of security and to be hardened against attacks on the phone when sending the data. NFC technology is rapidly developing and therefore more security is becoming available as it grows.

Conclusion

In summary, the aim of this project was to create a receipt holder application utilising NFC technology.

Every aspect of the project needed careful planning. It was a research project however, so even with careful planning there were some difficulties along the way which had to be worked through or worked around. Every aspect of the app needed to be considered and made work as best as possible to make sure the application satisfies its users. In this report, it can be seen that, with adequate planning and implementation, a useful application can be developed, which can be deployed and used in the real world.

I feel I have delivered an app that users would be happy to use and I have built in scope for this application to be developed further and bring it to the next level.

Appendices

[1] Play.google.com, NFC Developer, https://play.google.com/store/apps/details?id=com.antares.nfc&hl=en

[2] Android Developers, 2018, https://developer.android.com/reference/classes.html

[3] Steven Brett, 2013, Android Gym Application Utilizing Near Field Communications (NFC), https://www.cs.cf.ac.uk/PATS2/@archive_file?c=&p=file&p=202&n=final&f=1-Final_Report_-_Scm9bs_-_Brett_Stevens.pdf

[4] Thomas Skjølberg, 2013, Near Field Communications. https://github.com/skjolber/Fagmote/tree/master/Android/Near%20Field%20Communications.

[5] Android Developers, 2018, https://developer.android.com/guide/topics/connectivity/nfc/advanced-nfc.html#foreground-dispatch

[6] How to add a hint to an Android Spinner, 2015 https://android--code.blogspot.ie/2015/08/android-spinner-hint.html

[7] Android Login and Register with SQLite Database Tutorial, 2016, http://www.androidtutorialshub.com/android-login-and-register-with-sqlite-database-tutorial/

[8] Android Developers, 2018, https://developer.android.com/training/notify-user/build-notification.html

[9] DZone, Create Database Android Application in Android Studio 3.0, 2017, https://dzone.com/articles/create-a-database-android-application-in-android-s

[10] Eddydn, Android Grid Layout demo with Click / Select Item, 2018, https://github.com/eddydn/AndroidGridLayout