UPPSALA
UNIVERSITET

IT 11 070
Examensarbete 15 hp
November 2011

An Android Application for Saving Ideas

Göran Nehlin

Institutionen för informationsteknologi
Department of Information Technology

Sammanfattning

En Androidapplikation är ett program för smartphones som kör operativsystemet Android. Det här examensarbetet handlar om hur jag utvecklat en Androidapplikation för att kunna spara idéer och ladda upp dem till en webbtjänst. Tanken med applikationen var att den skulle komplettera den nystartade webbtjänsten Creado, en tjänst som kunde liknas vid en delad whiteboard där användarna kunde ladda upp och kommentera bilder, video, text och andra filer. Webbtjänsten var centrerad kring att jobba kreativt och dela idéer. Tanken var att den skulle vara en plats att spara och dela sina idéer.

Tanken med Androidapplikationen var att användarna av webtjänsten skulle ha en möjlighet att spara ner sina idéer även när de inte hade tillgång till en dator. Användaren kan spara anteckningar i text-, bild- och ljudformat och ifall hon saknar internetuppkoppling ska anteckningarna kunna sparas lokalt på telefonen tills dess att de kan laddas upp.

Uppgiften handlade om att designa och implementera en applikation som klarade av användarinteraktion, skapande av anteckningar och uppladdning av de skapade anteckningarna. Målet var inte att göra en applikation färdig att släppas, utan snarare en stabil grund att bygga en sådan applikation från.

I den här rapporten beskriver jag designprocessen. Jag har inriktat mig på tre huvudområden: Datalagring, överföringsprotokoll och användargränssnitt. Datalagringen beskriver den bakomliggande SQLitedatabasen, hur den samarbetar med vanliga filer sparade på telefonenes externa SD-kort och hur applikationen hanterar avbrott i körningen från till exempel telefonsamtal eller SMS.

Överföringsprotokollet handlar om hur jag överför anteckningarna från databasen och SD-kortet till ett webbgränssnitt för Creado. Creados webbgränssnitt blev aldrig klart så därför designade jag ett protokoll som jag tror skulle fungera bra som webbgränssnitt. Det här avsnittet behandlar både hur anteckningarna från databasen omvandlas till ett format som lämpar sig för överföring och hur applikationen och det tänkta webbgränssnittet kommunicerar med varandra.

I avsnittet om användargränssnittet beskriver jag hur applikationen är designad och fungerar från ett användarperspektiv. Där beskriver jag också hur jag tagit hänsyn till konventioner och oskrivna regler som finns för Androidapplikationer. Eftersom den här applikationen är tänkt som ett tillägg till en befintlig webbtjänst så har jag också tagit hänsyn till hur den är designad för att försöka få användare att känna att applikationen och webbtjänsten hör ihop.

Avslutningsvis beskriver jag hur applikationen borde vidareutvecklas. Som jag nämnde tidigare är den en stabil grund att bygga vidare på snarare än en färdig produkt. Här beskriver jag vad som behöver göras med den innan den är klar att släppas.

Innehåll

1 Introduction

1.1 Background

1.1.1 Creado

Creado is a web based service for storing and sharing ideas in text, image, audio or video format. Initially, the format closely resembled a shared whiteboard where users could see what other users did in real time. During the summer, Creados focus moved away from the shared whiteboard concept and more towards a note taking/sharing service, though focus is still on sharing, viewing and editing files.

When Creados focus shifted, they took a break in developing the web application. As a result of this, they did not create the planned web interface that my mobile application was going to use.

1.1.2 Android applications

An Android application (commonly known as an app) is a program for smartphones running the Android operating system. Writing applications for mobile devices requires some extra consideration compared to writing traditional desktop applications. A smartphone, compared to a desktop computer, has very limited resources. The processor is slow, the memory is limited and power usage should be kept as low as possible. Interruptions (phone calls, text messages, etc.) may occur at any time or the device may simply run low on resources and suspend/kill applications.

Android devices usually have both internal and external memory. The internal memory is where applications are typically stored, but this memory is very limited in size, so storing large files there is not advised. The external memory however, has it's own problems. Normally external memory means an external SD card[1], which might be removed and replaced by the user.

Android applications have to take all of this into account and still act reliably and fast. This includes saving user data when applications are suspended/killed as well as resuming as if nothing happened when the application is relaunched and manually putting work intensive tasks on separate threads to offload the user interface thread and keep the application responsive.

Applications for Android are written in Java using the Android Software Development Kit (Android SDK). An application is a complete package, containing graphics, text resources and layout as well as the actual program.

1.1.3 Android concepts

Here is a few Android concepts that useful to know in order to understand the design process:

[1] Secure Digital memory card. A type of memory card typically used in cameras and smartphones

Activity A specific task running in its own thread and having a user interface. Examples of Activities in my application is the view where the user can list all existing notes or create a new note. From this Activity, the Activity for creating/editing text notes can be launched or if the user wants to look at pictures that she has taken before, she can do so using the picture viewing Activity.

Service A Service, unlike an Activity, does not have a user interface. They are typically work intensive or otherwise slow tasks running in the background. The point of services is to separate work intensive tasks that do not need user interaction from the user interface to keep it responsive. An example in my application is the synchronisation Service. Once it is started, it runs in the background until complete, while the user can keep working with the application.

Intent Intents are used when you want to specify what kind of task you want to be carried out. Once you specify an Intent and launches it in an (unspecified) Activity, the operating system launches an Activity that matches your Intent. For example if we specify the intent to be "pick a picture from the device memory", the file browser set to display only pictures or the gallery application may respond to this call. This is a way to use existing applications as components in your own application.

1.2 Problem description and limitations

The typical use case for the mobile application of Creado is a user being away from her computer when a great idea strikes her. She should then be able to record this idea in some way (text, picture, audio) as well as upload it to the web application. Even if her phone currently lacks internet access, she should be able to store the idea in the phone and the mobile application should upload it as soon as possible. Further, she should be able to review the ideas that she created using her phone as well as edit ideas in text form.

While being able to view and work with existing notes (I will refer to the data created in text, picture or audio format as "notes") would be useful for a mobile application, this is beyond the scope of what I am doing in this thesis. This application is focusing on creating, displaying and to some extent editing local notes, and to upload them to the web application. Since I make local notes viewable and editable, another developer may later use this functionality by fetching notes from the web application and work with them as local data, but for the scope of this thesis, the communication is basically one-way.

The task of creating the mobile application included both designing the architecture and implementing it. I did this while at the same time learning how Android applications works, which is why I had to limit the scope to just uploading, editing and viewing local notes.

Since the web interface would not be completed while I was working on the mobile application, I had to design a protocol for transmitting notes. My aim was that this protocol could later be used directly as the web interface, or at least

used after minor modifications. Lastly, the purpose of this mobile application is not to be a complete application ready to release to the public, but rather the basics of such an application that will require some small redesign, testing and possibly optimising before public release.

2 Design

2.1 Persistent storage

The Android application must be able to store notes in a way that ensures no data is lost if the application gets suspended/terminated by the operating system. The data storage must also be efficient and not waste too much space on the internal memory, but rather use the external SD card.

2.1.1 SQLite database

There are several ways to store data on an Android device (internal file, external file or in a database). A note consists of data (title, content) and meta data (creation date, type, etc.) and to be able to store all this in a file, I would have to design a file format.

Also, there are several occasions when querying the data is necessary. For example, synchronisation[2] requires finding all unsynchronised notes, and displaying the existing notes requires getting the title and thumbnails for all notes. Querying and the fact that I would rather not design my own file format (since the web application uses a database) if not necessary led me to think that a database was the best solution.

The Android platform provides a built in SQLite database[3] which is ideal for querying and being able to access parts of notes. The database is located on the internal memory. This database however, is not suitable for large files. In fact, trying to insert a too large file into it may actually cause it to crash and become corrupted, something I learned along the way. Thus files had to be saved elsewhere, namely on the external SD card.

The database in the application is a pretty straight forward one, containing a single table for all types of notes. This is mainly for convenience reasons as performance is not an issue in a database this small. The database contains the following fields:

id Identifier for the note in the local database.

title Title for each note. For text notes, this may be null. In that case the title is computed from the first few letters in content.

content For text notes, this simply contains the text. For other notes, this contains a URI[4] to the file where the content is located.

type This determines the type (text, picture, audio) of the note.

[2] Synchronising in this context simply means to upload a note as soon as it's created or updated

[3] A light-weight SQL database

[4] Uniform Resource Identifier, in this case the path and file name of the file.

thumb This field contains the data for image thumbnails[5]. Unlike the other raw data, thumbnails are small enough not to cause any trouble in the database.

creationDate The date when the note was created.

modificationDate The date when the note was last modified. Used to determine order when listing all notes.

synced A boolean value telling whether the note is synchronised or not. Used to count and find unsynchronised notes.

globalId Identifier for the note in the global database. Will be further discussed in the synchronisation section.

The field worth discussing is **thumb**. Initially I planned to store all data in the database. Compared to audio and full size images, the impact of storing thumbnail data and having empty fields in notes of other types was negligible. As it is now, the vast majority of the database probably contains thumbnail data which may be a good reason to reconsider this design decision. Right now though, it works, and before any profiling is done, there's no telling if this would make any significant improvement.

2.1.2 External SD card

As mentioned before, data on the internal storage is precious. Not only does it bother users if you use up too much of it, if the system gets critically low on storage, data stored there might get deleted. Simply put, storing large amounts of data is not for the internal memory.

The external memory also have its problems though. For one thing it's external, meaning the user might simply remove the card and replace it with another one. This would not affect the database (since it's stored on the internal memory) so notes would still appear to exist, even though their data might not. This is a problem all Android developers have to face, and the standard solution seems to be to assume that the card is not removed, but to test for and handle the case when it is. This is the approach I've chosen. If the memory card is removed (which is the most probable cause for not being able to read a file from it) the user is alerted that file loading failed.

Another problem with the external memory is that, unlike the internal, it is not private to the application. External memory might be accessed by any application on the device. For example, the gallery application on some Android devices search the entire memory card for pictures to display in the gallery, and this would include the pictures from picture notes from my application. There are solutions for restricting access to files on the external memory for other applications but I did not see this as a big problem, so I left this for future improvements.

[5] A thumbnail is a small preview of a picture.

2.1.3 Handling suspension/termination

Android has built in mechanism to handle the fact that the system might need to suspend or terminate a process at any time. When the operating system is about to kill a process, the method `onSaveInstanceState(Bundle outState)` is run. In the variable `outState` small amounts of data may be stored. In my application, I only need to consider data loss when editing text. Capturing images and recording audio is done by calling default components which handle the potential data loss by themselves. Besides, text notes are usually the ones requiring most time to make, so this is the part where preventing data loss is the most important.

In the code for the text editor I have a variable called `rowId`. This represents the row in the database where the currently edited node is located. If it is a new note (rather than an existing one being edited) `rowId` is null. I wrote a function called `saveState()` which saves the title and content field to a note in the database. If `rowId` is null the data is saved to a new note and `rowId` is set to the id of that note. Otherwise the note with id = `rowId` is updated.

Either way, after calling `saveState()` `rowId` contains the database id of the newly saved note. `rowId` is now all that's needed to resume the program without data loss, so I save it to `outState`. This is the code called when the application is about to be killed:

```
protected void onSaveInstanceState(Bundle outState){
    super.onSaveInstanceState(outState);

    /* Saves the title and content to the note with id = rowId.
     * If rowId = null a new note is created in the database and
     * rowId is set to the id of the newly created note.
     */
    saveState();

    outState.putSerializable(SparkDBA.T_KEY_ROWID, rowId);
}
```

Once the application is restarted, `rowId` can be read from a Bundle passed to the initialisation function `onCreate(Bundle savedInstanceState)`. Once `rowId` is read, the text fields can be repopulated and the text editor is back to the same state it was in before getting killed.

2.2 Synchronisation

The synchronisation is an integral part of the mobile applications functionality. It is the link between the mobile application and the web application but since the development of the web application was halted, I had to invent the protocol for communication with the web server myself. I wanted this protocol to be as close to the real thing as possible so that when Creado resumed the development of the web application they could use my protocol, or at least spend minimal time adjusting it.

2.2.1 Local versus global identifiers

Before going further, let me first explain the difference between the local and global identifier of a note. Notes are locally stored in a database, and a numerical identifier is typically used as the unique primary key. The local identifier is used everywhere in the mobile application, but outside the mobile application, it does not hold any meaning. The same global note may even have different local identifiers if loaded on different mobile devices. This of course means that the local identifier cannot be used as a global identifier.

The global identifier is the unique identifier in the web applications database. This database contains all existing notes on Creado, so this is truly a unique identifier. Since it is unique to the global database, only the global database may create it. As the mobile application is able to create notes without being in contact with the global database (creating notes in offline mode is one of the requirements of the mobile application), the global identifier cannot be used as local identifier.

Currently there is actually no need for the mobile application to know the global identifier. This is because the communication is one-way. When the web application receives a note from the mobile application, it knows that this note was created on the mobile application and that it should be inserted as a new item into the global database. In the future however, this is will not always be the case. A note may be created on the web application, uploaded to and edited on the mobile application and then returned to the web application. In this case, the web application must be able to identify this note as the same it sent before. The web application should update its existing copy of the note, rather than creating a new one and keeping the old. For this to work, the note needs a global identifier even on the mobile application. Since I wanted to make my protocol as close to a real protocol as possible, this was something I had to consider.

Because of this, there are two parts of the synchronisation. Firstly, the obvious one, to upload the notes from the mobile application to the web application. Because of the future need for the mobile application to know the global identifier, there is also the second part, where the web application responds that notes have been successfully received and lets the mobile application know about their new global identifiers.

2.2.2 Protocol

When we initially discussed the protocol with the developers of the web application we decided that we should use JSON[6] for all communication and this is what I used for my protocol. The protocol is a pretty straight forward wrapping of the data into a JSON array. Since JSON sends data in plain text, the binary data of the pictures and audio has to be converted first.

[6] JavaScript Object Notation, a light weight alternative to XML. All data is sent as plain text.

I chose to use Base64 encoding[7] to encode the data to text suitable for JSON. Base64 has a reasonable overhead (slightly above 33%[8]) and is widely supported. Android also has built-in support for it, making the encoding process very easy.

The idea is that when the web application receives and successfully decodes a note from the mobile application, this note is inserted into the global database. As this happens, the note gets a global identifier. To let the mobile application know this global identifier, the web application replies by a JSON array containing tuples of `{localIdentifier, globalIdentifier}` for each element that was successfully received.

The mobile application receives this reply. For each tuple `{localIdentifier, globalIdentifier}` in the received JSON array, the note with identifier `id == localIdentifier` gets its `globalId` set to `globalIdentifier`. They also get their `synced` field set to `true`.

Below is a (formatted) sample JSON object created when uploading a text note and the reply sent by the web application:

```
Mobile application sending a note to the web application:

[
   {
      "id":"1",
      "title":"Pancakes",
      "content":"2 eggs 2 dl wheat flour 4 dl milk a little salt",
      "type":"1",
      "createDate":"1312977857",
      "modifyDate":"1312991350",
      "globalId":"null"
   }
]


Web application replying to the mobile application:

[
   {
      "localIdentifier":"1",
      "globalIdentifier":TE9MQ0FUUzwz"
   }
]
```

This hopefully explains how the protocol works. What is also visible is that the title and the content is sent in plain text, which is not really a good idea. I will discuss this further in the future improvements section.

Since there was no real interface to test against, I had to be inventive. The fact that JSON is both human-readable and widely supported made the test process much easier, as all I needed to do was to set up a server to receive the

[7] Base64 is an encoding scheme where binary data is converted into a subset of the ASCII characters so that the characters used are only ones that are easy to unambiguously transmit. This means omitting blank spaces, commas, quotation marks and using alphanumerical characters.

[8] Since 6 bits are used per byte, and blank spaces are ignored.

input, write it to a file and then use online tools for validation and manual inspection to see that data was successfully transmitted. For testing the Base64 I could easily test the decoding using online tools to see that the correct data was sent. Knowing that the received data was valid JSON data containing the correct binary data, I knew that the sending worked as intended.

2.2.3 Local operations

Now that I have explained how the data is sent, I am going to explain what happens locally in the mobile application. Each time the application is started or a note is created/edited the synchronisation method is called. This method first queries the database to find out whether there are any unsynchronised notes in the database. Secondly it asks the device if it has a functioning internet connection. If both those conditions are met, the device needs to and is able to attempt to synchronise. This launches the synchronisation Service.

Synchronisation requires database queries, conversions of data and network operations, actions that are all potentially work intensive and time consuming. As such, I want to make sure that these operations do not block the user interface, and that is why I use a Service, launched in a separate thread. The synchronisation Service begins by extracting all unsynchronised notes from the database. Each such note is then converted to a JSON object, which includes encoding data in Base64 for the binary data. All objects are stored into a JSON array, which is later sent to the web server. The server reply (in my case just pre-defined to return a reply for local identifier 1 and 2, for testing reasons) is stored into a JSON array, which is then iterated over to update the global identifier and synced fields of the notes for which a reply was received. Once this is done, the database is once again queried to see if it has any unsynchronised notes, and if it has not, the synchronisation is reported as successful.

2.2.4 Duplication problem

A potential problem with this protocol could occur if the reply from the web application gets lost. The web application (according to my protocol) is supposed to treat any note with a global identifier equal to null as a new note. Let us look at an example where that could go wrong:

1. Mobile application sends note N with local identifier L and global identifier = null.
2. Web application receives N.
3. As global identifier of N is null, a new note is created in the global database. The new note in the global database gets a global identifier G.
4. Web application replies with reply R containing L and G.
5. R gets lost/corrupted in transfer.

6. Mobile application does not receive a reply and therefore considers N to be unsynchronised and its global identifier equal to null.
7. Mobile application is relaunched or some note is being created/edited.
8. Synchronisation Service is called on the mobile device.
9. N is among the unsynchronised notes and will therefore be sent having null as its global identifier.
10. N is received by the web application and since its global identifier equals null, a duplicate note is created in the global database.

One way to solve this problem would be to assign a unique identifier for each mobile device. The combination of that identifier and the local identifier could be used to uniquely identify notes. This would enable the web application to detect that it had already created the note that it was about to make a duplicate from. Another solution would be to not create the global note unless the reply is successfully sent and received. Since my knowledge of network protocols is quite limited, I will simply mention this problem, rather than try to solve it.

2.3 Interface design

Designing the interface for an Android device is a little different from designing interfaces for desktop applications. The user interacts with the device primarily by the touch screen. Other than the touch screen, all android devices have a hardware button for going back. If you are in an Activity A and from there you launch an Activity B, pressing the back button while in B will take you back to A. If you press it again while being in A, you return to wherever A was launched from. This is not something you as a developer have control over.

There are also conventions for how things are done in an Android application. For example if items are listed, users can expect that long pressing (pressing and holding down for a few seconds) one of the items will show a context menu where the item can be modified (deleted, renamed, etc.). Another example is that Android users expect that using the back button don't cause them to lose data.

Another problem when you design something for an Android device is that an Android device can be a lot of different things. An android device is simply a device running a specific operating system. The most relevant aspect that I had to consider for the design was the screen size. A design should look good on everything from the 240 x 320 pixel screen of the Sony Ericsson Xperia X10 mini to the 540 x 960 pixels of the Motorola Droid X2.

2.3.1 The design process

I created the design of the application together with the people from Creado and Emilio Nyaray, the developer of their iPhone application. We wanted the mobile applications to look and function roughly the same, while still respecting

device specific conventions. Furthermore we wanted the mobile application to resemble the web application, or at least what the web application looked like when we started the design process.

2.3.2 Description of the user interface

In the web application there was a grid of notes sorted first left to right, then top to bottom based on most recently edited. In the place where the first note would be, a button for adding new notes was placed. This design was mimicked from the web application. A good thing about using a grid is that it is an efficient way to adapt to different screen sizes. On really small screens, there would be only two columns, while the really large screens can take advantage of their size and display 4-5 columns. The grid layout also works well in landscape mode (when the phone is rotated 90°) as the fewer visible rows are compensated by more visible columns, unlike a layout where each node takes up a separate row, which in landscape mode would just mean fewer but wider rows.

This grid is where the application starts. Pressing any of the existing notes launches the Activity to view/edit that kind of note. For text notes, this means launching the text editor. For picture notes a very simple Activity that simply displays the picture and its title is launched. For audio notes, there is no available Activity, simply because I did not have time to make one. The important task for this application was to be able to save ideas and upload them, so I felt that reviewing existing ones could be added later.

When the button for adding a new note is pressed, a menu is shown over the grid where the user may pick what type of note to create: Text, Picture or Audio). Pressing Text launches the note editing Activity, pressing picture shows two new menu options, Camera or Gallery. For each of the options Camera, Gallery and Audio I specify an Intent and launch a built in application application that matches it.

2.3.3 Design choices

The big reason for using Intents is that writing my own applications for browsing galleries, taking pictures and recording audio would be very time consuming. The only benefit of doing so would be that I had direct control over exactly how they looked and worked. Using Intents I instead uses existing applications that the user is likely to be familiar with. These applications are either supplied by the phones manufacturer or installed by the user. The ones supplied by the phones manufacturer are most likely of higher quality than I would be able to write as a part of this project. The user installed applications are probably even better, since the user got through the trouble of installing them instead of using the ones provided by the phone manufacturer. Either way, most likely better than if I wrote them myself. The only real downside of using Intents is cosmetic: not everything looks like a part of the same application. This is something that could be added in later versions.

As mentioned, one of the conventions for Android applications is that you

are supposed to be able to press the back-button without losing data. Since I use Intents for creating all but text notes, the text editor is the only place where I need to worry about this. My solution to this is that when the user presses the back button I simply save the note, overwriting the old if the note already existed or creating a new note if it did not. Another convention I adhere to is the convention of long pressing listed items. If the user long presses a note in the grid view she gets the option to delete the note.

3 Results

The result of this project is a working Android application which provides the basic functionality of creating and uploading notes in text, picture and audio format. The application is not tested on any real users, nor is it bug free, optimised or tested on a lot of different devices. There is a lot of work left to do, but the code is well documented and structured in a way that should make it easy to continue developing the application. The source code is commented using Javadoc.

In this section I will show how the application looks. I will also describe where whoever keeps developing the application should continue my work.

3.1 Screenshots

This is what the application looks like:

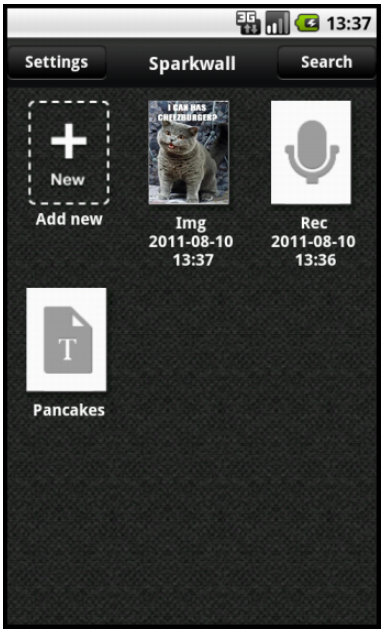

Figure 1: In this view existing notes are displayed and the user may access the menu for creating new notes. The order of the notes are based on when they were last created/modified. The picture note was the most recently modified, while the text note was the least recently modified. Pressing "Add new" takes us to Figure 2

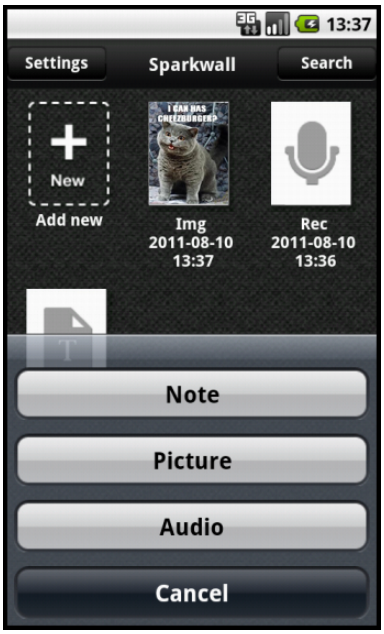

Figure 2: The buttons functions as follows: "Note" launches the text editor with a new empty note (Figure 3, except the fields are empty). "Picture" accesses the menu where the gallery or camera Intents are launched from. "Audio" launches the audio recording Intent. Cancel simply removes the menu and returns the user to the main view.

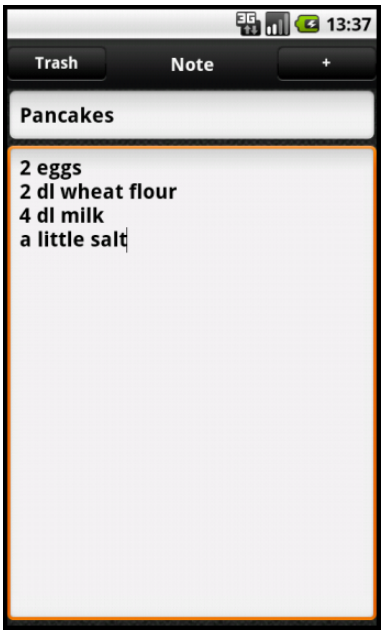

Figure 3: In this Activity text notes are created, edited or deleted. The title field is optional, and if it is left blank, a title will be computed from the first few letters of the content.

3.2 Future improvements

In this section I will describe what I think should be added to the application in the future. These are both things I did not have time to complete myself and things I was not able to complete because of the absence of a working interface to the web application. Many of the things mentioned here are also available as TODO-comments in the code. TODO-comments are supported by Eclipse, the development environment recommended for Android by Google.

3.2.1 Logging in

Since there is currently no interface between the web application and the mobile application, I had no way of implementing any scheme for logging in. This is a task that should be done as soon as the interface between the web application and the mobile application is done.

3.2.2 Thumbnail data moved to SD card

As I mentioned before, my initial plan was to store all data in the database on the internal memory. With the database full of large files, having the thumbnails there as well wouldn't make much difference. Now the thumbnails are the only

large files in the database, and as such their impact on the overall database size is significant. Possibly this does not have any significant impact on the application, but whoever continues my work should at least look into the possibility of moving the thumbnail data to the SD card.

3.2.3 Protecting data on SD card

As it is now, the data is stored fully visible on the SD card. One consequence of this is that on many phones this data is visible through the gallery view. Since the data can come from the internal gallery on the phone, this will just create duplicates in the gallery, which will probably be annoying to the user. A good change would probably be putting the data in a hidden folder. As for saving the data from unauthorised access, this seems a little unnecessary to me, but at least I want it to be known to whoever will keep developing the application that data located on the SD card is available to any application for reading and/or modifying.

3.2.4 Encode title/content before transfer

The title and content field is now sent in plain text. The plain text is simply inserted between quotation marks in the JSON string. If the user would have quotation marks in either the title or the context, this would cause the JSON to malfunction. It also causes some problems with white spaces and line breaks. This is simply something I forgot about and did not have time to solve myself. The solution is simple. Just use the same Base64 encoding as was used for the binary data of the pictures and audio.

3.2.5 Adding audio playing Activity

Audio notes cannot currently be reviewed. Before the application is ready to be released to the public, an Activity for listening to recorded Audio should be written. As it is now, picture and text notes can be reviewed while audio cannot. Users will probably expect all types of notes to have similar functionality, so implementing this should be a priority task.

3.2.6 Replace built in components with self made

This task is, in my opinion, not the highest prioritised one. The application functions and as mentioned before, the benefits of implementing this is mainly cosmetic. However, for the application to look its best it should probably have its own self made Activities for recording audio, browsing the gallery and taking pictures eventually.

3.2.7 Serialise the synchronisation Service

Currently the synchronisation service works as follows: When it is called it queries the database if there are unsynchronised notes. If there are, it launches

a new thread which grabs them, uploads them, gets a response and (assuming the synchronisation was successful) once the response is parsed updates the database. This means that from the time where the thread is started to where it has done all its work the state of the database is not altered.

If a synchronisation attempt is made while a synchronisation thread is already running trying to update the notes N but has not reached the point where it updates the database, the new thread would query the database and find that the the notes N are not updated and would attempt to update them at the same time as the initial thread does. This is an unwanted behaviour and a solution would be to only allow one synchronisation thread to run at a time.

3.3 Move code away from `onCreate()`

Until the very last day of implementing it was my firm belief that the onCreate method was the Activity equivalent of a main function. This belief came from the fact that basically every single Android tutorial I have read so far used it in that way. It however turned out that this is not the case. The onCreate method is called when an Activity is launched, but it is also called when the phone is rotated.

This turned out to cause a rather amusing bug where you can crash the application by simply tilting the phone back and forth. This is because I call the synchronisation service in my initialisation code for the main view. Since the synchronisation Service is currently allowed to run in parallel (which should be corrected) it launches a new thread every time the the phone is tilted until the max number of allowed threads is exceeded. Needless to say, fixing this problem has very high priority.

4 Conclusion

Before starting this project I had never developed anything significant in Java and I did not know anything about smartphones. Because of this a large part, possibly the majority, of the project was about learning how Java and Android applications work. As a result, the application probably contains some poor design choices due to my lacking experience in designing for the Android platform.

While there are some issues with the current code, I still believe that it is a good foundation for building a good functioning application that could be released. As a learning experience it has been very interesting. Some parts, for example the main view for displaying existing notes, took considerably more time than planned while adding support for audio recording (which we had planned to take a week) took about 2 hours.

One of the biggest challenges was understanding the Android devices. The varying screen sizes and different varying input devices was a lot to consider. I had to make some assumptions: mainly that a external SD card existed and that some kind of gallery/file browsing application and a camera was available. This is true for the vast majority of Android devices, but having to consider things like how I would handle the situation if no external storage was available made me understand what a difficult task it is to write good Android applications.

Another big challenge was understanding the structure of an Android application. How to link the database to the main grid view and how to store the resulting data of calling the camera Intent are two examples of tasks where I had to spend considerable time just learning about how things fit together. Understanding good practices comes with experience, but I tried to learn as many of them as possible so that the application would be as easy to extend as possible. One example of such a good practice is that all text strings in the application are located in a separate XML-file. This is a good way to do it as it makes translating the application to different languages easy.

All in all I am quite satisfied with the results. The application is not very advanced, but that is expected since I knew nothing about Android before starting this project. While it is not advanced, it still does what it should (except for a few bugs mentioned earlier) and I have learned a lot about Java, Android and programming in general.