Mutext: The Notetaking App for Songwriters

Jason Bromfield - 100934833
Supervised by Dr. Tony White
COMP 4905 Honours Project
15 December 2017
Abstract

A lot goes into creating a good mobile app. Even the simplest app can be tedious to develop and test to a sufficient capacity. With simplicity comes room for imagination, building upon a basic design to express creativity, as well as creating new and useful features. One of the ways people express themselves and capture imagination is by using music. Through combining my knowledge of mobile application development as well my interest in music, I have created a notetaking app designed with songwriters in mind. I named this app Mutext as a combination of music and text; it coincidentally sounds like the word mutex, but is not related to it.
Acknowledgement

I would like to thank Dr. Tony White for being my supervisor for this project. His guidance and responsiveness to my concerns regarding this project allowed me to have fun working on this project as well as develop new skills that will assist me in my future career. I would also like to thank stackoverflow.com user Jeremy Lee for giving me a suggestion to tackle one of the issues I was facing during this project.
# Table of Contents

Abstract  
Acknowledgement  
Table of Contents  
Background Information  
Development Environment  
Motivation  
Why Android?  
Design  
Issues  
Future Work  
Conclusion  
References  
Appendix
Background Information

MIDI

MIDI (Musical Instrument Digital Interface) is a technology that allows musical information to be represented as data for use in software.

Android Development

Android is a mobile operating system developed by Google. It is the most popular mobile operating system in the world with approximately 88% market share. Android Studio is an IDE (Independent Development Environment) that specialises in the development of Android applications. It is based off the IntelliJ IDEA IDE by JetBrains. It is the official tool for Android application development.

Gradle

Gradle is a build management tool used by Android Studio projects to manage an app’s dependencies.

Kotlin

Kotlin is a programming language developed by JetBrains; it is the Android equivalent of Swift for iOS. Kotlin has a less verbose syntax than Java and it is designed to work with Java despite not having compatible syntax. The purpose of Kotlin in the context of Android programming is to provide a more concise, easy to use alternative to Java. Kotlin became fully supported in Android development with the release of Android Studio version 3.0.
React

React is a library that allows Android and iOS applications to be created using Javascript.

Development Environment

- Android Studio version 3.0.1
- Android device or emulator running API level 21 (5.0 Lollipop) or higher

Motivation

My motivation for creating this app for my honours project is to address a need as well as learn something useful for my career. As a student in the Software Engineering stream, I learn a lot about creating reliable software built to industry standards and following established methodologies. What I do not learn in my stream is how to develop mobile applications. Mobile devices are now ubiquitous in today’s society, so it is very useful to be able to develop mobile applications. I also have an interest in music and do song writing in my free time. I currently use the Google Keep app to write lyrics, however, Google Keep was meant more for general notetaking, lists, and reminders than writing lyrics. These reasons inspired me to create Mutext; it is designed to be an application that combines the best features of Google Keep along with features that make it useful to songwriters.
Why Android?

I chose Android as the operating system Mutext runs on for a couple of reasons. Android was the first mobile platform I learned to program for. Out of my own interest in learning something useful and already being familiar with using Java, I taught myself how to develop Android apps. The other reason is because I own an Android device; from my experience, it is a lot more useful to test apps on a real device rather than an emulator because it uses its own processing power, and it is easier to real-world test because it does not have to be attached to the development computer.

Design

In Android, there are objects that represent individual views called activities. Mutext uses four activities: LoginActivity, which represents the app’s login screen; MainActivity, which is the main screen of the app, containing the list of the user’s notes; NoteActivity, which is the screen shown to a user while they are editing a note; and RhymeActivity, which is an activity for showing words that rhyme with a selected word (more on that later).

```xml
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
```

Figure 1 Permissions used by Mutext
Permissions

Mutext only uses the permissions it needs, which are just for checking if there is a network connection and for connecting to the internet. Doing this minimizes the amount of access the app has to a user’s personal information.

App Size

A good thing about Mutext is that it takes up a negligible amount of space on the device it is installed on. This is because I import a minimum amount of third party libraries and try to implement the majority of Mutext’s features using my own code and the features supplied by the Android SDK.

Other Design Decisions

One of the components of Material Design is a Floating Action Button. A floating action button represents the primary action in an application. Initially, I was planning on using this to create a button for creating new notes. I did not end up using this in Mutext because it covered elements in the recycler view and there was space for an “Add” button in the toolbar at the top of the screen. To be noted as well, the Google Keep app does not use a floating action button and instead has a bar at the bottom of the screen for creating new notes. I decided not to use the collapsing toolbar layout in the app because I wanted to implement a feature that allowed the user to tap the app bar and be taken to the top of whatever screen they were on.
Features

Signing In

When a user opens the app, they are greeted by a login screen which allows them to sign in/create an account and have their notes backed up. They can also choose to skip this screen and go straight to using the app; the option to sign in is still available from the app’s main page. If a user chooses to not sign in, their notes will only be saved to their device via the app’s SQLite database.

Taking Notes

The core feature of the app, like all other notetaking apps, is to create and edit notes. Notes are implemented in the code as objects. Every note has a title, content, a date created, and a date modified. When a new note is created, its fields are set to their default values. The default value for a note’s title is “untitled”; the default for its content is an empty string, and the default value for a note’s date created and date modified are the current date. When a user wishes to create a new note, they are taken to an empty NoteActivity where they can type text into text fields for the title and the content of the new note. When a user wishes to edit a note, they must tap on the desired card in the list of notes in MainActivity. When they do this, the note associated with the tapped card gets packaged into an intent as a Parcelable object; this intent is then used to start the NoteActivity. Within the NoteActivity, the note object is removed from the intent and used to populate the title and content text fields of NoteActivity.
Saving Notes

The saving of notes is done in two ways: through writing to the app’s SQLite database and through writing to the online Firebase Database. When a user has finished editing a note and navigates back to the main screen, the `onBackPressed` method of NoteActivity is called. In this method, it calls the code for saving notes to each database.

Deleting Notes

Users can delete notes by selecting them from the main activity via a long press, or by clearing its content and title. If a note, upon the completion of its editing by a user, is un titled and has no content, it is automatically deleted. If a user has deleted one or more notes, they are shown a snack bar for a short period of time. A snack bar is a short message displayed after an action has been performed by a user for a short period of time; it may or may not provide an action that the user can perform. The documentation on snack bars states that their messages should be short and relate to the operation performed, and they should not contain icons. I use a snack bar to give the user the option to undo their deletion of notes. Doing this allows and encourages the user to make mistakes, because they can still recover from them.

Reading Notes Aloud
A simple yet useful feature I implemented in Mutext is the ability to have a user’s text-to-speech system read the contents of the note for them. One problem I have noticed with this feature is that the voice does not easily recognize the context of a word; it may mispronounce some words because they are homographs. Some examples of this are the words read, read (past tense), bow, and bow (tie).

Figure 2 shows how chord insertion could work

Chord Insertion

A feature that I did not have enough time to implement was the feature that allowed a user to insert chords above selected text. This feature would allow a user to select a body of text, choose a chord, and have the name of that chord shown above the selected text, similar to how lyrics are written on paper. The plan was to achieve this in three ways; via manual input, via MIDI input, and via raw audio input.

Rhyming Selected Words
This feature allows the user to select a word and get rhymes for it. This feature had the fortunate affect of being usable outside of the application because of how it was implemented.

**Accessing Notes from Different Devices**

This feature lets users backup and access their notes from different devices running the app; it is implemented through the use of Firebase. Users sign into the app with their Google account, which then creates a profile for them within Firebase. I did it this way because it was easier and more secure; I would not have to worry about encryption of passwords.

When a user saves a note, it is saved to their device as well as to the online Firebase database. If the same user signs into another device running the app, they will see the updated note. When a note is updated from another device, the SyncService updates the other devices’ databases from the background.

Testing this feature was not easy because it required that I use my own device as well as an emulator. I had to account for the same note being edited at the same time from both devices. The current behaviour of this is to keep the most recent edit.

*Figure 3 shows the original text*
Figure 4 shows the highlighted text.

Figure 5 Genius.com’s Check that Rhyme video series

Highlighting

The feature that inspired me to develop this app was the highlighting of rhyming syllables feature. I got the idea from Genius.com’s video series *Check That Rhyme*, where they highlight the syllables that rhyme in the lyrics of a song (See Figure).
This feature, which is the most useful to me, took the most time to implement.

In keeping with using the latest technology, to make requests to the APIs I used for highlighting syllables, I used Volley rather than the combination of AsyncTask and HttpURLConnection. I am used to doing this the latter way but decided to learn to use Volley because it's easier to set up and is recommended by Android. The drawback to using Volley is that it is an external component that needs to be installed via Gradle. The highlight feature makes use of a class called SpannableString. This class allows a text field to show markup data such as changes in text colour, typeface, and background colour among other things.

Issues

I was not able to implement some of the features that I had user stories for because I ran out of time.

The biggest thing missing from the application is the ability to insert chords above the lyrics. I had planned to achieve this using React as per the suggestion given to me by my supervisor, however, I found it difficult to get React to achieve the behaviour I needed it to. I wanted to create a component made up of an editable TextInput component, and a Text component that could not be directly edited by the user. The view would wrap together, allowing it to look like there was a TextInput in between the lines of a Text component. I ended up spending too much time trying to implement this feature but ended up giving up, deciding to work on other features and add features I thought of while developing the app. The chord feature also ties into the MIDI chord
insertion feature; since I could not implement the chord feature, I also could not implement the MIDI feature.

A problem I encountered while working on this project was in implementing the highlighting feature. The algorithm to do the highlighting is relatively slow because of its reliance on the Wordnik API and the Rhymebrain API. The feature is restrained by the fact that the Rhymebrain API has a 350 request per hour limit. A reasonably sized note could easily surpass this limit, which would cause the algorithm to fail. The method I devised to improve the running time of the algorithm was to store the rhymes of words in firebase. I did not get around to implementing this feature as I ran out of time.

Future Work

I plan on implementing all the features I have in my user stories as well as finding a way to improve the highlighting algorithm.

Conclusion

I was disappointed that I could not implement all the features that I wanted to, however, this gives me an opportunity to continue working on this application outside of this project. I had fun working on it despite a few times it became frustrating. I hope what I have learned from working on this project will be useful to me in the future. If anything, I am more inspired now to build more Android apps in Kotlin and experiment with the latest technologies of mobile development.
References

https://material.io/guidelines/

https://facebook.github.io/react-native/

https://kotlinlang.org/


https://material.io/components/android/catalog/snackbar/

https://material.io/guidelines/components/snackbars-toasts.html

https://en.wikipedia.org/wiki/Kotlin_(programming_language)

https://en.wikipedia.org/wiki/Java_virtual_machine

https://www.sqlite.org/

https://www.tutorialspoint.com/android/android_text_to_speech.htm
Appendix

User Stories

The following are the user stories I used to prioritise feature development in Mutext.

Some of these user stories were not finished at the time this project was submitted.

1. As a user, I would like to see a list of my notes.
2. As a user, I would like to be able to edit notes.
3. As a user, I would like to be able to access my notes offline.
4. As a user, I would like to be able to delete notes. (Epic)
5. As a user, I would like to be able to quickly scroll through notes.
6. As a user, I would like to be able to save notes for later.
7. As a user, I would like to search for notes.
8. As a user, I would like to be able to add chords over words.
9. As a user, I would like to be able to cancel a note entry.
10. As a user, I would like to have empty notes discarded.
11. As a user, I would like to delete my account.
12. As a user, I would like to be able to recover notes I accidentally delete.
13. As a user, I would like to have my notes automatically saved.
14. As a user, I would like to be able to delete all my notes at once.
15. As a user, I would like to group my notes together.
16. As a user, I would like to see notes I have deleted in the past.
17. As a user, I would like to have a night mode for editing notes in low light environments.
18. As a user, I would like to be able to export my notes.

19. As a user, I would like to be able to choose a colour for my notes.