MailKey

(SoftKeyboard with app-launching capabilities)

COMP 3000

Fall 2016

Dr. Michel Barbeau

Rob Shrives - 100842848

Ngo, Pham Minh Thong - 100848224

1 Introduction

1.1 Context

Cellular devices are generally no longer produced with physical keyboards which has created a demand for on-screen keyboards to be implemented in their place. An Input Method Editor (IME) is a program which enables users to send touchscreen gestures or keystrokes as input data. IME's provide the foundation and framework for the development of such things as software keyboards.

Some major benefits that on-screen keyboards offer is that they can be updated and customized to fit a user's needs, whereas once a physical keyboard has been created it cannot no longer be altered. Developing a software keyboard IME with previously non-existing behaviours was the main focus of this project.

1.2 Problem Statement

The objective of this project was to create a custom IME, in Android, which would allow users to launch an email client directly from their keyboard. Our intention was to remove some of the time consuming effort that is taken by the user in order to navigate between different applications. This problem was relevant to us because, in our busy lives as students, communication and efficiency are extremely key to being successful.

1.3 Result

Originally our project was centered around the idea of creating a software keyboard for Android that had predictive text-emoji behaviours, or as we referred to it Text2Emoji. The functionality of the text-emoji behaviours were supposed to similar to what is currently found in Apple's iOS 10[1] keyboard. Unfortunately as junior programmers/students we discovered some of the topics related to this subject matter were out of our scope of comprehension and it was best to move in another direction.

That is when our group came up with the idea for MailKey, which we were able to fully design and implement. We feel that our final product is visually pleasing, easy to use, and most importantly provides the user with functionality that was not currently available before our project was developed. Overall we are very pleased with our results and we are considering to continue the development of MailKey after this course's completion.

1.4 Outline

The remainder of this report is outlined as follows: Section 2 discusses the background information relevant to this report, including justifications on Android, IME in general and our original idea Text2Emoji specifically. A description of how we planned on accomplishing our original idea and the reason we switched from Text2Emoji to MailKey. Section 3 describes in detail the result of our project, a brief description of MailKey's usability plus conveniences, and also how we created and tested it. The result is evaluated in Section 4 in which we've tried to be honest, self-critical, and objective about our work. Section 5 concludes with a summary of accomplishments and a brief overview of future features and goals that have not yet been accomplished. Next, section 6 is regarded to each team member contribution. Over the course of this project, our team learnt and used information from all the cited sources in section 7. Team member frustrations and thoughts while making this project will be shared in section 8.

2 Background Information

2.1 What is an IME?

As previously stated in section 1.1 an Input Method Editor[2] provides the framework for developing programs which handle user input. This input is generally in the form of on-screen gestures or keystrokes. Since IME's are responsible for the handling these types of input they make an ideal foundation for the creation of software keyboards. This mainly because of their structure but also because they provide a great amount of flexibility for customization. Below is a high level diagram of how user input is handled in an IME:

2.2 Why Android?

We decided to develop our project for the Android mobile platform because our original idea, Text2Emoji, has only been implemented in Apple's iOS 10. After a great deal of searching we were unable to find any similar solutions for software keyboards on the android platform. This presented us with the perfect opportunity to augment something onto an existing operating system component that had not been implemented yet. Although our initial idea fell through we had already created the software keyboard and decided to continue with it.

Android is an open source platform that offers a great wealth of existing resources and examples[3]. This paired with the fact that Thong is already an Android user made the decision for developing on this platform a lot easier.

2.3 How Text2Emoji would have worked

Text2Emoji was intended to work by having a list of predefined terms such as "angry" or "happy" stored in a user dictionary[4]. Once a user's input connection matched a specific term then an emoticon would have shown up in the predictive text box above the keyboard. The user would then be able to click it and it would be shown in place of what the user initially typed. Adding this functionality ended up being far more difficult than expected and we hit a roadblock that we were no longer able to pass. It was at this point that we decided to rework our project to something more manageable but also still met the project requirements.

3 Result

3.1 Project Outline

MailKey was our secondary backup plan since we started to get stuck on our original idea Text2Emoji. MailKey is an IME that has one specific button that we predefined to go directly to composing an email on Gmail app from wherever the users are on there device. Then they are returned to their point of origin after they have successfully sent an email.

3.2 Why MailKey?

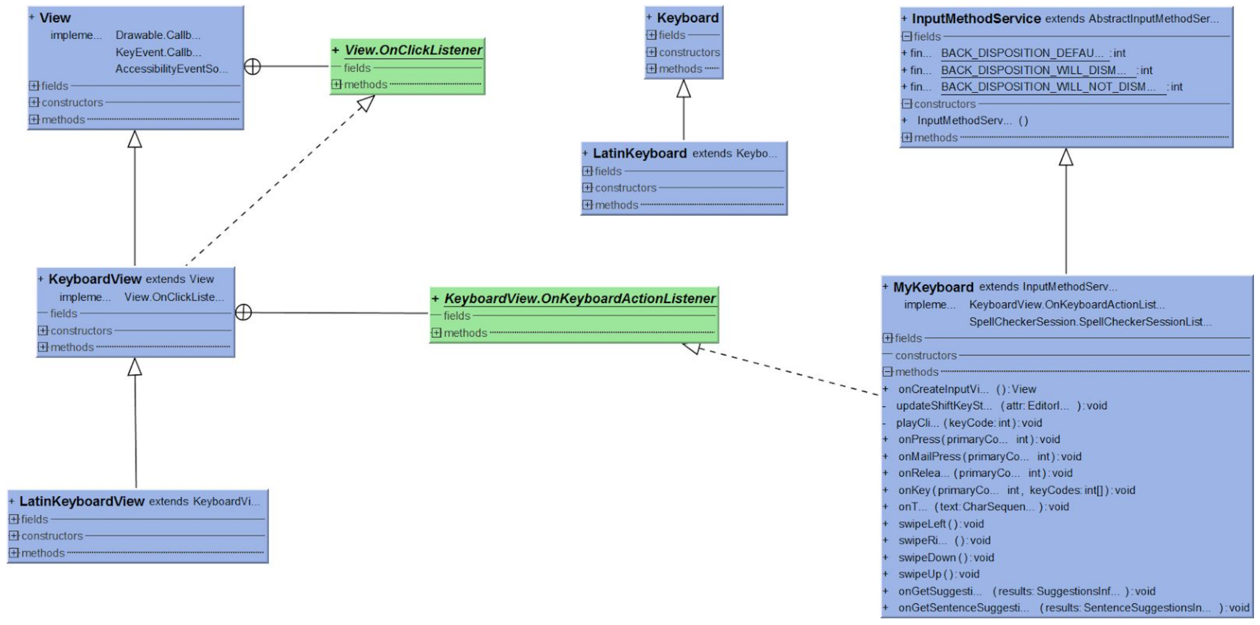
There are various reason why we made MailKey

- We successfully imported the Emojicon[5] library into Android Studio and also managed to implement CandidateView[6] box for Text2Emoji however we could not produce the suggestion like feature for emojis. Time was running out so we decided to do shift our focus towards a concept that was more achievable.
- MailKey also allowed us to use the well polished keyboard that we had already made for our previous idea so we didn’t have to start from scratch
- How this idea come to us is surprisingly related and worth mentioning. As we were talking over text messaging about getting stuck on Text2Emoji and how we should email either the TA or the professor asking if we can do something different. That’s when Rob came up with the idea to make a custom IME with a

special key that linked to an activity within a specific application (Email composition in gmail)

3.3 Technical Aspects of MailKey

We based our keyboard design off a couple of different examples we found on the internet [3][7]. In this section we will not waste time discussing how keyboards work because this information is already well known. Instead we will focus on the new feature we have implemented. For the reader's reference figure 1 below is a class diagram of our software keyboard:



+ View
impleme... Drawable.Callb... KeyEvent.Callb... AccessibilityEventSo...
fields
constructors
methods
+ View.OnClickListener
fields
methods
+ Keyboard
fields
constructors
methods
+ LatinKeyboard extends Keybo...
fields
constructors
methods
+ InputMethodService extends AbstractInputMethodSer...
fields
+ fin... BACK_DISPOSITION_DEFAU... : int
+ fin... BACK_DISPOSITION_WILL_DISM... : int
+ fin... BACK_DISPOSITION_WILL_NOT_DISM... : int
constructors
+ InputMethodServ... ()
methods
+ KeyboardView extends View
impleme... View.OnClickListe...
fields
constructors
methods
+ KeyboardView.OnKeyboardActionListener
fields
methods
+ MyKeyboard extends InputMethodServ...
impleme... KeyboardView.OnKeyboardActionList... SpellCheckerSession.SpellCheckerSessionList...
fields
constructors
methods
+ onCreateInputVi... () : View
- updateShiftKeySt... (attr: EditorI...) : void
- playCli... (keyCode: int) : void
+ onPress (primaryCo... int) : void
+ onMailPress (primaryCo... int) : void
+ onRelea... (primaryCo... int) : void
+ onKey (primaryCo... int, keyCodes: int[]) : void
+ onT... (text: CharSequen...) : void
+ swipeLeft () : void
+ swipeRi... () : void
+ swipeDown () : void
+ swipeUp () : void
+ onGetSuggesti... (results: SuggestionsInf...) : void
+ onGetSentenceSuggesti... (results: SentenceSuggestionsIn...) : void
+ LatinKeyboardView extends KeyboardVi...
fields
constructors
methods

Figure 1: MailKey class diagram

Figure 1 - The class MyKeyboard extends the Input method service which enables the user input to be read and handled appropriately. The MyKeyboard class also instantiates objects of type LatinKeyboard and LatinKeyboardView which are responsible for collecting and displaying the user input.

The special functionality we have implemented to launch applications from our keyboard is very straight forward. All keys in the English language correspond with specific code representations. After we had mapped all of the keys we felt were necessary to have a fully functional English keyboard that included all letters, numbers, and symbols we were left with some unused keys. This provided us room for customization, in particular we made use of the unutilized “KEYCODE_ALT”. Immediately we saw this as an opportunity for adding in some specific behaviour or customization.

In our switch statement that reads the different codes and handles them appropriately we added a case for “KEYCODE_ALT”. In this body of logic we build an intent[8] for the mail application. This intent is then launched and the mail application is opened.

3.6 Use case scenario

In figure 2 below a typical user scenario is shown with a step-by-step description of how MailKey works:

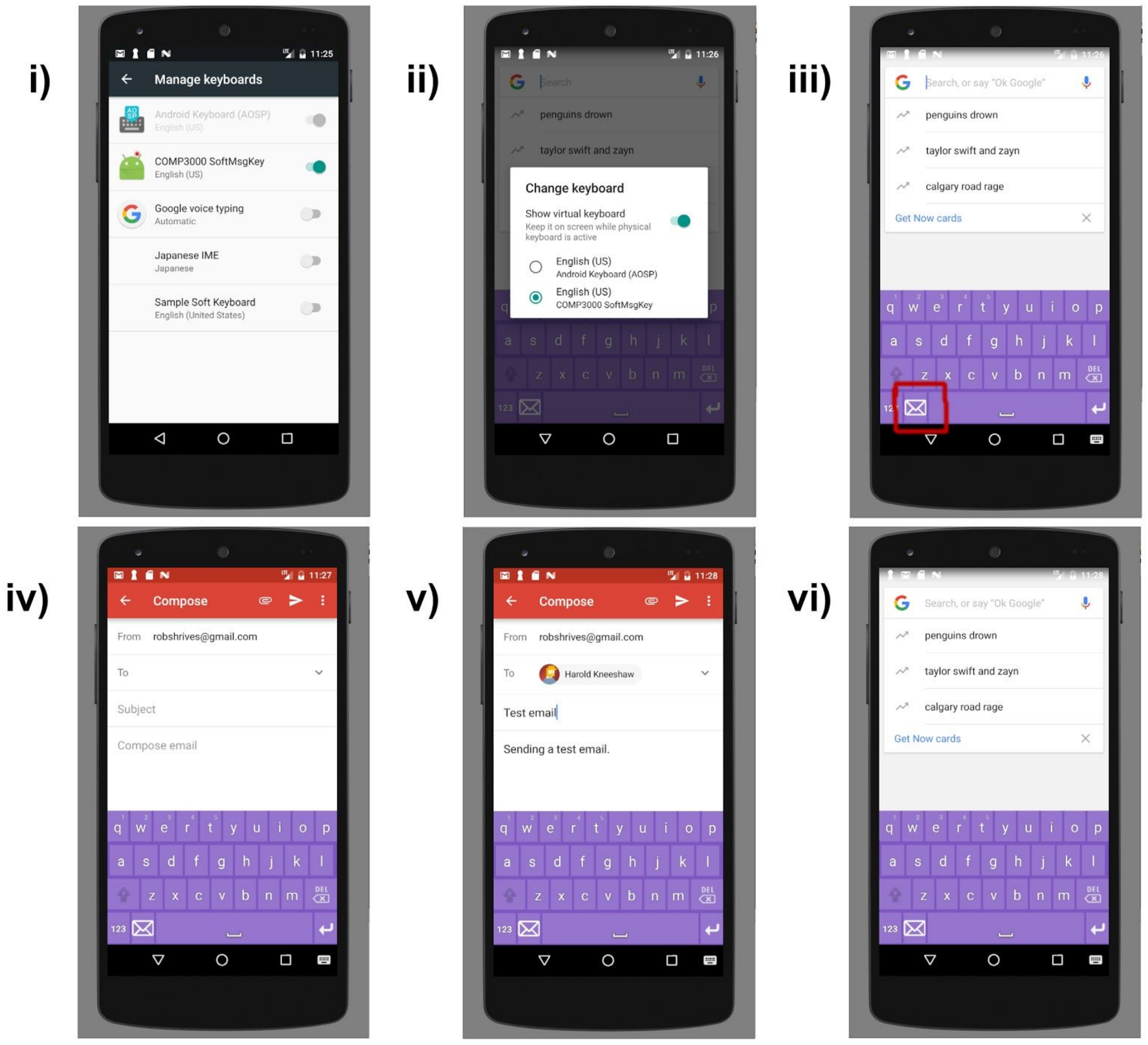


Figure 2 - i)The user enables the keyboard. ii) User selects the keyboard. iii) User presses mail button, iv) Mail app opens. v) User composes email and sends. vi) User is returned to point of origin.

3.5 Testing

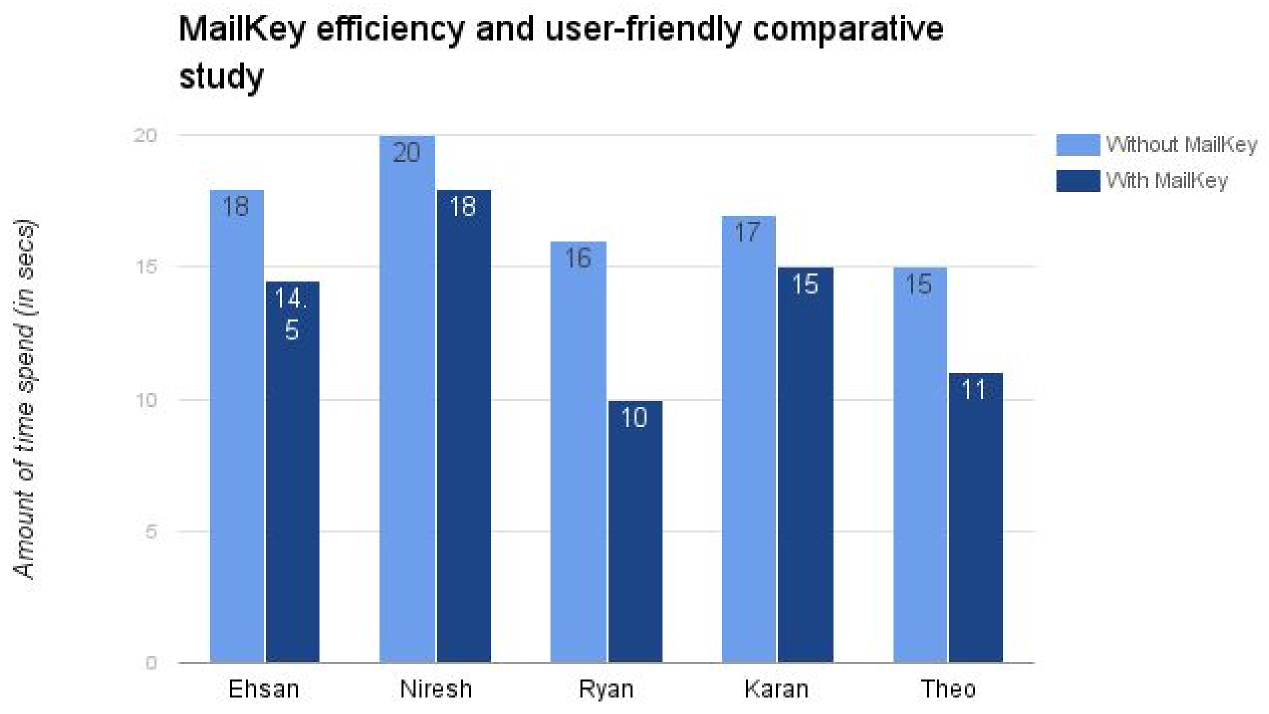
Rob did all of his testing from android studio’s emulator Nexus 5X, Android 7.1, and API level 24. Thong did all of his testing on a Xperia Z2, android 6.0, and API level 24.

We tested by simply launching the mail button from various different scenario:

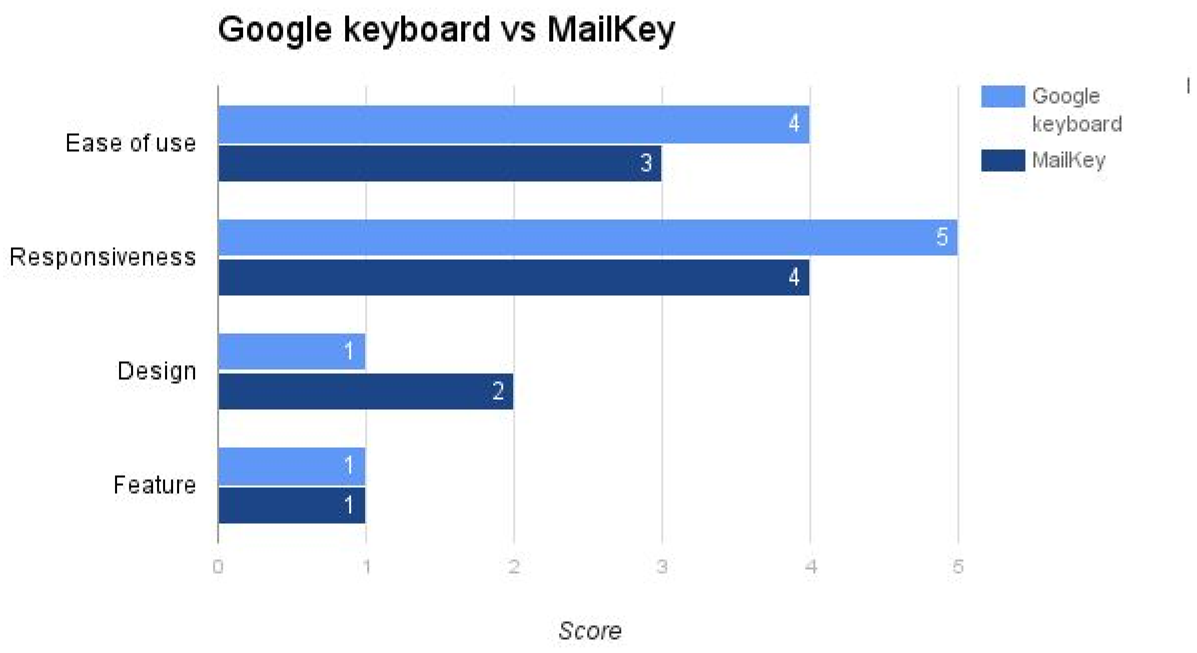
- From Messenger the full app: Successfully go back to wherever user were before
- From Messenger the chat head: MailKey is not able to reopen the correct chat head user were on.
- From Whatapp’s instant reply dialogue: Successfully send the email and go back to the instant reply dialogue
- From SnapChat review/editing screen before sending the image: Camera intent restarted after user returned from sending an email

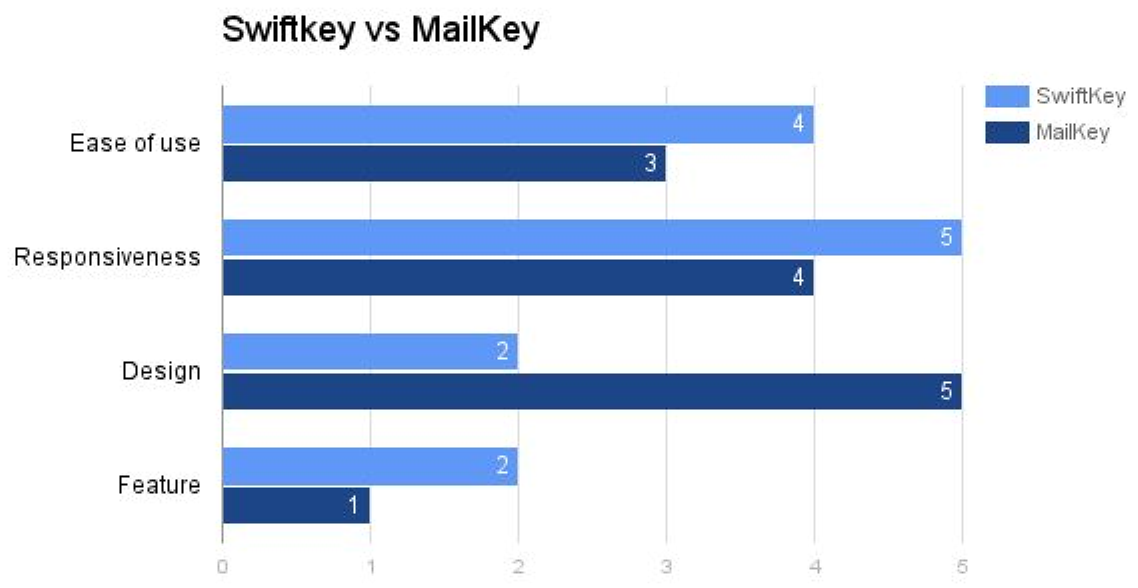
4 Evaluation

MailKey is not a complete product. It's only useful in one specific case at this time of development, composing mail. However, from a usability study we conducted involving our circle of friends in testing MailKey, showed that MailKey is easy to use. We gave every tester the same scenarios where they have to google something then email the answer to me.



We received all positive feedbacks on its simplicity and straightforwardness. Testers also expressed their interests in MailKey if the button can be created and modified by user for personal use. One tester suggested a youtube share button that once pressed will bring user to a Youtube like UI where they can search for any Youtube video and share (copied video URL to clipboard + bring user back to where they were before + paste video URL on TextView) with one tap. This is a rather interesting idea that we might use in the future if we decide to work on MailKey.





We asked 10 testers to rate our keyboard in comparison with other popular one.

5 Conclusion

The goal of this project was to create custom IME with specific behaviours that minimizes user navigation while being smooth and reliable. The focus was to accomplish this by mapping the alt key with desired functionality and choosing the most simple routes between MailKey and the launched application it connected to. A elegant design was created to meet these goals and some preliminary testing indicates that runtime for tasks can be reasonable reduced with Mailkey.

Unfortunately, issues of time management and consistent focus prevented this project from being completed as expected. Future work includes more tests on different user-cases to ensure no interruption in term of workflow and navigation between current task and launched task. The implementation of user-defined key as well as useful and interesting predefined key such as Youtube share button (mentioned in 4. Evaluation)

6 Contributions of Team Members

A. Robert Shrives developed the concept of the project and dealt with the programming/backend of the keyboard.

B. Ngo, Pham Minh Thong designed the UI/frontend and was responsible for software verification and validation (SV&V).

C. Both group members contributed equally to the content and writing of this report.

7 References

[1] Unknown Author, “iOS 10 predictive emojis”,
http://www.imore.com/how-use-emoji-and-tapbacks-imessage-ios-10

[2] Android Developers, “Input Method editor”,
https://developer.android.com/guide/topics/text/creating-input-method.html

[3] GoogleSource, “Android softkeyboard example”,
https://android.googlesource.com/platform/development/+/master/samples/SoftKeyboard?autodive=0%2F%2F%2F%2F%2F%2F

[4] Android Developers, “UserDictionary”,
https://developer.android.com/reference/android/provider/UserDictionary.html

[5] Rockerhieu, “Supporting Emojicon library”, https://github.com/rockerhieu/emojicon

[6] Android Developers, “Candidate View”,
https://developer.android.com/reference/android/inputmethodservice/InputMethodService.html#onCreateCandidatesView()

[7] Chris Black, “Custom Android keyboard“.
http://www.blackcj.com/blog/2016/03/30/building-a-custom-android-keyboard/

[8] Android Developers, “Build an intent”,
https://developer.android.com/training/basics/firstapp/starting-activity.html#BuildIntent