University of Glasgow | School of Computing Science

An end-to-end encrypted mobile chat application with accessibility features

Daniel Furnivall

School of Computing Science
Sir Alwyn Williams Building
University of Glasgow
G12 8QQ

A dissertation presented in part fulfilment of the requirements of the Degree of Master of Science at The University of Glasgow

October 15th, 2021

Contents

Chapter 1

Introduction

1.1 Aim

To develop a synchronous/asynchronous peer-to-peer instant messaging Android application that includes a central server, end-to-end encryption and various accessibility features such as text-to-speech functionality.

1.2 Objectives

1.2.1 Ability to send messages to other users

This is the basic requirement of the service, and should be prioritised above all else. Messages should be stored on the server until they are claimed by the relevant client (i.e. until they log in). Once claimed, they will be removed from the server. If the user is already logged in, the message can be sent directly to the user and deleted from the server immediately.

1.2.2 End-to-end Encryption (E2EE)

There are several algorithms that we can use to implement E2EE (e.g. Diffie-Hellman key exchange [6], PGP [7] or RSA [13]), each with their own strengths and weaknesses for this type of application. The algorithm implemented will depend on the outcome of the literature review.

1.2.3 Text-to-Speech (TTS) Functionality

TTS functionality will be developed using an online API. There are several API options to choose from, including several which are artificially generated by neural networks such as Azure TTS[18] and Google WaveNet[15].

1.2.4 Client-side message history

Unencrypted message history will be stored on the client side in an SQLite database. Message storage allows for lots of additional functionality and reduces server-side workload. For example, this allows us to keep the text to speech functionality on the client-side, as well as backup, message visualisation, analytics etc.

This approach maintains user privacy and allows for interesting functionality and separation of concerns by design.

1.2.5 Push notifications

It would be very helpful to include push notifications alerting users when they have new messages or other relevant information. The Android API makes this fairly trivial to implement.

Chapter 2

Analysis/Requirements

2.1 Requirements

2.1.1 Minimum Viable Product

- Creation of accounts
- Logging in to accounts
- Pairing with other users (e.g. Friend Requests)
- Sending messages to paired users
- Push notifications for received messages on the relevant client
- Server-side db stores messages which have not been received yet and broadcasts them when user logs back in.

2.1.2 Stretch Goals

- Text-to-speech (TTS) with Google WaveNet API or Azure TTS
- Group chats
- Ability to mute chats
- End-to-end encryption
- Chat logs saved on client side
- File and image sharing

2.2 Technologies Used

2.2.1 Rust Server

I've elected to use the Rust language[12] for the server, as it provides many advantages over other languages in terms of reliability, security and speed.

Rust is a statically typed language like Java or C, which aids with software maintainability by reducing type errors and makes the documentation process simpler[8].

Due to its extensive use of syntactic sugar [17] as well as type inference, Rust allows us to create programs which are slightly less verbose, more readable and and easier to structure than C, C++ or Java equivalents[2].

Additionally, it is not a garbage-collected language[9] like Java, which allows for a significantly lower memory footprint as it allows for manual memory management. The major disadvantage of languages with direct memory management (e.g. C/C++) has traditionally been the lack of "memory safety"[5] - where direct memory allocation exposes the software to complex vulnerabilities such as buffer overflows and dangling pointers. Rust uses the twin concepts of "ownership" and "borrowing" [10] to elegantly provide memory safety while also maintaining the low memory footprint and performance benefits of direct memory managed languages.

2.2.2 Android Client

Initially, the client application will be written for Android using Kotlin. Traditionally, Android apps have been written in Java[14], but since the endorsement of the Kotlin language as an official development language for Android in May 2017[3], many developers have moved to using Kotlin.

The Kotlin programming language, like Java, runs on the Java Virtual Machine (JVM), which means that Java and Kotlin code can coexist within projects without issue. It provides a number of quality of life improvements for Java developers and reduces verbosity.

Development with Kotlin instead of Java is associated with higher quality applications[11], as measured by presence (or lack) of "code smells". Code smells, defined by Kent Beck and popularised by Martin Fowler[4], refer to small signs or indications within the code of a program that there is a deeper problem lurking beneath.

Ardito et al found that in practice, Kotlin is a more concise language than Java and produces significantly fewer null pointer exceptions[3].

It is likely that later on in the development stage SQLite3[16] will be used for a client-side db of chat history. SQLite is a highly performant embedded database which allows for local database storage without any networking required. Additionally, it has been built into Android since version 1.0 [1]

Chapter 3

Timeline & Final Product

3.1 Week by week

3.1.1 Week 0

Initial meeting discussing potential project options.

3.1.2 Week 2

Final discussion and agreement on project proposal/outline.

3.1.3 Week 4

- Build wireframes of android client using Balsamiq.
- Develop User Stories for application with MoSCoW prioritisation
- Begin populating Design chapter

3.1.4 Weeks 6-8

- Begin writing and researching for literature review
- Start building Kotlin and Android development competency
- Work on communication between Rust server and Android app
- Make a decision about which E2EE algorithm to implement
- Design initial testing strategy
- Complete Design chapter

3.1.5 Weeks 8-16

- Write Kotlin client application
- Write Rust server application
- Implement initial testing strategy

3.1.6 Weeks 18-20

- Additional coding as required
- Write a plan for evaluation
- Final polish based on evaluation outcomes

3.1.7 Weeks 22-24

- Focus on writing up final report
- Editing all previous work and proofreading
- Final polish on all areas

3.2 Final Product

The final output should be a Kotlin Android app which communicates with a Rust Server and includes all of the features of the Minimum Viable Product above and as many of the Stretch Goals that are achievable within the given timeframe.

Bibliography

[1] Aditya, S. K., and Karn, V. K. *Android sQLite essentials.* Packt Publishing, 2014.

[2] Ardito, L., Barbato, L., Coppola, R., and Valsesia, M. Evaluation of rust code verbosity, understandability and complexity. *PeerJ Computer Science* (2021).

[3] Ardito, L., Coppola, R., Malnati, G., and Torchiano, M. Effectiveness of kotlin vs. java in android app development tasks. *Information and Software Technology 127* (2020), 106374.

[4] Beck, K., Fowler, M., and Beck, G. Bad smells in code. *Refactoring: Improving the design of existing code 1*, 1999 (1999), 75–88.

[5] de Amorim, A. A., Hriţcu, C., and Pierce, B. C. The meaning of memory safety. In *International Conference on Principles of Security and Trust* (2018), Springer, pp. 79–105.

[6] Diffie, W., and Hellman, M. E. Privacy and authentication: An introduction to cryptography. *Proceedings of the IEEE 67*, 3 (1979), 397–427.

[7] Garfinkel, S. *PGP: pretty good privacy.* " O'Reilly Media, Inc.", 1995.

[8] Hanenberg, S., Kleinschmager, S., Robbes, R., Tanter, É., and Stefik, A. An empirical study on the impact of static typing on software maintainability. *Empirical Software Engineering 19*, 5 (2014), 1335–1382.

[9] Jones, R., and Lins, R. *Garbage collection: algorithms for automatic dynamic memory management.* John Wiley & Sons, Inc., 1996.

[10] Levy, A., Andersen, M. P., Campbell, B., Culler, D., Dutta, P., Ghena, B., Levis, P., and Pannuto, P. Ownership is theft: Experiences building an embedded os in rust. In *Proceedings of the 8th Workshop on Programming Languages and Operating Systems* (2015), pp. 21–26.

[11] Mateus, B. G., and Martinez, M. An empirical study on quality of android applications written in kotlin language. *Empirical Software Engineering 24*, 6 (2019), 3356–3393.

[12] Matsakis, N. D., and Klock, F. S. The rust language. *ACM SIGAda Ada Letters 34*, 3 (2014), 103–104.

[13] Milanov, E. The rsa algorithm. *RSA laboratories* (2009), 1–11.

[14] Murphy, M. L. *The busy coder's guide to Android development.* United States: CommonsWare, 2008., 2008.

[15] OORD, A. V. D., DIELEMAN, S., ZEN, H., SIMONYAN, K., VINYALS, O., GRAVES, A., KALCHBRENNER, N., SENIOR, A., AND KAVUKCUOGLU, K. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499* (2016).

[16] OWENS, M., AND ALLEN, G. *SQLite*. Springer, 2010.

[17] POMBRIO, J., AND KRISHNAMURTHI, S. Resugaring: Lifting evaluation sequences through syntactic sugar. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation* (2014), pp. 361–371.

[18] TAN, X., QIN, T., SOONG, F., AND LIU, T.-Y. A survey on neural speech synthesis. *arXiv preprint arXiv:2106.15561* (2021).