SPENGERGASSE
ausbildung mit zukunft

BRIAR

DIPLOMA THESIS

BriarJar
A GNU/Linux Desktop Client for Briar

Alexander Zeidler

Software Architecture and Implementation of Peer-to-Peer Applications

Farman Khan

From Terminal to Graphical
Comparison of Two Different User Interface Frameworks in Java

Vienna, April 2022

Supervisor	DI Dr. Andreas Chwatal
Department	Advanced Course for Computer Science
Technical College	HTL Spengergasse
Project Partner	Briar Project (https://briarproject.org)

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich den vorliegenden Diplomarbeitsteil / die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche erkenntlich gemacht habe.

Wien, am 05.04.2022

_____

Farman Khan

_____

Alexander Zeidler

Translated Declaration

I declare in lieu of oath that I have written this part of the thesis / thesis independently and without outside help, that I have not used any sources or aids other than those indicated, and that I have identified the passages taken verbatim and in terms of content from the sources used as such.

Kurzfassung

Im Zuge der Diplomarbeit BriarJar wurde ein gleichnamiger Messenger-Prototyp erstellt, der zeigt, wie der bekannte Messenger Briar abseits von Android auf dem gewohnten Desktop, aber auch im Terminal aussehen kann, um die Anzahl der Benutzer durch eine weitere Plattform zu erhöhen.

BriarJar wurde in der Programmiersprache Java für GNU/Linux-Betriebssysteme entwickelt, basiert auf dem freien (FLOSS) Messenger Briar und ist somit ebenfalls freie Software. BriarJar besteht aus zwei verschiedenartigen Benutzeroberflächen, einer grafischen und einer textbasierten. Diese sind über eine Schnittstelle (API), die ebenfalls im Rahmen der Diplomarbeit entwickelt wurde, mit dem Kern Briar verbunden. BriarJar bietet die Grundfunktionen der Erstellung eines lokalen Benutzerkontos, der Kontaktverwaltung und des Austauschs von Textnachrichten mit hinzugefügten Kontakten.

Der Messenger Briar selbst ist im Jahr 2018 nach sieben Jahren Entwicklungszeit offiziell in der Version 1.0 als Android-Applikation erschienen. Neben der freien und kostenlosen Nutzung, umfassen dessen Ziele unter anderem auch einen möglichst unabhängigen, zuverlässigen und unversehrten Transport von Nachrichten, selbst unter unglücklichen Netzwerkbedingungen. Als Netzwerk-Architektur kommt dabei das Peer-to-Peer-Modell zum Einsatz.

Der API-Teil der Diplomarbeit besteht aus zwei Teilen. Im theoretischen Teil werden zu Beginn relevante Netzwerk-Architekturen, deren Eigenschaften und (soziale) Auswirkungen erläutert. Anschließend werden kurz die beiden Peer-to-Peer-Applikationen Retroshare und Briar gegenüber gestellt, sowie ein Blick auf die Kommunikationsschicht von Briar geworfen. Zuletzt werden technische Details der beiden Overlay-Netzwerke "Invisible Internet Project" (I2P) und "The Onion Router" (Tor) verglichen und mit aktuellen Statistiken ergänzt. Der praktische Teil befasst sich hingegen mit der Entwicklung der API und gibt dabei einen Einblick über die verfolgten Ziele, deren Implementierung und abschließende Dokumentation.

Der Benutzeroberflächen-Teil der Diplomarbeit wurde in drei Abschnitte unterteilt. Der erste bietet eine generelle, geschichtliche Einleitung in Benutzeroberflächen (User Interfaces). Gefolgt von einer technischen Einleitung in graphischen (GUI) und text-basierten (TUI) User Interfaces (UI). Dabei werden sowohl Vergleiche von UI Frameworks aufgestellt, um festzustellen welche für das Projekt geeignet sind, als auch die Entwicklungsergebnisse beider UIs präsentiert. Der letzte Abschnitt besteht aus einem konklusiven Vergleich der beiden verwendeten UI Frameworks.

Das Diplom-Projekt wurde zeitgerecht und erfolgreich abgeschlossen, alle Eingangs erwähnten Ziele erreicht und der Quellcode, samt der Diplom-Arbeit selbst, auf Briar's Code-Hosting-Plattform (GitLab-Instanz) veröffentlicht.

Abstract

In the course of the diploma thesis BriarJar, a messenger prototype of the same name was created that shows how the well-known messenger Briar can look apart from Android on the familiar desktop, but also in the terminal, in order to increase the number of users through another platform.

BriarJar has been developed in the Java programming language for GNU/Linux operating systems, is based on the free (FLOSS) messenger Briar and is thus also free software. BriarJar consists of two different user interfaces, one graphical and one text-based. These are connected to the core Briar via an interface (API), which has also been developed as part of the diploma thesis. BriarJar offers the basic functions of creating a local user account, managing contacts and exchanging text messages with added contacts.

The messenger Briar itself was officially released in 2018 as an Android application in version 1.0 after seven years of development. In addition to being completely free to use, its objectives include ensuring that messages are transported as independently, reliably and uncorrupted as possible, even under unfortunate network conditions. The peer-to-peer network model is used as the network architecture.

The API part of the thesis consists of two parts. In the theoretical part, relevant network architectures, their characteristics and (social) impacts are explained at the beginning. Then the two peer-to-peer applications Retroshare and Briar are briefly compared, and a look is taken at Briar's communication layer. Finally, technical details of the two overlay networks "Invisible Internet Project" (I2P) and "The Onion Router" (Tor) are compared and supplemented with current statistics. The practical part, on the other hand, covers the development of the API and provides an insight into the objectives pursued, their implementation and final documentation.

The user interface part of the thesis was divided into three sections. The first provides a general, historical introduction to user interfaces. This is followed by a technical introduction to graphical (GUI) and text-based (TUI) user interfaces (UI). Comparisons of UI frameworks are made to determine which are suitable for the project, and the development results of both UIs are presented. The last section consists of a conclusive comparison of the two UI frameworks used.

The diploma project was successfully completed on time, all the objectives mentioned at the beginning were achieved and the source code, including the diploma thesis itself, was published on Briar's code hosting platform (GitLab instance).

Acknowledgments

First and foremost, we would like to express our gratitude to our supervisor, *Andreas Chwatal*, for his unwavering guidance, his very kind and always constructive criticism, but also his professionalism. We are very glad that he was our teacher and supervisor.

We would like to extend our deepest gratitude to the *Briar Project Team*. Their always unparalleled and immediate support - independent of formal or technical nature - was amazing.

In addition, we are especially indebted to our contact person from Briar Project, *Nico Alt*. From the moment we have initiated contact with the Briar Project, Nico has expressed his enthusiasm and motivation for our project to succeed. From lending a supportive hand for the formalities of project management to the practical advice during the course of development, Nico has played several important roles for our project, thank you!

We would also like to thank our English professor, *Lisa Haslinger*, for proofreading.

A special thanks also to *Alice* for her fundamental support including her regular feedback.

A thank you also to *Rosi* for her early support.

This document is a combination of two individual theses

Chapter two is the individual thesis topic of Alexander Zeidler

Software Architecture and Implementation of Peer-to-Peer Applications

Chapter three is the individual thesis topic of Farman Khan

From Terminal to Graphical
Comparison of Two Different User Interface Frameworks in Java

References and indexes are provided at the end of each individual thesis

Table of Contents

1 Introduction 12

1.1 General Design Concept 13

2 Software Architecture and Implementation of Peer-to-Peer Applications 15

2.1 Different Network Types and Their (Social) Impact 16
2.1.1 Network Topology of Client-Server, P2P and F2F Model 16
2.1.2 World-Wide Coherence and Influence 17

2.2 Architectural Details on Briar and Retroshare 20
2.2.1 Communication Layers 22
2.2.2 Briar's Application Protocol Stack 23

2.3 Invisible Internet Project and The Onion Router 25

2.4 BriarJar API Development 30
2.4.1 Introduction and API Functionalities 30
2.4.2 Handling of Invalid Parameter Input 31
2.4.3 Making Use of Checker Interface 32
2.4.4 General Exception - UI Related Unified Error Handling 33
2.4.5 Documentation with Javadoc 36

2.5 Conclusion and Lessons Learned 37
2.5.1 Result and Future Development Possibilities 37
2.5.2 Lessons Learned 38

2.6 Bibliography 40

2.7 Bibliography of Tables and Figures 41

2.8 Index of Tables and Figures 42

3 From Terminal to Graphical: Comparison of Two Different User Interface Frameworks in Java 45

3.1 Motivation 46

3.2 Historic Introduction – User Interfaces 47
3.2.1 Definition and Scope 47
3.2.2 Command-Line Interface 47
3.2.3 Terminal User Interface 48
3.2.4 Graphical User Interface 49

3.3 Introduction – Graphical and Terminal User Interface 49
3.3.1 Designing User Interfaces 49
3.3.2 Non-Functional Requirements – Usability 50
3.3.3 Usability – Target User 50
3.3.4 Usability – Measurement 50
3.3.5 Mapping Functional Requirements to UI Elements 51
3.3.6 Elements of the GUI – WIMP 51
3.3.7 Technical Requirements – Peripheral Hardware 52
3.3.8 Technical Requirements – Operating System and Software 52

3.3.9 Technical Requirements – User Interface Framework....52
3.3.10 Adherence to Limitations – Terminal User Interface....53
3.4 User Interface Frameworks – Terminal User Interface....54
3.4.1 Considered Frameworks....54
3.4.2 Used Framework - Lanterna....55
3.5 User Interface Frameworks – Graphical User Interface....56
3.5.1 Considered Frameworks....56
3.5.2 Used Framework - JavaFX....57
3.6 Approaches to Solutions – Terminal User Interface....58
3.6.1 Considered Approaches....58
3.6.2 Used Approach - Text GUI Toolkit....59
3.7 Results – Terminal User Interface....60
3.7.1 Initialisation....60
3.7.2 Accessing the BriarJar API and Event Handling....60
3.7.3 TUI Helper Class....60
3.7.4 Structure of Classes....61
3.7.5 Clipboard – Accessing AWT in the TUI....62
3.8 Approaches to Solutions – Graphical User Interface....63
3.8.1 Considered Approach - Declarative Programming....63
3.8.2 Used Approaches - Imperative Programming....63
3.9 Results – Graphical User Interface....64
3.9.1 Initialisation....64
3.9.2 Material Design....65
3.9.3 Accessing the BriarJar API and Event Handling....65
3.9.4 Structure of Classes....66
3.9.5 Differences in Main Screen Layout....66
3.10 Conclusive Comparison – Usability and Development....67
3.10.1 From a Developers Perspective....67
3.10.2 From a Users Perspective....67
3.11 Conclusion....70
3.12 Bibliography....71
3.13 Table of Figures....71

4 Contents of Compact Disk (CD)....73

5 Attachments....75

1 Introduction

A prototype of an free/libre and open source (FLOSS) desktop messenger named BriarJar has been developed. It is based on the peer-to-peer FLOSS messenger Briar.

The Android messenger Briar, officially released in 2018, has been engineered by The Briar Project (https://briarproject.org). Behind the scenes, they engineered much more to make the messenger possible, but also to ensure exemplary reliability, security and ease of use. Some of their protocols are described in chapters 2.2.1 "Communication Layers" and 2.2.2 "Briar's Application Protocol Stack".

BriarJar, written in Java, runs on GNU/Linux operating systems and offers the basic functions of local user account creation, contact management and the exchange of text messages with added contacts. Furthermore, BriarJar consists of two different user interfaces (UI). A graphical (GUI) one for usage on systems with an installed display server (usually all personal computers) and a text-based (TUI) one if no display server exists (usually on servers). With an existing display server the user can run either of both variants. Both GUI and TUI are included in the same runnable `.jar` file (hence, the project name). Per default the GUI variant is started if not otherwise specified by appending the command-line option `--tui` or `tui` when starting BriarJar.

To make the implementation of multiple user interfaces more convenient, an API has been developed between the UIs and the core Briar as shown in Figure 1.1.

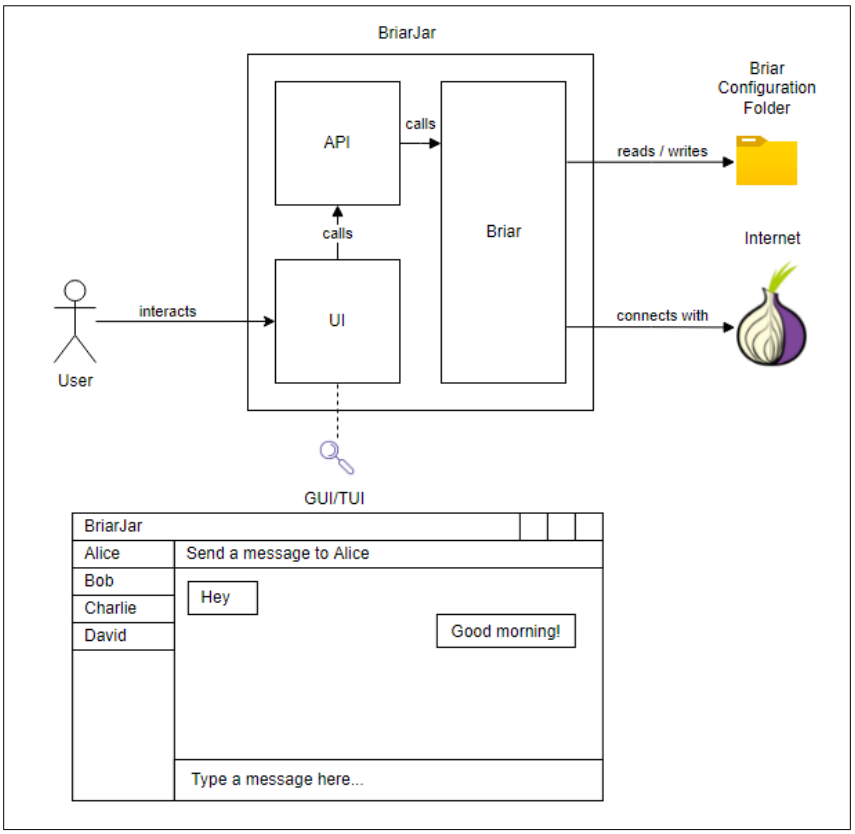


Figure 1.1: *BriarJar's system architecture*

1.1 General Design Concept

Various design and pattern decisions are applied throughout the diploma project. This intends to improve the readability and comprehension of the source code with its comments as well as the text within the provided project management documents.

The code style is purposely not completely consistent with Briar Project's source code or within the BriarJar project itself. The reason is to be able to experiment with different and individually-favoured styles beside not enforcing a one-fits-all style at this time.

Trying to keep the code clean in general has the advantage of making it easier for someone to continue the work later and reducing the risk of not finding bugs quickly.

To sum it up in one sentence, a major desire within the diploma project was that …

> **"[e]verything should be as simple as possible, but not simpler"**
> — Albert Einstein

2 Software Architecture and Implementation of Peer-to-Peer Applications

This chapter is the individual thesis topic of

Alexander Zeidler

References and indexes are provided at the end of the individual thesis

2.1 Different Network Types and Their (Social) Impact

This chapter provides an introduction to important network topologies (Figure 2.1) and their characteristics, followed by an outline of the different conflicts of interest.

2.1.1 Network Topology of Client-Server, P2P and F2F Model

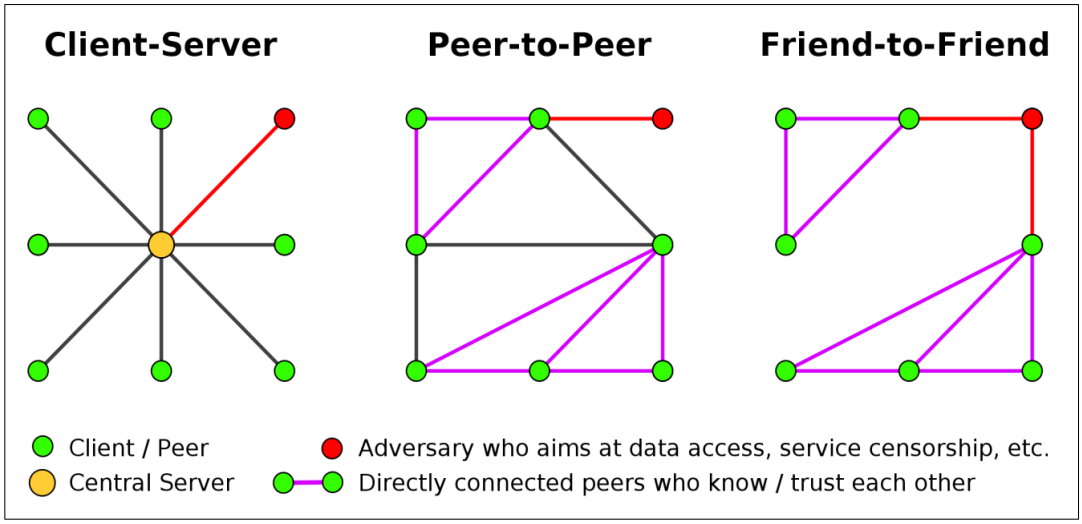


Figure 2.1: *Network topology of client-server, peer-to-peer and friend-to-friend model*

The centralised client-server network approach is nowadays widely used for accessing services like social media or "private" messaging. The peer-to-peer network approach is otherwise used, for example, to distribute files with BitTorrent clients when they operate in "trackerless" mode using a distributed hash table (DHT)[1]. The friend-to-friend network approach is a private variant of the peer-to-peer one, where connections are only established between trusted (previously verified) peers. Some software also offers the possibility to switch between network topologies or combine some of them.

Client-Server Model

The central approach uses the client-server model which is not only a powerful monopoly every user has to trust, furthermore, it is a single point-of-failure. For a successful adversary, or the service provider, it is easy and cheap to get information about the whole network or block access for certain users or even completely. Internet service providers are frequently forced by law to block (censor) certain services.

Peer-to-Peer (P2P) Model

The peer-to-peer approach can provide a high level of redundancy and load sharing. Each peer is usually both a server and client. While receiving public files or their metadata is no problem at all, taking unwanted files completely and permanently down can be exhausting. It depends on the implementation and the use of further defence techniques on how easy it is to get information like, for example, the physical location of a peer.

Friend-to-Friend (F2F) Model

The friend-to-friend approach provides a higher level of privacy since only trusted peers (friends) are connected with each other. But this comes with a drawback, since the network is much smaller than a global connected peer-to-peer one. Usually, there are less peers online and available around the clock but that is crucial to get data transferred. A big advantage is, that an adversary has it very difficult to launch large scaled attacks, since each friend-to-friend network has to be attacked on its own.

2.1.2 World-Wide Coherence and Influence

"Social problems need social solutions

[...] Digital and data driven "solutions" distract us from the real issues in society, and away from examining real solutions. [...]"

— Doug Specht, The Conversation [2]

In the last decades, the centralised network approach got more and more present. Nowadays, P2P applications seem like a useless relict for users since everything can be easily done by just using all the centralised services, just registrations are needed.

The P2P approach in comparison typically grants users more freedom since they do not have to rely on good will of service providers. But drawbacks exist, for example if peers are not online at the same time they will not be able to exchange the desired information.

There is also a conflict of interests related to the quite different network approaches. While the Internet started with the P2P approach, current commerce, authorities and users are seldom familiar with it any more. Furthermore, commercial vendors and authorities grew powerful with the centralised approach, thus, they would be anything but pleased by a change.

Commercial Interests

A move away from centralised services could mean that many people are either no longer dependent on a provided service (central server) or, in other areas such as the collecting society (copyright holders), are no longer willing to pay (with money or personal data) for something if cheaper or free access is also available. Since more and more people have access to the Internet through technology that does not reflect the real costs, the world is changing faster than in the 20th century. Thus, it would require an adaption to business models to the current time, for example, not publishing a song until a predefined sum has been collected, but after that releasing it in public domain. But in the last decade vendors like Adobe, Microsoft and Google started to advertise or partly even switch inevitably to subscription models where customers remain with a time-based usage-only right instead of keeping the ownership right of a legal bought copy like before. Such subscription business models cut users' rights and are per se nothing new. Ross Anderson references in his book "Security Engineering — Third Edition" [3], chapter 8.3.2 "The value of lock-in", to the book "Information Rules" from the year 1998:

"There is an interesting result, due to Carl Shapiro and Hal Varian: that the value of a software company is the total lock-in (due to both technical and network effects) of all its customers"

Authority Interests

Another party are state actors who also strive for finding ways to keep abreast with technological progression and the modern society. While some countries are notorious for their censorship and others for their intelligence agencies, yet others likely would not be that concerned about a come back of the P2P approach due to their established appreciation of privacy and freedom.

Whether it is justified responsibility or misuse of power, both works effectively with centralised services and even more effective when there is no publicly verifiable control committee existing or intact. Nevertheless, independent of the network architecture, no accountability at all is also no solution. This already starts with commercial spam as discussed in the book Peer to Peer [4].

Concerning global surveillance, it is not expensive to extract data in a client-server architecture compared to a P2P approach as seen by Edward Snowden's revelations [5] …

"in 2013, when over 50,000 Top Secret documents about the NSA's signals intelligence activities were leaked to the press. The scale and intrusiveness of government surveillance surprised even cynical security engineers. This brings us to the third big change, which is a much better understanding of security threats. In addition to understanding the capabilities and priorities of western intelligence agencies, we have a reasonably good idea of what the Chinese, the Russians and even the Syrians get up to." [6]

Winners

While users (citizens), authorities and commerce actors all have much more powerful opportunities available than ever before, users are making the least use out of it.

Undeniably, users can benefit the most from P2P applications. But also networking providers can benefit, their networks could get less stressed by the distributed load.

Developers who decide in favour of the P2P approach need also much fewer (financial) resources by not having to provide and maintain performant servers due to direct user connection to each other. Nevertheless, P2P applications are more complex [7] than client to server applications. But complexity also raises the error occurrence thus an essential advantage in the first place is required.

With P2P applications developed in good faith, users have the freedom to connect to who they want at any moment without being censored in their private communication, free expression or data sharing. Depending on the purpose of a P2P application, users can also benefit from a fail-safe network as long as there is either any other route available to the destination or other reachable peers are providing the same information requested.

With Great Power Comes Great Responsibility

P2P networking applications shift social responsibility from vendors to the users, by enabling freedom and protecting privacy too. To reduce socially inappropriate use, users are encouraged to either reveal such or in harmless cases to revoke peers' granted connection right.

Where large, in fact often unexpected, personal data collections are illegally or even legally stored, for lifetime, out of control of the individual, but accessible by "unknown" employees, adversaries of all sorts are attracted to obtain access and draw advantage of it. While data holders have to take cost-intensively special care of the data all the time, an adversary just needs to be successful once to cause irreparable proceeding damage. The past has already shown that it is only a matter of time before adversaries succeed.

"Mistrust authority - promote decentralization."

— Chaos Computer Club e. V. [8]

2.2 Architectural Details on Briar and Retroshare

This chapter introduces the application Retroshare, describes which overlay networks are used and how making contact and exchanging messages works in Briar.

Retroshare [9] and Briar both use the peer-to-peer (P2P) network approach or, more precisely, friend-to-friend (F2F). Retroshare uses the term F2F in their publications while Briar uses P2P. Figure 2.2 shows the F2F approach by Briar's current capabilities.

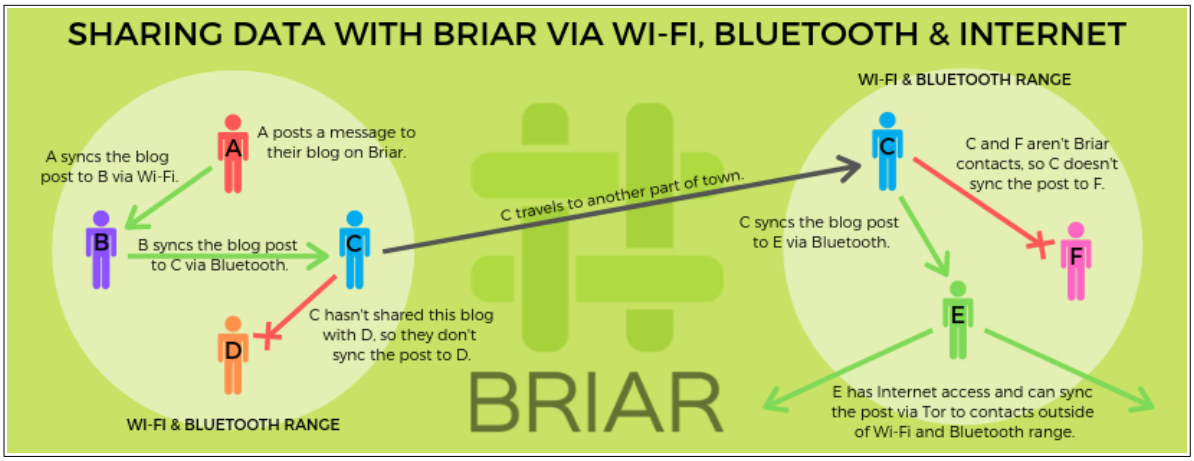


Figure 2.2: *F2F network approach based on Briar's current capabilities*

Retroshare exists primarily as a desktop application (Figure 2.3), but an experimental Android application with minimum features (for now) has been first released in 2018. Conversely, Briar exists primarily as an Android application (Figure 2.4), although they are working on a full-featured desktop client too.

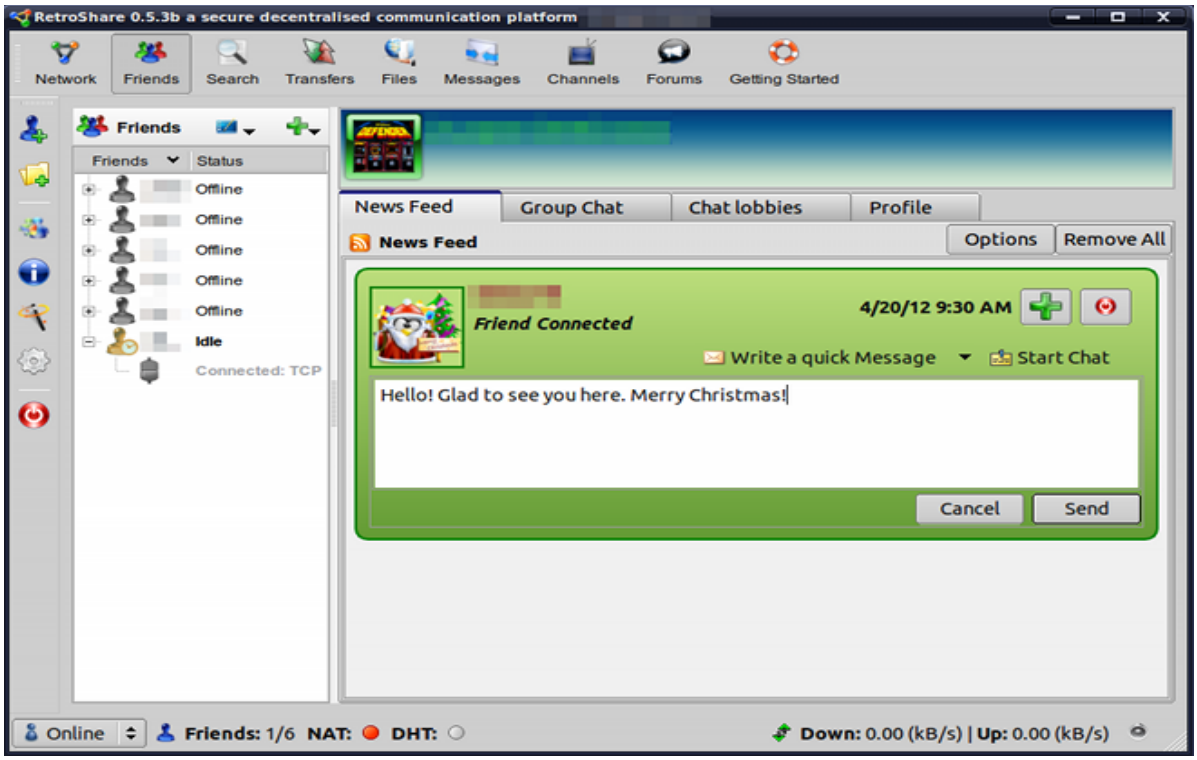


Figure 2.3: *Retroshare 0.5.3b (Year 2012)*

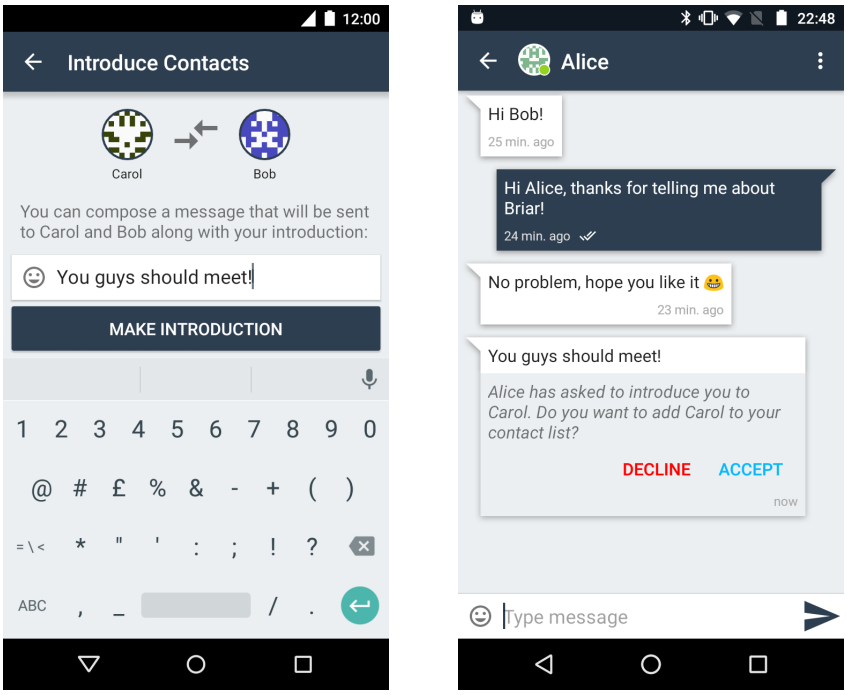


Figure 2.4: *Briar 1.2 on Android, Bob receives a contact introduction sent by Alice*

Retroshare shares some features with Briar like messaging with friends or discussing in forums. Albeit, Retroshare offers more such features at the moment, for example voice and video calls. But Retroshare is also less intuitive, or in other words more complex, regarded its configuration even though they provide depicted documentation as well as Briar does.

Retroshare's goals are quite similar to Briar's as perceptible in their documentation [10]:

"Retroshare creates encrypted connections to your friends. Nobody can spy on you. Retroshare is completely decentralized. This means there are no central servers. It is entirely Open-Source and free. There are no costs, no ads and no Terms of Service."

Comparing their timeline, Retroshare started as an idea in the year 2004 followed by the first official release in 2006, according their published history [11]. The development of Briar in contrast started in 2011 followed by its first release in 2018. Retroshare and Briar not only differ regarding their technological circumstances of their starting time, but also on how long it took until their first public releases (two years vs. seven years).

Skipping security details, where both try to satisfy higher standards than common widely used messengers, it is more relevant for this thesis on how their network architectures look like. Those are described in the following chapters.

2.2.1 Communication Layers

Table 1 shows the simplified communication protocol stack of Briar. The included application layer protocols are described in the next chapter.

Both Briar and Retroshare are able to use overlay networks on top of the regular Internet network layer (IPv4/IPv6) in order to anonymise and protect their traffic further. Their supported overlay network protocols are described in detail in the chapter 2.3 Invisible Internet Project and The Onion Router.

When Briar uses an Internet connection the traffic is unconditionally routed over the overlay network The Onion Router (Tor), as seen in the table below. But Tor is not used on the non Internet connections Wi-Fi, Bluetooth and removable media. According to their documentation [12], they plan to even support dial-up modems which are useful during Internet blackouts.

On the other hand, Retroshare can be used with any combination of Tor and I2P or none of both at all. Retroshare does not support further network layer protocols beside the regular Internet.

Table 1: *Simplified communication protocol stack of Briar*

<table>
<tr><th></th><th colspan="2">Contact Establishment</th><th colspan="2">Contact Communication</th></tr>
<tr><td rowspan="3">Application Layer</td><td rowspan="2">BQP</td><td>BRP</td><td colspan="2">Sync Clients</td></tr>
<tr><td>BHP</td><td colspan="2">BSP</td></tr>
<tr><td colspan="3">BTP[1]</td></tr>
<tr><td>Overlay Network</td><td>-</td><td>The Onion Router</td><td>-</td></tr>
<tr><td>Network Layer</td><td>Short-Range[2]</td><td>Internet</td><td>Non-Internet[3]</td></tr>
</table>

1: Briar's transport layer security protocol, operating in handshake / transport mode
2: Wi-Fi (no Internet), Bluetooth
3: Wi-Fi (no Internet), Bluetooth, Removable Drive, Dial-up Modem

2.2.2 Briar's Application Protocol Stack

As depicted in the previous chapter (Table 1) Briar consists of many application layer protocols. In the following, the protocols for establishing mutual contact and exchanging messages are described in summary. Details of the respective protocols are available on their wiki page [13]. Briar offers two ways for becoming contacts. Which one the user chooses depends mainly on the possibility of meeting in person, which is also recommended for various reasons.

Contact Establishment - Making Contact in Person

Traditionally, there is the Bramble QR Code Protocol (BQP) [14] which is used on the Android client since the beginning. Basically, making contact consists of two steps. First, both clients must scan each others QR code mutually (Figure 2.5). Such a QR code contains a commitment to an fresh ephemeral public key beside short-range connectivity information needed for step two. Second, both clients try to connect (unsecured) to each other via Wi-Fi and/or Bluetooth according to their received peer's connectivity information. On success, they exchange their previously committed public keys and verify them. Now they are ready to complete the Diffie–Hellman [15] key exchange, resulting in a common secret and passing a freshly derived ephemeral master key to the calling application for any further usage.

Contact Establishment - Making Contact on Distance

With method two, the Bramble Rendezvous Protocol (BRP) [16] and Bramble Handshake Protocol (BHP) [17] is used to add each other from distance (Figure 2.6) by mutually sharing a so called handshake link starting with `"briar://"` . It should only be used when method one is either not possible or would require a longer or risky journey. Sharing links enables attack vectors like to perform a man-in-the-middle (MITM) [18] attack. The here used protocols can neither detect nor protect against such a MITM attack. Therefore, it is a good idea to exchange the handshake links in the most secure way available. Splitting them across different communication channels, although it leaves additional traces, can also lower the success rate of an MITM attack.

Although Briar's handshake link looks similar to an onion link, it is not the same. Briar's handshake link, e.g. `briar://adb7ghqxhv3gpjezwi2hygcsb2pg5zf3mlxq6tmuth72zrlgqdsd6` , is just the own public key. After the successful exchange, both contacts are creating the same shared secret with their received public and their own private key. This shared secret is then used to create a static master key together with both public keys. The result is a rendezvous key. In further steps, this rendezvous key is used to create endpoints for each transport. Both clients create the same pseudo-random contact details needed to find each other. Each contact is now waiting for the other and tries to connect once per minute for up to 48 hours. If they fail constantly during this time, both are closing their network endpoints and stopping to reach each other. On success, the peers are exchanging their long-term contact details and drop their used temporal ones.

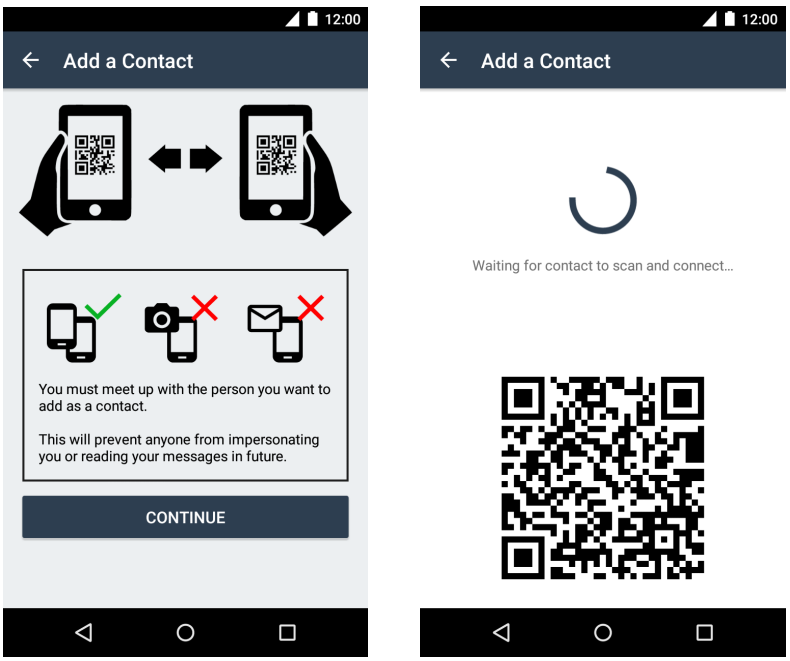


Figure 2.5: *Making contact nearby through QR codes*

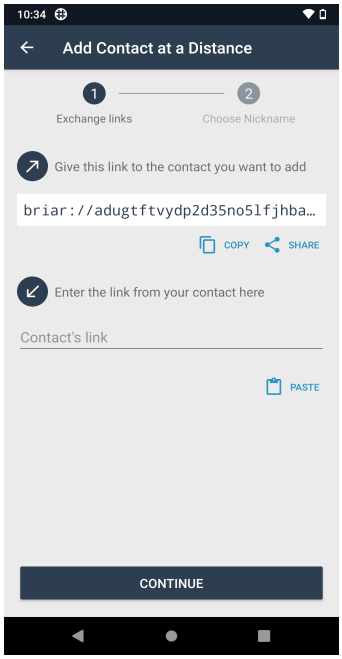


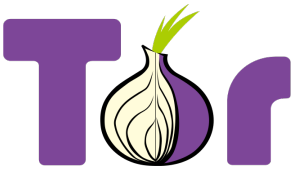
Figure 2.6: *Making contact at distance through handshake link*

Contact Communication

The Bramble Synchronisation Protocol (BSP) [19] is used by various functional clients, each responsible for its own task, to synchronise data (called messages) between contacts via BTP. For example, there are clients for private messages, forums or for exchanging available transport information. The clients also have sovereignty over what is to be synchronised. The clients' messages are organised into groups (labels). Devices can subscribe to groups and thus stay up to date. In reality, the whole implementation is more complex, but the principle is outlined.

2.3 Invisible Internet Project and The Onion Router

Source: https://geti2p.net



This chapter describes the characteristics and use of the Invisible Internet Project (I2P) [20] and The Onion Router (Tor) [21].

Retroshare can make use of either I2P or Tor or both. Their use is advertised when, for example, connecting to untrusted contacts ("friends") and thereby hiding the own real IP address. But Retroshare can also be configured to use those anonymising networks all the time, unconditionally. Briar on the other hand routes its Internet traffic unconditionally over the Tor network. Currently, Briar does not make use of I2P.

I2P and Tor share some similarities while also differing completely in other fields as explained in this chapter. But it is important to consider, that both have limitations of their anonymity effectiveness when, for example, their networks are largely monitored.

I2P was released in the year 2003 as a fork of Freenet [22]. Its parts are licensed under different free and open source licenses [23]. I2P uses the so called "garlic routing" [24], a variant of the "onion routing" [25] technique. Garlic routing, compared to onion routing, uses multiple and uni-directional tunnels from the origin to the destination. Default there are six I2P routers (hops) included in one direction (three in the own outbound tunnel and three in destination's inbound tunnel). But the I2P Client Protocol (I2CP) [26] configuration can also be changed, since it is always a trade-off between speed, privacy and security. I2P's garlic routing also means that multiple messages are sent to the outbound tunnel endpoint in bundles, called cloves, which improves performance, plus there is great flexibility for further endpoint routing techniques.

Tor was also publicly released in the year 2003 and is nowadays open source BSD 3-clause licensed. It uses the onion routing technique which was developed in the mid-1990s to protect U.S. intelligence's online communication. In the year 2006, the non-profit organisation "The Tor Project" was founded to maintain software for Tor.

I2P's website offers an overview [27] about the differences between I2P and Tor. Important differences include that I2P is designed and optimised for hidden services (similar to Tor's onion services), has a fully distributed design, continuously fact-based peer selection, varying and untrusted directory servers, advanced load balancing and resilience, and unidirectional (short-lived) tunnels. Tor on the opposite is designed and optimised for exit traffic (leaving the Tor network after some hops to reach services on the regular Internet), therefore consists of a large number of exit nodes, has a larger user

base, greater academic interest and research, significant funding, a large (partly funded) development team and greater resistance to state-level censorship (TLS transport and bridges). Tor offers furthermore a higher data throughput and lower latency. Although, when using onion services there are six hops between origin and destination, not just three. I2P default uses six hops to the destination and different six hops back to the origin. Details are explained on their performance page [28], beside they are also continuously working on performance improvements as recently achieved in the releases 0.9.48 [29] (2020-12-01) and 1.7.0 (API 0.9.53) [30] (2022-02-21).

When participating in the Tor network as a user, mostly the recommended Tor Browser Bundle is used for visiting regular websites or URLs ending with .onion (onion services). While doing so, the user's device is just a client and forwards no traffic for other users, neither as a middle relay nor as an exit node. Renting a server and participating as a middle relay is easily possible by first getting information on Tor's official website, followed by installing and configuring Tor appropriately. Running an exit node is also possible, but this comes with a higher responsibility and time effort since there is a much higher risk of legal confrontations. This will happen when Tor network users are visiting websites or using services through this exit node which they should not or doing other illegal things with the exit node's public IP address.

I2P follows another approach regarding their client and relay affiliation. Every I2P installation, called a router, is per default a client as well as a relay for the whole I2P network. By doing so, I2P reaches currently a relay amount of almost 30.000, running on about 18.000 IP addresses as shown in Figure 2.7. The regularly observed drops in the number of routers in the figure could, for example, be due to design-related I2P software updates or blockade attacks. On the next page are further figures (Figure 2.8, 2.9, 2.10) to get a first overview about network's usage. The depicted statistics are public available [31], along with two related papers ("An Empirical Study of the I2P Anonymity Network and Its Censorship Resistance" [32], "Measuring I2P Censorship at a Global Scale" [33]).

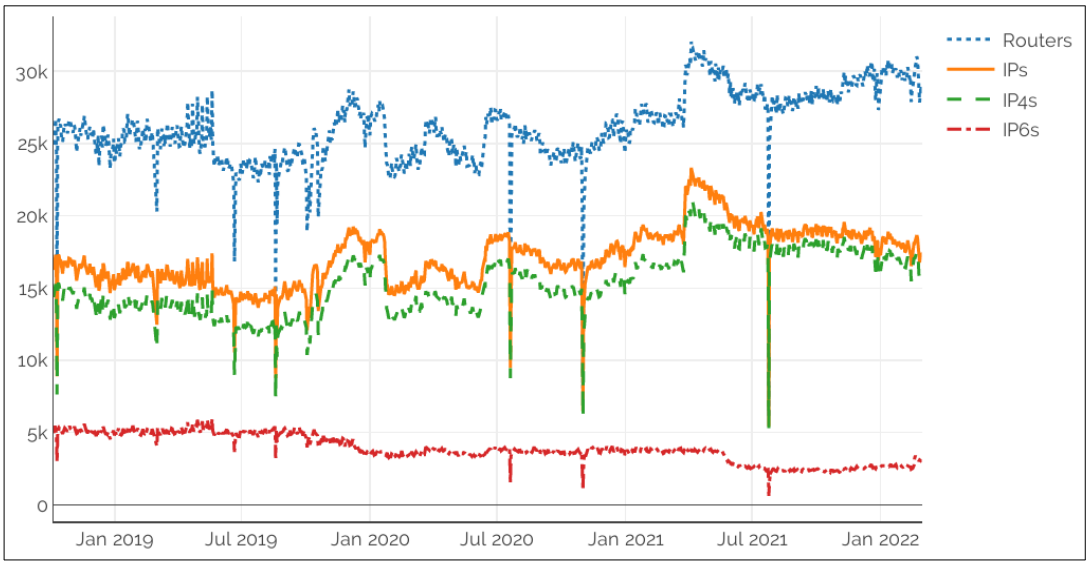


Figure 2.7: *Number of routers and IP addresses*

As seen in Figure 2.10, most I2P routers (compared to common Tor relays) are offering a rather low bandwidth either due to its configuration (automatically or manually set) or limitations by non I2P related network factors.

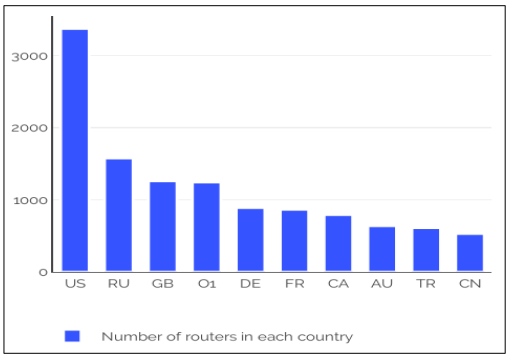


Figure 2.8: *Top ten countries (on 2022-03-01)*

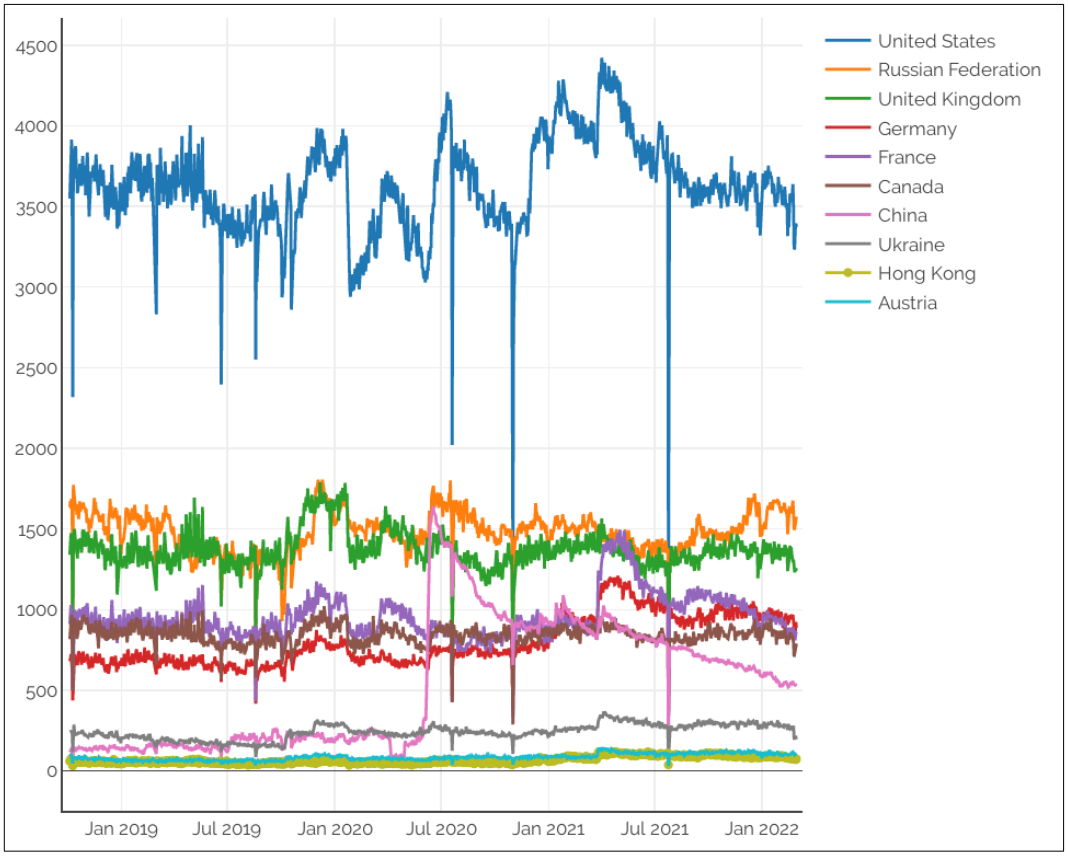


Figure 2.9: *Geo-distribution of I2P routers for some countries/regions*

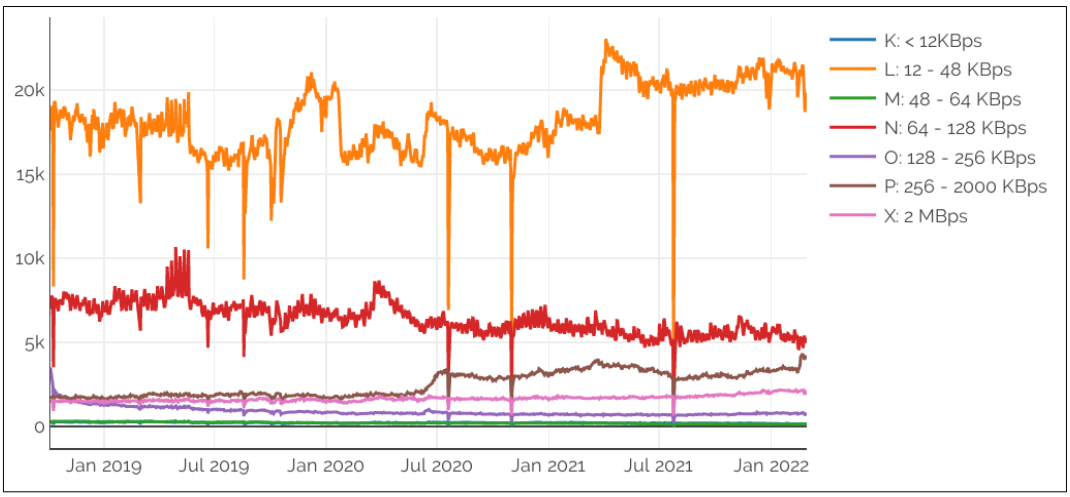


Figure 2.10: *Routers by shared bandwidth flag*

In comparison to I2P, the Tor network is used much more as Figure 2.11 shows.

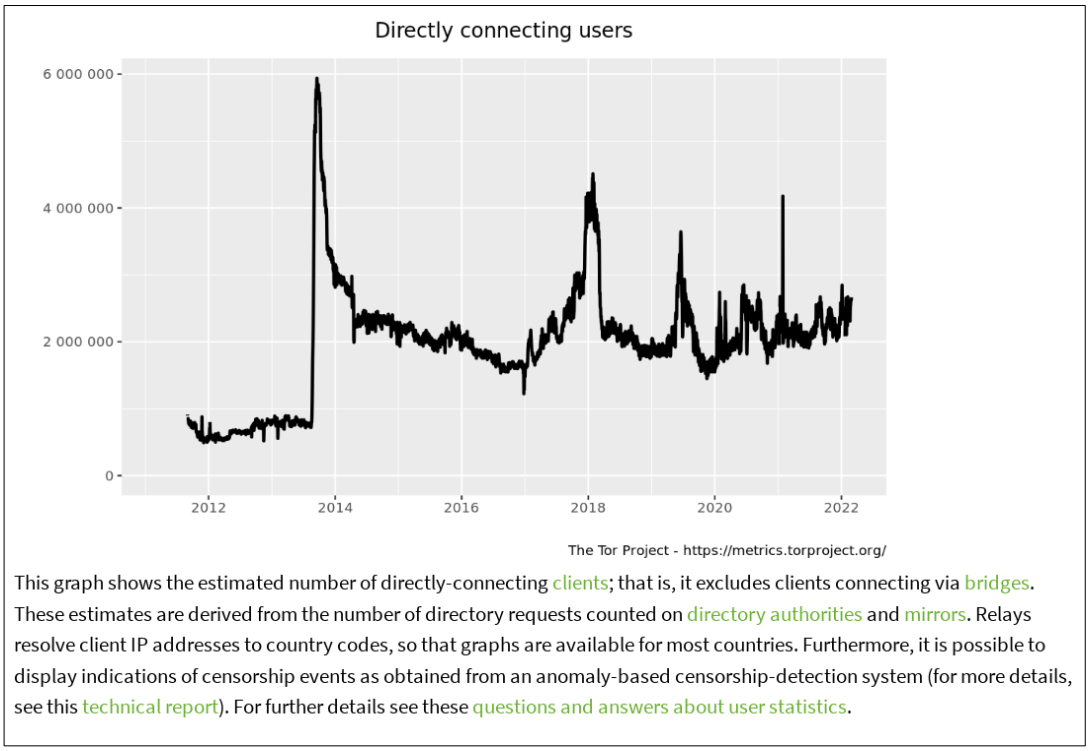


Figure 2.11: *Amount of directly connected Tor users world wide*

Figure 2.12 shows the same period of time for connections from Austria. The biggest sudden increase of Austrian users in the year 2013 is the same as seen global in Figure 2.11. It was the reaction on Edward Snowden's revelations described in chapter 2.1.2 "World-Wide Coherence and Influence" at the end of Authority Interests.

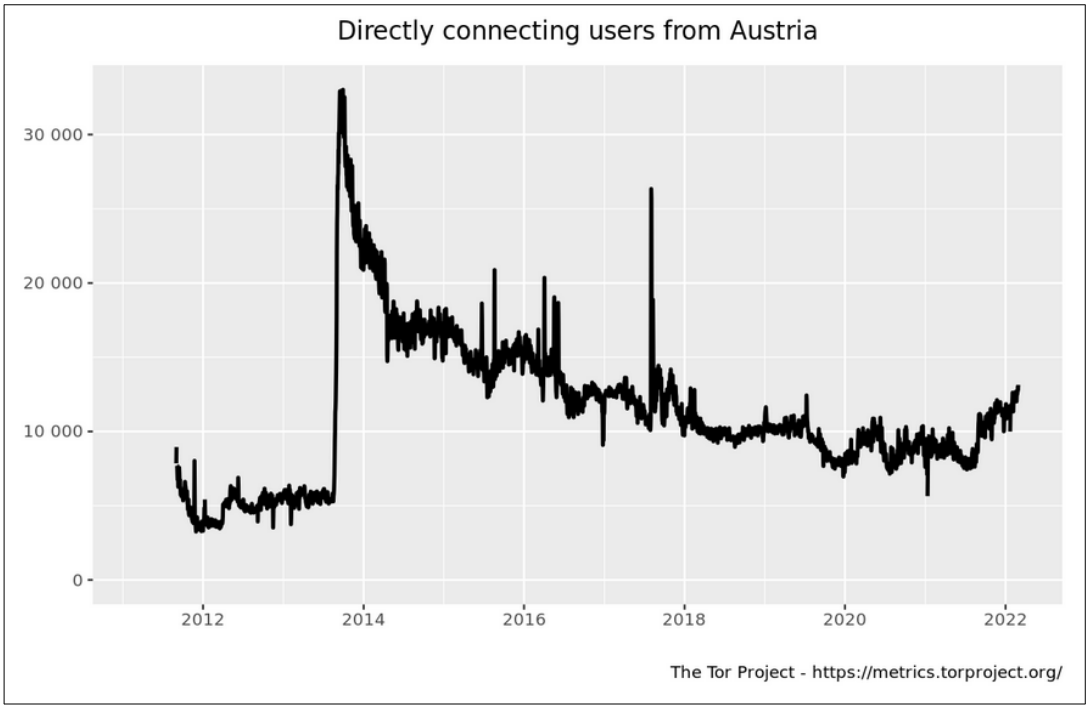


Figure 2.12: *Amount of directly connected Tor users from Austria*

Figure 2.13 and 2.14 are estimations of the top ten countries connecting directly or via bridges to the Tor network. IP addresses of bridges are exchanged on a regular base and not publicly listed. Therefore they can be used to try to circumvent censorship.

Country	Mean daily users
United States	530078 (19.90 %)
Germany	247654 (9.30 %)
Finland	159501 (5.99 %)
Russia	133477 (5.01 %)
India	110657 (4.15 %)
Ukraine	104831 (3.93 %)
France	85644 (3.21 %)
United Kingdom	79515 (2.98 %)
Netherlands	76382 (2.87 %)
Uzbekistan	75094 (2.82 %)

This table shows the top-10 countries by estimated number of directly-connecting clients. These numbers are derived from directory requests counted on directory authorities and mirrors. Relays resolve client IP addresses to country codes, so that numbers are available for most countries. For further details see these questions and answers about user statistics.

Figure 2.13: *Estimated number of directly connected users*

Country	Mean daily users
Russia	32734 (45.13 %)
United States	6988 (9.63 %)
Germany	3343 (4.61 %)
Iran	2816 (3.88 %)
Netherlands	1782 (2.46 %)
United Kingdom	1756 (2.42 %)
France	1739 (2.40 %)
China	1441 (1.99 %)
Belarus	1396 (1.92 %)
India	1332 (1.84 %)

This table shows the top-10 countries by estimated number of clients connecting via bridges. These numbers are derived from directory requests counted on bridges. Bridges resolve client IP addresses of incoming directory requests to country codes, so that numbers are available for most countries. For further details see these questions and answers about user statistics.

Figure 2.14: *Estimated number of users connected via bridge*

Figure 2.15 shows that four out of seven relays could be used to enter the Tor network (Guards) and every fifth to exit it, by connecting to the regular Internet (IPv4/IPv6).

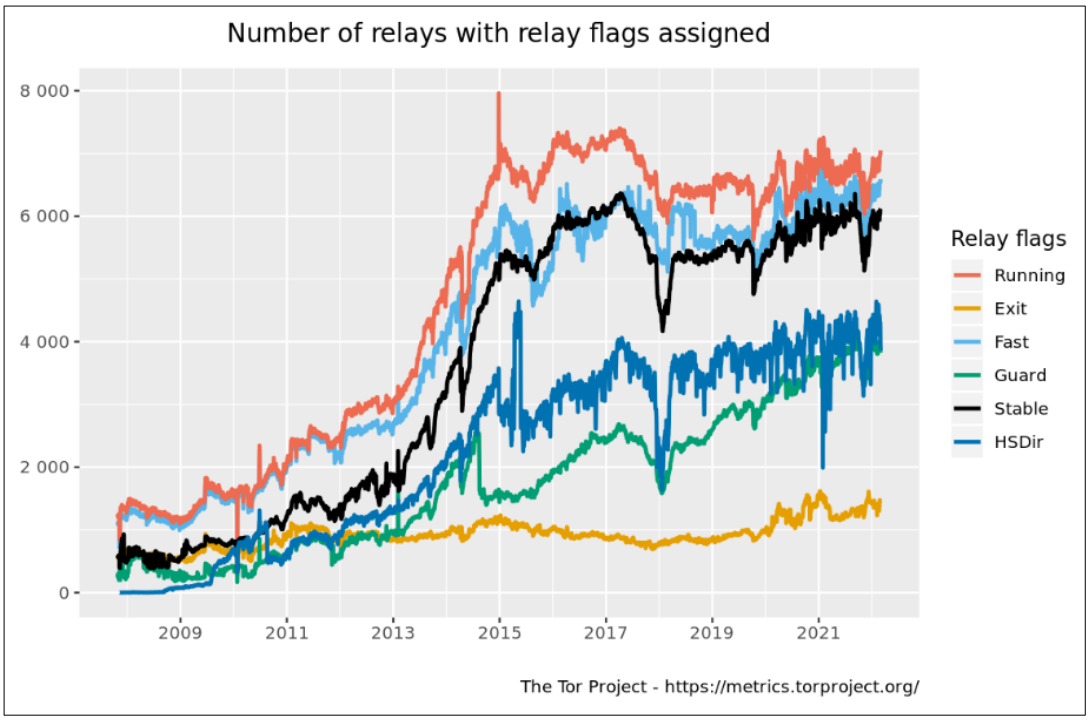


Figure 2.15: *Running Tor relays with their flags assigned by directory authorities*

2.4 BriarJar API Development

After much has been explained about Briar and the background of peer-to-peer networks in the previous theoretical part, this part is dedicated to the considerations and implementation of the BriarJar API.

2.4.1 Introduction and API Functionalities

As illustrated in chapter 1 "Introduction", BriarJar's API acts between the UI and Briar. In BriarJar's source code the API classes are named "view models" and are documented as exemplary shown in chapter 2.4.5 Documentation with Javadoc. Calling the API instead of Briar's methods directly has a few advantages as quoted in the following.

Firstly, since two UIs are implemented in BriarJar both of them would need to include the same logic resulting in duplicated boilerplate code. But duplicate code is susceptible to inconsistency, increases the amount of maintenance required and is also more error-prone than a common code that is used and hopefully reviewed by many people.

Secondly, extras can be implemented that are not included in the core Briar, for example "General Exception - UI Related Unified Error Handling", chapter 2.4.4.

Thirdly, the UI code includes much less logic so it is easier to focus on the needed UI development and push the proper execution responsibility more towards the API.

Finally, as seen in Table 2 below, the few API classes include everything needed by the BriarJar UIs and it is located on a single place. This makes it easy for developers who are not yet familiar with the Briar source code to get started. Additionally, The Briar Project includes well documentation online but also in their source code.

Table 2: *Classes (View Models) and functionalities of BriarJar's API*

View Model	Provided Functionalities
Event Listener	Subscribe / Unsubscribe as listener (messages sent, private message received, ...)
Login	Does an account exist Sign up (inclusive passphrase strength feedback) Sign in Is signed-in currently (to perform account / messaging related actions) Delete account
Lifecycle	Start / Stop Briar services Automatically stop Briar services when exiting the application
Contact	Add a new pending (yet not mutually accepted) contact Remove a non-/pending contact Get a contact for more information (username, alias, ...) Set an alias for a contact
Conversation	Get "group count" per contact (message count, unread count, latest timestamp) Write a new message Get message headers (timestamp, outgoing or incoming, read by us, sent, seen by contact, ...) Get message text Set read flag Delete multiple / all messages

2.4.2 Handling of Invalid Parameter Input

Since the API is called from outside (usually foreign UI code) it is important to verify passed input for its reasonability to avoid undesired events or application crashes.

The input can be distinguished into entered by the user and (unintentionally) passed by the programmer who calls the API. In the end both can be risky.

While some invalid input will be handled by Briar itself, other invalid input would cause problems and is therefore checked up front by the BriarJar API. When invalid input is detected the API does inform the outside caller (UI) as explained in chapter 2.4.4 "General Exception - UI Related Unified Error Handling" and abort the called method.

This fail fast principle is implemented by the API where the input value can be checked for compliance certainly or not in cases where Briar already takes care about and the API only has to process Briar's reaction and inform the outside caller about.

By implementing (partly additional) validation into the API instead into each outside caller's code it further more meets best practise to not write duplicate code where sensible. This leads also to higher code quality when audited by more people.

Last but not least, early handling of discovered improper values could lead to shorter error logs and more precise error messages as shown in the example below. Note that the source code differs from more recent versions, but the principle remains the same.

```
contactId = null;  // simulate implementation error (reference to nowhere)
throwOnNullParam( "ContactId", contactId );  // red scenario, no check up front
throwOnNullParam( "ContactId", contactId );  // green scenario, check up front
return contactManager.getContact( contactId );
```

```
Exception in thread "main" java.lang.NullPointerException: Cannot invoke
"org.briarproject.bramble.api.contact.ContactId.getInt()" because "c" is null
at ..bramble.db.JdbcDatabase.containsContact(JdbcDatabase.java:1212)
at ..bramble.db.JdbcDatabase.containsContact(JdbcDatabase.java:101)
at ..bramble.db.DatabaseComponentImpl.getContact(DatabaseComponentImpl.java:533)
at ..bramble.contact.ContactManagerImpl.lambda$getContact$2(ContactManagerImpl.java:177)
at ..bramble.db.DatabaseComponentImpl.transactionWithResult(DatabaseComponentImpl.java:212)
at ..bramble.contact.ContactManagerImpl.getContact(ContactManagerImpl.java:177)
```

```
java.lang.IllegalArgumentException: ContactId can not be null  // text used to display an
at ..briarjar.utils.Checker.throwOnNullParam(Checker.java:36)  // error message to the user
```

```
at ..briarjar.viewmodel.ContactViewModel.getContact(ContactViewModel.java:157)
at ..briarjar.tui.ContactList.getAliasForList(ContactList.java:151)
at ..briarjar.tui.ContactList.updateContactList(ContactList.java:124)
at ..briarjar.tui.ContactList.createWindow(ContactList.java:67)
at ..briarjar.tui.ContactList.render(ContactList.java:103)
at ..briarjar.tui.TUIUtils.switchWindow(TUIUtils.java:124)
at ..briarjar.tui.SignIn.enterPassphrase(SignIn.java:100)
at ..briarjar.tui.SignIn.createWindow(SignIn.java:53)
at ..briarjar.tui.SignIn.render(SignIn.java:76)
at ..briarjar.tui.TUIUtils.switchWindow(TUIUtils.java:123)
at ..briarjar.tui.MainTUI.start(MainTUI.java:50)
at ..briarjar.Main.main(Main.java:34)
```

2.4.3 Making Use of Checker Interface

As seen in the example of chapter 2.4.2 "Handling of Invalid Parameter Input", a static method is called to test a passed value (reference) to not be null as a simple first check or otherwise throw an exception (hence, trigger an error situation).

```
public interface Checker {

/** Documentation ... */
static <T> T throwOnNullParam(String name, T object) throws
IllegalArgumentException
    {
        if ( object == null )
            throw new IllegalArgumentException( name+" can not be null" );
        return object;
    }
}
```

This method from the interface called `"Checker"` is straight forward and avoids boilerplate (duplicated) code. Currently, that is the only method implemented and it is almost the same as `"Objects.requireNonNull"` . The only difference is that at BriarJar's `"Checker.throwOnNullParam"` an `"IllegalArgumentException"` (IAE) is thrown instead of a `"NullPointerException"` (NPE). Although nowadays the official `"Objects.requireNonNull"` method exists and states to throw an NPE when "doing parameter validation in methods and constructors" [34], often it is still a matter of opinion which one to throw. Since a helpful message can be included either way, it is rather important make a decision per project and stay consistent.

BriarJar's API only throws IAE. That is the case if a passed argument is invalid whether it is `"null"` or does not meet another expected condition. While an NPE can occur unexpectedly when an invalid resource is tried to being used and the API perhaps missed beforehand to recognise, catch and handle an illegal argument properly.

Both IAE and NPE extend a `"RuntimeException"` which in turn is an unchecked exception. Integrated Development Environments (IDE) will therefore not complain when it is not declared to be caught.

In BriarJar's API IAE are caught and handled differently according their situation.

If the called API method returns a `"String"` to be shown to the user (e.g. a message text) and this API method will likely be called multiple times, then the returned `"String"` will be simply replaced by a message like: `"<Can not get message text from a null message ID>"` This will neither force the application to stop nor is the user perhaps being flooded with identical error message windows. Instead this approach will draw attention moderately.

Where such a procedure is not possible or appropriate, an exception is wrapped into a `"GeneralException"` (GE) as explained in the following chapter 2.4.4 General Exception - UI Related Unified Error Handling.

2.4.4 General Exception - UI Related Unified Error Handling

When a method of BriarJar's API is called from outside (usually foreign UI code) and something went wrong during processing, the outside caller has to be notified.

In order to provide the caller with appropriate information (at least error title and error message) a unified exception class named `"GeneralException"` (GE) has been created. Such a GE is intended do be directly used by appearing error windows and can wrap (include) a preceding occurred exception.

In the example below a code snippet of BriarJar's API is shown where a new contact should be added. Depending on the error situation either a `DbException` or `IllegalArgumentException` would be thrown.

The `if (...)` part is implemented by BriarJar and ensures that the user is not able to add the own handshake link instead of the contact's one. Since this error cause is obvious a message should implicitly be specified when throwing the `"IllegalArgumentException"`. This exception (including the message) will then be used by the GE.

On the other hand, the `.getHandshakeLink()` method is implemented by Briar and therefore it is perhaps not certainly clear up front what the reason is if the call fails. Ideally, a thrown exception already has a meaningful message included. Otherwise, there is a mechanism implemented as explained later in "Further Constructor Parameters".

In the end, both exceptions are caught and handled in the same way by throwing a new General Exception provided with the common title `"Checking handshake-link"`.

```
public void
       addPendingContact( String link,     // expect new contact's handshake-link
                          String alias )
throws GeneralException
{
    try {
        throwOnNullParam("Handshake Link", link);
        (...)
        if ( link.equals( contactManager.getHandshakeLink() ) )
            throw new IllegalArgumentException( "You entered your own " +
                 "handshake-link, but the link of your contact is needed" );
    }
    catch (DbException | IllegalArgumentException e) {  // handle either of both
        throw new GeneralException( e, true, "Checking handshake-link" );
    }
    (...)
}
```

Constructors

The General Exception constructors were created with the objective in mind to minimise the amount of required code written by callers and thereby hold the constructor call short. This objective has been reached, see for example this single line:

```
throw new GeneralException( e, true, "Checking handshake-link" );
```

While making use of throwing General Exceptions it has soon been noticed that this approach comes with two complexity drawbacks. First, when writing the code ("Which parameter is next?") and second when reading the code ("Which of the two available booleans is it?").

While Java supports constructor and method overloading (same constructor or method names within a class but different order of input parameter types), it is mandatory that the parameter signatures (Figure 2.16) are distinguishable during creation. So in the following example both constructors can not exist in the same class since just the parameter names are interchanged and it is not clear which one should be called when creating the exception for throwing:

```
public GeneralException( String message, String title ) … // one of them has to be
public GeneralException( String title, String message ) … // removed or altered
```

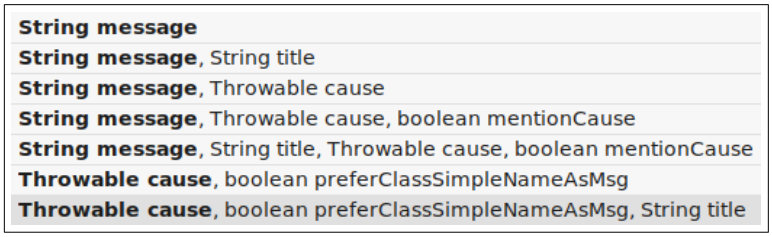


Figure 2.16: *Too many constructors are rather confusing*

Retrospectively, it presumably would have been better to avoid those confusing constructors and rather implement a builder to be easily useable, for example:

```
throw new GeneralExceptionBuilder().setTitle( "Checking ..." )
                                   .setMessage( "Can not..." )
                                   .setCause( e )
                                   .(...)
                                   .build();
```

Further Constructor Parameters

When no message is specified when creating a General Exception with its constructor, so it just inherits a thrown and caught exception, and `"preferClassSimpleNameAsMsg"` is set to `"false"` in the constructor than the original exception message (Figure 2.17) is being inherited by the General Exception unconditional.

Since the original message could be useless by either being null, blank or merely the fully-qualified exception class name, `"preferClassSimpleNameAsMsg"` can be set to `"true"` in the constructor, in order to analyse if it is reasonable to set at least the shortened (not fully-qualified any more) exception class name (Figure 2.18) as message for the user to be less confusing respectively more informative.

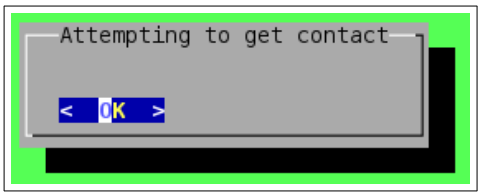


Figure 2.17: *Inherited (preceding) exception message could be e.g. empty*

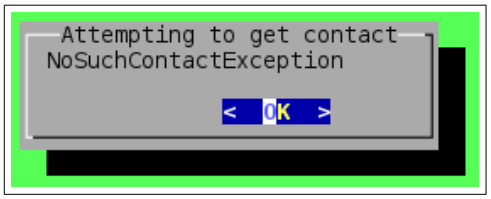


Figure 2.18: *Message influenced by* `"preferClassSimpleNameAsMsg = true"`

In other cases where the cause of a thrown exception is roughly clear, a message can be specified in General Exception's constructor beside inheriting the exception itself anyway and `"mentionCause"` can be set to `"true"` . In this way the simple exception class name from the caught and inherited exception gets appended within parenthesis (Figure 2.19) in order to, hopefully, create a more clear or detailed error message.

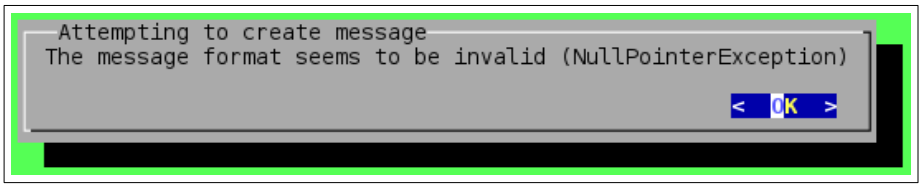


Figure 2.19: *Appended simple exception class name within parenthesis*

2.4.5 Documentation with Javadoc

The API documentation strives to follow the KISS (keep it simple, stupid) principle, too.

For the creation of descriptions of applicable tags (depending on the related method, constructor or class), it was also necessary to go again recursively through the code of all called Briar methods to properly document the behaviour in the respective error cases.

The created Javadoc is manually specified above methods (Figure 2.20), constructors and classes, could be rendered and directly used within the IDE (Figure 2.21) and finally generated to HTML pages (Figure 2.22) via JDK's official command line tool `"javadoc"` or graphically via IDE.

```
/**
 * Gets a collection of private message headers exchanged with the passed
 * {@link org.briarproject.bramble.api.contact.ContactId}, useful for
 * calling {@link #getMessageText(MessageId)}.
 *
 * @param   c  contact's {@link org.briarproject.bramble.api.contact.ContactId}
 *           , not null
 * @return  a {@link Collection} of {@link org.briarproject.briar.api.conversation.ConversationMessageHeader}s
 *
 * @throws GeneralException if compliance is not met or returning is not
 *                          possible for another reason
 *
 * @see org.briarproject.briar.api.conversation.ConversationManager#getMessageHeaders(ContactId)
 *
 * @since 1.0
 */
public Collection< ConversationMessageHeader >
       getMessageHeaders( ContactId c )
throws GeneralException
{
```

Figure 2.20: *Specifying Javadoc*

Gets a collection of private message headers exchanged with the passed ContactId, useful for calling getMessageText(MessageId).

Params: c – contact's ContactId , not null

Returns: a Collection of ConversationMessageHeaders

Throws: GeneralException – if compliance is not met or returning is not possible for another reason

Since: 1.0

See Also: ConversationManager.getMessageHeaders(ContactId)

```
public Collection< ConversationMessageHeader >
       getMessageHeaders( ContactId c )
throws GeneralException
{
```

Figure 2.21: *Javadoc rendered by IDE*

OVERVIEW PACKAGE CLASS USE TREE INDEX HELP

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD SEARCH: Search

getMessageHeaders

public Collection<ConversationMessageHeader> getMessageHeaders(ContactId c)
throws GeneralException

Gets a collection of private message headers exchanged with the passed ContactId, useful for calling getMessageText(MessageId).

Parameters:

c - contact's ContactId , not null

Returns:

a Collection of ConversationMessageHeaders

Throws:

GeneralException - if compliance is not met or returning is not possible for another reason

Since:

1.0

See Also:

ConversationManager.getMessageHeaders(ContactId)

Figure 2.22: *Generated HTML pages of Javadoc*

2.5 Conclusion and Lessons Learned

The idea and development of BriarJar arose from the desire to be able to use the Briar Android Messenger on free (as in freedom) desktop operating systems as well. As explained in the theoretical part, Briar has a number of features that the usual messengers cannot compete with. Be it the free creation of user accounts, the great independence from service providers or the exemplary implementation of data security and data protection. The discussion of the different network topologies and their (social) impact brings closer why Briar is already a sought-after and crucial tool in difficult areas of this world. Also compared are the proven overlay network technologies "Tor" and "I2P", which form the foundation for peer-to-peer applications such as Briar or RetroShare. While "Tor" is more publicly known than "I2P", both are rather little used in the mainstream Internet world.

The practical implementation of BriarJar basically consisted of two development parts, the one that is rather invisible to the user, which represents the interface (API) to the core Briar, and the one that is visible via GUI or TUI, which communicates with this API. The implementation of the API is explained in a relatively technical way in chapter 2.4 BriarJar API Development. At its beginning, the advantages of using the API are highlighted and the implemented functionalities that can be accessed by user interfaces are listed. It is then explained in detail how the API reacts to invalid input and how it passes valuable error information to the user interface in the event of an error. Finally, different images of the written API documentation are shown.

2.5.1 Result and Future Development Possibilities

With the successful completion of the API, it is now possible for the user interfaces developed within BriarJar, as well as possible future ones, to conveniently access its functions.

Although the current implementation of BriarJar is working well, there is of course room for further development as described below.

This diploma project builds on a frozen and thus in the mean time out of date Briar code base inclusive their shipped Tor network binary. Therefore, BriarJar should not currently be used outside of a test environment until it is maintained and its dependencies are regularly updated.

The current General Exception solution can be reimplemented with a builder which would reduce the complexity on callers side.

Own automated software tests would be great. For example, a whole iteration from creating multiple accounts to verifying messages sent and received. This was not possible until now due to lack of time and because of the statically configured communication

ports between Briar and Tor. In recent Briar versions this has been changed and it is now easy to run multiple Briar clients for testing purposes at the same time.

Further functionalities can also be added to the Checker class.

2.5.2 Lessons Learned

Working on the diploma project brought a lot of new knowledge and experience. From the definition of the project goals and the formation of the project team, to the practical and formal realisation of the project, to the acquisition of further programming knowledge in the different areas.

The idea of developing a desktop client based on Briar already came up in autumn 2019, two years before the start of the project, and was included in the project management exercises at the time. The number of project objectives at that time was greater than the number of objectives actually planned and completed. Even then, care was taken to make the most realistic estimates possible, despite the lack of programming experience in Java. The immense workload of the last two semesters finally determined how extensive the result could be.

In order to achieve the objectives of the diploma project as well as possible, it was necessary to adhere to a few principles. For example, it was tried to find team members who were as interested in Briar and free software as possible, but who had also completed all previous semesters. The size of the team also played an important role, which is usually between three and five people for diploma projects. Due to the small number of people and the principles set, only a two-person team was possible, despite a preference for two to three people. A team of maximum three persons should, among other things, reduce the necessary communication effort and thus also be oriented towards the KISS principle. However, a possible loss of one team member would have had a stronger negative impact on the completion of the project.

The work on the diploma project was planned in such a way that progress would be rather irregular but substantial. The holiday periods were particularly suitable for this, as there was little time available during the semesters. The intensive development periods lasting several days ensured that the focus was correspondingly high and did not have to be repeatedly interrupted in the middle for an indefinite period of time. However, one disadvantage is that questions that arise have to be answered very promptly in order not to block the compact progress of the project for too long. The holiday period is also only moderately suitable for questions, which is why more emphasis was placed on joint self-study instead. Nevertheless, the teamwork was great due to the mutual flexibility.

The work on the project management documents was well divided and did not follow a rigid pattern. While each one was responsible for certain parts, the progress of these was regularly discussed together. The mostly common critical feedback had an increased time cost, but it also raised the quality of the elaborations and broadened the perspectives.

It took some time to read the source code of Briar and to understand how the individual components interact. The dependency injection framework Dagger2 also caused confusion in the beginning, as the use of such a framework had not been part of the lessons until then. Accordingly, the Java class design, which was revised several times, was also delayed. Once the class design was reasonably established, work could begin on the individual parts. Of course, there were also further joint reflections and partial code reviews. While one tried to penetrate new subject areas and write initial code, the other then worked on optimising it.

Generally, during the work on the diploma project, it would have been helpful in part to work less in self-study and instead have more frequent discussions with programming professors and Briar. In the end, it was a compromise that could have been improved, which was also influenced by the necessary pandemic measures.

Nevertheless, the whole diploma project was a valuable experience. It was also a lot of fun, brought a lot of exchange, and the planned objective, the prototyping of a free GNU/Linux desktop client based on Briar, was also achieved.

2.6 Bibliography

[1] https://en.wikipedia.org/wiki/Distributed_hash_table, web.archive.org, 2022-03-04

[2] https://theconversation.com/tech-companies-collect-our-data-every-day-but-even-the-biggest-datasets-cant-solve-social-issues-118133, web.archive.org, 2022-01-01

[3] "Security Engineering - Third Edition" by Ross Anderson (2020), chapter 8.3.2, p.270, https://www.cl.cam.ac.uk/~rja14/book.html, free accessible after 2024, April, web.archive.org, 2022-01-01

[4] Nelson Minar & Marc Hedlund in "Peer to Peer" by Andy Oram (2001), p.12, 1.2.2 The breakdown of cooperation

[5] https://en.wikipedia.org/wiki/Global_surveillance_disclosures_(2013-present), web.archive.org, 2022-01-01

[6] "Security Engineering - Third Edition" by Ross Anderson (2020), Preface to the Third Edition, p.1, https://www.cl.cam.ac.uk/~rja14/book.html, free accessible after 2024, April, web.archive.org, 2022-01-01

[7] Nelson Minar & Marc Hedlund in "Peer to Peer" by Andy Oram (2001), p.16, 1.3.2 Decentralization, 1st paragraph

[8] https://www.ccc.de/en/hackerethics, web.archive.org, 2022-01-01

[9] https://retroshare.cc/, web.archive.org, 2022-03-01

[10] https://retrosharedocs.readthedocs.io/en/latest/, web.archive.org, 2022-03-01

[11] https://retroshare.readthedocs.io/en/latest/about/history/, web.archive.org, 2022-03-01

[12] https://code.briarproject.org/briar/briar/-/wikis/A-Quick-Overview-of-the-Protocol-Stack, web.archive.org, 2022-03-01

[13] https://code.briarproject.org/briar/briar-spec/-/tree/master/protocols, web.archive.org, 2022-03-01

[14] https://code.briarproject.org/briar/briar-spec/blob/master/protocols/BQP.md, web.archive.org, 2022-03-01

[15] https://en.wikipedia.org/wiki/Diffie-Hellman_key_exchange, web.archive.org, 2022-03-03

[16] https://code.briarproject.org/briar/briar-spec/blob/master/protocols/BRP.md, web.archive.org, 2022-03-01

[17] https://code.briarproject.org/briar/briar-spec/blob/master/protocols/BHP.md, web.archive.org, 2022-03-01

[18] https://en.wikipedia.org/wiki/Man-in-the-middle_attack, web.archive.org, 2022-03-03

[19] https://code.briarproject.org/briar/briar-spec/blob/master/protocols/BSP.md, web.archive.org, 2022-03-01

[20] https://geti2p.net/en/, web.archive.org, 2022-03-01

[21] https://www.torproject.org/, web.archive.org, 2022-03-02

[22] https://freenetproject.org/, web.archive.org, 2022-03-01

[23] https://geti2p.net/en/get-involved/develop/licenses, web.archive.org, 2022-03-04

[24] https://geti2p.net/en/docs/how/garlic-routing, web.archive.org, 2022-03-04

[25] https://en.wikipedia.org/wiki/Onion_routing, web.archive.org, 2022-03-04

[26] https://geti2p.net/en/docs/protocol/i2cp, web.archive.org, 2022-03-04

[27] https://geti2p.net/en/comparison/tor, web.archive.org, 2022-03-02

[28] https://geti2p.net/en/about/performance, web.archive.org, 2022-03-02

[29] https://geti2p.net/en/blog/post/2020/11/30/0.9.48-Release, web.archive.org, 2022-03-02

[30] https://geti2p.net/en/blog/post/2022/2/21/1.7.0-Release, web.archive.org, 2022-03-02

[31] https://i2p-metrics.np-tokumei.net/, web.archive.org, 2022-03-02

[32] Nguyen Phong Hoang, Panagiotis Kintis, Manos Antonakakis, and Michalis Polychronakis. 2018. An Empirical Study of the I2P Anonymity Network and its Censorship Resistance. In Proceedings of the Internet Measurement Conference 2018 (IMC '18). ACM, New York, NY, USA, 379-392. https://arxiv.org/pdf/1809.09086.pdf

[33] Nguyen Phong Hoang, Sadie Doreen, and Michalis Polychronakis. 2019. Measuring I2P Censorship at a Global Scale. In Proceedings of the 9th USENIX Workshop on Free and Open Communications on the Internet (FOCI '19). USENIX, Santa Clara, CA, USA. https://www.usenix.org/conference/foci19/presentation/hoang

[34] https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/Objects.html#requireNonNull(T,java.lang.String), web.archive.org, 2022-02-16

2.7 Bibliography of Tables and Figures

Page	Table	Reference
Page 22	Table 1	*© 2022 Alexander Zeidler, licensed under Attribution 4.0 International (CC BY 4.0)* *https://creativecommons.org/licenses/by/4.0/*
Page 30	Table 2	

Page	Figure	Reference
Title Page	Briar Logo	https://briarproject.org/, *© 2018 Sublime Software Ltd. Creative Commons 4.0 Attribution License (CC BY 4.0)* *https://creativecommons.org/licenses/by/4.0*, Note according to CC BY 4.0: Unchanged redistribution
Page 12	Figure 1.1	*© 2022 BriarJar Project Team, licensed under Attribution 4.0 International (CC BY 4.0)* *https://creativecommons.org/licenses/by/4.0/*
Page 16	Figure 2.1	*© 2022 Alexander Zeidler, licensed under Attribution 4.0 International (CC BY 4.0)* *https://creativecommons.org/licenses/by/4.0/*
Page 20	Figure 2.2	https://briarproject.org/img/diagram_sharing.png, *https://briarproject.org/copyright/, Created by the Open Technology Fund's Localization Lab.*
Page 20	Figure 2.3	https://delightlylinux.wordpress.com/2012/04/20/retroshare-secure-private-file-sharing/, *© Delightly Linux*
Page 21	Figure 2.4	https://briarproject.org/manual/, *https://briarproject.org/copyright/, The mobile phone image is reproduced or modified from work created and shared by Vernon Chan and used according to terms described in the Creative Commons 2.0 Attribution License https://creativecommons.org/licenses/by/2.0*, Note according to CC BY 4.0: Unchanged redistribution
Page 24	Figure 2.5	
Page 24	Figure 2.6	
Page 26	Figure 2.7	https://i2p-metrics.np-tokumei.net/, *© Nguyen Phong Hoang, All plots and data are free to be used for academic research purposes.*
Page 27	Figure 2.8	
Page 27	Figure 2.9	
Page 27	Figure 2.10	
Page 28	Figure 2.11	https://metrics.torproject.org, *© The Tor Project, Inc., Graphs are licensed under a Creative Commons Attribution 3.0 United States License (CC BY 3.0 US) (https://creativecommons.org/licenses/by/3.0/us/).* Note according to CC BY 3.0 US: Unchanged redistribution
Page 28	Figure 2.12	
Page 29	Figure 2.13	
Page 29	Figure 2.14	
Page 29	Figure 2.15	
Page 34	Figure 2.16	*© 2022 Alexander Zeidler, licensed under Attribution 4.0 International (CC BY 4.0)* *https://creativecommons.org/licenses/by/4.0/*
Page 35	Figure 2.17	
Page 35	Figure 2.18	
Page 35	Figure 2.19	
Page 36	Figure 2.20	
Page 36	Figure 2.21	
Page 36	Figure 2.22	

2.8 Index of Tables and Figures

Table 1: Simplified communication protocol stack of Briar....22
Table 2: Classes (View Models) and functionalities of BriarJar's API....30

Figure 1.1: BriarJar's system architecture....12
Figure 2.1: Network topology of client-server, peer-to-peer and friend-to-friend model...16
Figure 2.2: F2F network approach based on Briar's current capabilities....20
Figure 2.3: Retroshare 0.5.3b (Year 2012)....20
Figure 2.4: Briar 1.2 on Android, Bob receives a contact introduction sent by Alice....21
Figure 2.5: Making contact nearby through QR codes....24
Figure 2.6: Making contact at distance through handshake link....24
Figure 2.7: Number of routers and IP addresses....26
Figure 2.8: Top ten countries (on 2022-03-01)....27
Figure 2.9: Geo-distribution of I2P routers for some countries/regions....27
Figure 2.10: Routers by shared bandwidth flag....27
Figure 2.11: Amount of directly connected Tor users world wide....28
Figure 2.12: Amount of directly connected Tor users from Austria....28
Figure 2.13: Estimated number of directly connected users....29
Figure 2.14: Estimated number of users connected via bridge....29
Figure 2.15: Running Tor relays with their flags assigned by directory authorities....29
Figure 2.16: Too many constructors are rather confusing....34
Figure 2.17: Inherited (preceding) exception message could be e.g. empty....35
Figure 2.18: Message influenced by "preferClassSimpleNameAsMsg = true"....35
Figure 2.19: Appended simple exception class name within parenthesis....35
Figure 2.20: Specifying Javadoc....36
Figure 2.21: Javadoc rendered by IDE....36
Figure 2.22: Generated HTML pages of Javadoc....36

3 From Terminal to Graphical: Comparison of Two Different User Interface Frameworks in Java

This chapter is the individual thesis topic of

Farman Khan

References and indexes are provided at the end of the individual thesis

This thesis deals with the application of two different user interface (UI) frameworks onto a single interface, which is written by the project group “BriarJar”.

Both frameworks differ in modality and each framework will be utilized in the same Java application in the form of a Java Archive (jar file). One allows human-computer interaction (HCI) using the prevalent graphical user interface (GUI) and the other allows HCI using a terminal/text-based user interface (TUI). Both require a computer with a GNU/Linux operating system, a monitor and keyboard, but the GUI application extends it’s requirements to a computer mouse and a display server (e.g. Xorg), which implies an enhancement in accessibility for the TUI application (on “headless” servers), but may limit the ability to depict multimedia, like images and audio. However, since the project requirements are limited to text-based messaging only, no major visual limitations between GUI and TUI shall apply.

3.1 Motivation

This thesis is part of the a diploma project where the project outcome is a messaging software client, which shall be ready-to-use on a GNU/Linux operating system where the current general availability version of Java is running. One of the non-functional requirements of the project is the freedom to decide in which way human-computer-interaction should occur. The motivation to write about this topic arose from great interest towards the users of the target platform and commonly observed opinions from the UNIX-community on the topic user interface, experience and interaction.

3.2 Historic Introduction – User Interfaces

The first part of this thesis provides a brief historic introduction into user interfaces to generate interest about this topic and make the decisions in the following sections easier to follow.

3.2.1 Definition and Scope

A user interface is defined as the space where interaction between humans and machines occur. The interaction has the goal of operation and control, while the machine has to provide the human with real-time or near-real-time feedback to the interaction.

Historically, this wasn't necessarily an accurate definition. In the time of batch interfaces and punch cards (1945 – 1968) where holes were punched in cards according to a prearranged code, user interfaces were considered rudimentary and the feedback was not real-time, but instead took hours to days usually. Since – by definition – batch interfaces are not aligned with what is generally perceived as a user interface, they are out of the scope of this introduction.

User interfaces which meet the criteria defined before are command-line interfaces (CLIs), text-based or terminal user interfaces (TUIs) and graphical user interfaces (GUIs). In the project-related part of this thesis (after the historic introduction), the focus will be aimed towards TUI and GUI development respectively.

3.2.2 Command-Line Interface

By connecting batch monitors directly to the system console the first console applications allowed users to input text, receive real-time or near-real-time input feedback and send the input text to programs. CLIs have allowed user to be interactive in ways which weren't possible before 1969 and are still frequently used due to their efficiency and the manageability using less Human Interface Devices than the Graphical user interface would require.

The rise of command-line interfaces began initially from a remote information exchange over teleprinter (TTY), which was later replaced by "glass tty" also called "computer terminals" or "smart terminals" which then also permitted to the interaction with cursor movement. In the 1980s and 1990s the primary user interface of Microsoft Windows and Apple Macintosh PCs was the CLI which quickly made it's way to the secondary UI while being replaced by the GUI. To this day, it's still the secondary UI on most platforms to allow system administrators and power users efficient computer usage and batch processing.

```
bash-5.1$ ip
Usage: ip [ OPTIONS ] OBJECT { COMMAND | help }
       ip [ -force ] -batch filename
where  OBJECT := { link | address | addrlabel | route | rule | neigh | ntable |
                   tunnel | tuntap | maddress | mroute | mrule | monitor | xfrm |
                   netns | l2tp | fou | macsec | tcp_metrics | token | netconf | ila |
                   vrf | sr | nexthop | mptcp }
       OPTIONS := { -V[ersion] | -s[tatistics] | -d[etails] | -r[esolve] |
                    -h[uman-readable] | -iec | -j[son] | -p[retty] |
                    -f[amily] { inet | inet6 | mpls | bridge | link } |
                    -4 | -6 | -I | -D | -M | -B | -0 |
                    -l[oops] { maximum-addr-flush-attempts } | -br[ief] |
                    -o[neline] | -t[imestamp] | -ts[hort] | -b[atch] [filename] |
                    -rc[vbuf] [size] | -n[etns] name | -N[umeric] | -a[ll] |
                    -c[olor]}
```

Figure 3.1: *The efficiency of command-line applications depends on the understanding of their usage*

3.2.3 Terminal User Interface

Chronologically, GUI development began at least 17 years earlier than TUI. TUIs arose from the Common User Access from the System Application Architecture. Microsoft adheres to that model in their Disk Operating System (DOS) and Windows Console Applications.

TUIs were often used for the input screens of mainframes. In comparison to the character-oriented protocol Telnet, the protocol tn3270 (which was made for mainframes) sent complete input masks to a terminal emulator. In 1990, a DOS-based TUI framework called Turbo Vision by Borland was developed and in 1991, Microsoft shipped Visual Basic with DOS. In the UNIX-world, TUIs are often created using ncurses or curses.

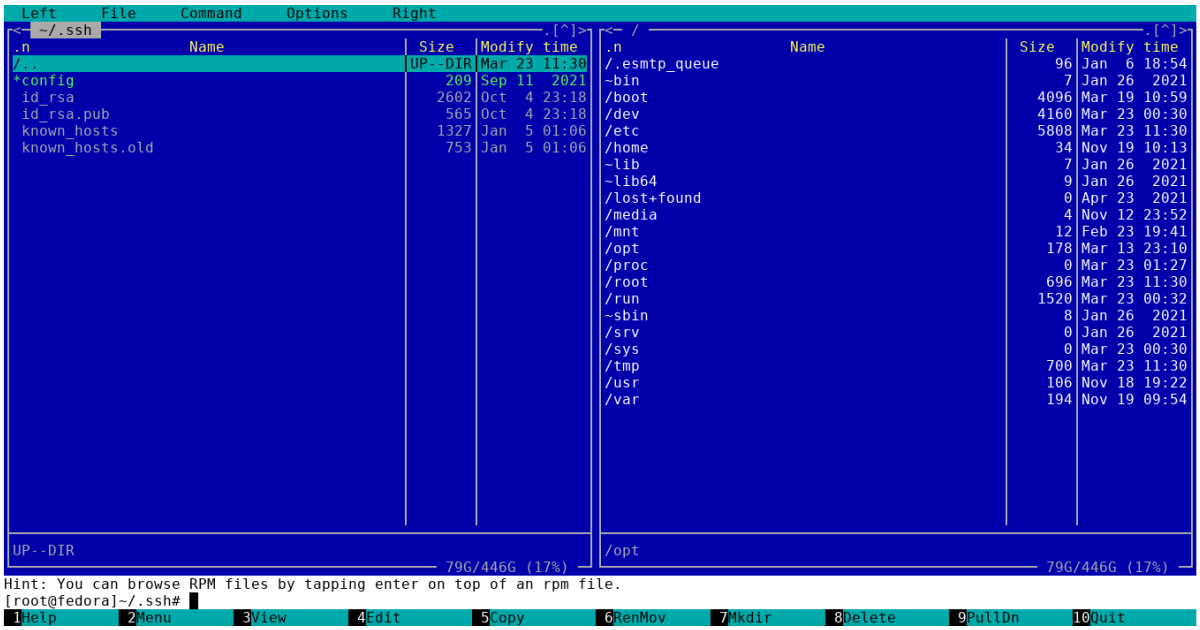


Figure 3.2: *Midnight Commander (mc) is a TUI-based shell for UNIX-like systems*

3.2.4 Graphical User Interface

"A display connected to a digital computer gives us a chance
to gain familiarity with concepts not realizable in the physical world.
It is a looking glass into a mathematical wonderland."

- Sutherland, The Ultimate Display (1965) [1]

The earliest step into the development of GUIs has been done by Ivan Edward Sutherland, who invented the Sketchpad, the predecessor of the GUI. The Sketchpad was a computer-aided design program which used a light pen to draw engineering sketches and also allowed to edit and manipulate them in real-time.

The pioneer of graphical user interfaces was the oN-Line-System (NLS), which was developed in the 1960s by Douglas Engelbart. It was the implementation of the computer mouse, hypertext links and screen windowing, raster-scan video monitors and other modern concepts. The following years, a lot of GUI development happened at the Xeros Palo Alto Research Centre, Apple, Digital Research, IBM and Microsoft. A lot of research has been done on the topic of user interaction and standards like WIMP (windows, icons, menus, pointing device) as well as principles like POLA ("principle of least astonishment") have gained popularity in the field of user interface development.

Other research includes multiple studies done on children ranging from the age of 5 to 15 where the main focus was to establish an understanding of *what makes a UI user-friendly,* because "designs that accommodate the needs of children, older adults, and users with disabilities can improve the quality for all users." [2]

By changing the target group from working-class adults to children, software development has shifted it's focus from utility-centrism to consumer-centrism and various principles have been made by different companies to increase the intuitiveness of user interfaces.

3.3 Introduction – Graphical and Terminal User Interface

In this section several design decisions, limitations, technical, functional and non-functional requirements and general concepts and paradigms for both user interfaces will be explained in depth. Since the terminal user interface is historically inherited (see Historic Introduction) from graphical user interface, almost all design choices are equivalent for both UIs with the exception of certain limitations (see Adherence to Limitations).

3.3.1 Designing User Interfaces

The primary goal of designing a UI is to enhance *usability.* Programming design patterns like MVC (Model-View-Controller) allow the user interface to be independent of and indirectly linked to the functions of an application. In order to design the UI, the

functional requirements, the necessary UI elements and their functions have to be declared while adhering to the non-functional requirements (*usability*).

3.3.2 Non-Functional Requirements – Usability

Standardized requirements allow UI developers to make development decisions which are well suited for most users. The European norm EN ISO 9241-110 has defined *usability* as "The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use" [3]. Jakob Nielsen, a usability consultant, and Ben Shneiderman, a computer science professor, have declared that *usefulness* is composed of the following [4]:

Learnability	How easy is it for users to accomplish basic tasks the first time they encounter the design?
Efficiency	Once users have learned the design, how quickly can they perform tasks?
Memorability	When users return to the design after a period of not using it, how easily can they re-establish proficiency?
Errors	How many errors do users make, how severe are these errors, and how easily can they recover from the errors?
Satisfaction	How pleasant is it to use the design?

It's noteworthy that experts on HCI discourage the use of the term *intuitiveness* and promote the usage of the term *familiarity* instead [5].

3.3.3 Usability – Target User

In order to answer the questions of usability, our target user has to be defined. The target user of BriarJar...

- is a GNU/Linux user.
- has a basic understanding of computer software download, installation and usage (e.g. has used other desktop messengers before).
- is informed about how Briar is ought to be used on a basic level (e.g. handshaking).
- is motivated to protect his or her privacy and security.

3.3.4 Usability – Measurement

Evaluating these criteria accurately would require a lengthy and quantitative study. As Ben Shneiderman has put it:

> "Every designer would like to succeed in every category,
> but there are often forced trade-offs" [6].

In our case, several trade-offs had to be done, including the limited usability for users with little to no computer experience or the quantitativeness and accuracy of the usability evaluation. So, in order to measure the usability, a *subjective* score (S) between 1 and 10 will be given to each of the before mentioned five pillars for each user interface implementation respectively in the Conclusive Comparison section.

3.3.5 Mapping Functional Requirements to UI Elements

The following table lists the functional requirements according to Project Requirements Document (PRD) and maps those to commonly used UI elements to display those functions:

Functional Requirement	UI Elements	Details
Register and Login user account	Label	e.g. “Please Sign Up”
	Input Field	Username, Passphrase
	Button	Sign In-, Sign Up-Button
	Window	Error Dialogue
Delete the registered user account	Button	Delete-Button
	Window	Confirmation Dialogue
Add a contact	Label	e.g. “Enter peer handshake link”
	Button	Start Handshake
	Field	Peer Handshake-Link-Button
Delete a contact	Button	Deletion-Button
	Window	Confirmation Dialogue
Send and Receive text messages with a contact	List Item	Sent and Received Messages
	Input Field	Message-Box
	Button	Send-Button

While on the GUI, an optional requirement is to receive tray notifications when the application is minimized or unfocused.

3.3.6 Elements of the GUI – WIMP

> “WIMP can be implemented in a text-based system,
> but graphical displays are often assumed.”
>
> - Ashley George Taylor, WIMP Interfaces [7]

The predominant elements of a GUI can be summarized using the acronym “WIMP”, which stands for **W**indows, **I**cons, **M**enus (including Controls) and **P**ointer.

Windows are the containers of all GUI components. Depending on the used window manager, windows are usually floating, but can also be stacked or tiled. They are used to display and compartmentalize information, but also to guide the user’s attention (for example, a warning dialogue) using focusing techniques. For the GUI of this project, there will be two larger windows (main window and add contact dialogue) and two smaller, but modal windows (error/info dialogue, change alias dialogue).

Icons are usually small images[1], which are used to represent functions of GUI components and are often adjacent to the textual description of a function. The usage of icons is encouraged, since icons can be quickly recognised (*familiarity*) by the user and therefore allow for a better usability.

1 Sometimes fonts, vector graphics or other visual depictions are used.

Menus allow for another familiar way to navigate to common application functions. The GUI of this project will utilise the so-called pull-down menu, which most desktop applications have on the top of the window and the context-menu. It is invoked by pressing the right mouse button and allows extended functionality on the selected object. The menus which a user interacts with (through direct manipulation) are referred to as graphical control elements or widgets. They include but are not limited to buttons, text fields, input fields, radio buttons, checkboxes, et cetera.

The Pointer is also called the Mouse Pointer or Cursor. By default it is managed by most window managers or desktop environments. An additional element of GUIs which is not a part of the WIMP paradigm, but can increase user satisfaction is the use of focused GUI components. By highlighting components while the mouse hovers on them, the likelihood to miss the intended component to click decreases. The terminal user interface does not require a pointer. It's noteworthy, that originally, the P in WIMP meant Pointing Device, which also includes trackballs, trackpads, isopointers, pens and more.

3.3.7 Technical Requirements – Peripheral Hardware

The following table compares the peripheral hardware devices which are necessary for each user interface respectively. Note, that this table does not include the necessary technical specifications of the computer.

Type of Peripheral Hardware	Terminal User Interface	Graphical User Interface
Display Output Device	Computer Terminal (VDU, VDT, VTTY, Emulators, etc.)	Computer Monitor (TFT-LCD, CCFL, etc.)
Input Device	Any Computer Keyboard	
Pointing Device	Not necessary	Any Computer Mouse

3.3.8 Technical Requirements – Operating System and Software

The following table compares the necessary software component to use each user interface respectively.

Type of Software Component	Terminal User Interface	Graphical User Interface
Operating System	Any GNU/Linux Operating System (Tested on Fedora 34 and Debian 10)	
Kernel	Linux (Tested on 5.16.9)	
Runtime Dependency	Java Development Kit Version 17	
Display Server	Not necessary (VTTY is pre-installed on most GNU/Linux Systems)	Xorg or Wayland (Tested on Xorg)

3.3.9 Technical Requirements – User Interface Framework

A UI Framework (also called Toolkit) allows software developers to use UI control elements, but also often include their own rendering engines and possibilities to layout an application in familiar ways, which decreases the complexity for the developer and reduced the necessary time to implement a user interface.

In order to select the appropriate framework for our project, several requirements have been declared in the following table and given a weight w (between 1 and 5), which is multiplied with an integer m between 5 (meaning that the requirement is completely met) and 0 (requirement is not met at all). The sum of all products is then the score s, the framework which achieved the highest score will be chosen. The calculations are done in the following sections.

Requirement	Details	Weight (w)
Actively maintained	Last update after February 2019	5
Well documented	Preferably Javadoc or similar	5
Independent of native library	Pure Java, no calls to low-level UI code	3
Has Gradle dependency or plug-in	Increases the maintainability of our project	4

3.3.10 Adherence to Limitations – Terminal User Interface

The text-based user interface has less software and peripheral hardware requirements, but is limited in other ways.

- Windows: Since the TUI runs within a terminal, the number of floating windows is either zero (e.g. VTTY) or one (e.g. a terminal emulator). Note, that within the TUI, the concept of windows can still exists, but is independent and not as flexible as a window, which is managed by a desktop environment or window manager.
- Icons: Implementing visual depictions will not allow for the usage (decrease usability) on VTTY and can therefore be omitted. A possible work-around would be to implement ASCII depictions, but these are out of scope for this project.
- Menu: Without a pointer, there is no feasible way to implement a context-menu. Other controls can be implemented as long as the used framework allows it.
- Pointer: There is no pointer required. The user navigates using a computer keyboard.
- Tray Notifications: Implementing tray notifications requires the usage of a display server, which is not included in the requirements of the TUI. Tray notifications are only feasible for the GUI.

3.4 User Interface Frameworks – Terminal User Interface

The sections deals with the evaluation of considered and used user interface frameworks for the TUI using a simple scoring system as described in Technical Requirements – User Interface Framework.

3.4.1 Considered Frameworks

Java Curses

JCurses is a TUI library consisting of two parts:

- Platform independent: Only Java classes which allow for the creation a TUI in a AWT-like style (though it's not based on AWT).
- Platform dependent: Native shared library in C, which makes primitive input/output operations to the UNIX curses window system.

It is built upon GNU ncurses. It supports both Windows and GNU/Linux. Apart from having an installation and compilation guide, there is no official documentation to be found. Using JCurses requires the installation of a curses implementation (e.g. libjcurses.dll on Win32 and libcurses.so on Linux).

Requirement	Weight (*w*)	Factor (*m*)	Score (*s*)
Actively maintained	5	**0**	**0**
Well documented	5	**1**	**5**
Independent of native library	3	**1**	**3**
Has Gradle dependency or plug-in	4	**0**	**0**
		Total Score	**8**

CHARVA

CHARVA is based on Swing, but is also not pure Java. It is composed of a Java library for the implementation of graphical widgets and a dynamically-loaded shared library written in C which is linked with the GNU ncurses library.

Requirement	Weight (*w*)	Factor (*m*)	Score (*s*)
Actively maintained	5	**0**	**0**
Well documented	5	**5**	**25**
Independent of native library	3	**1**	**3**
Has Gradle dependency or plug-in	4	**0**	**0**
		Total Score	**28**

3.4.2 Used Framework - Lanterna

Lanterna is visually very similar to ncurses, but is based on Swing and is completely independent from native UI libraries. The Lanterna API automatically recognises if a display server is running and uses the Swing terminal emulator instead of standard output if necessary. It is structured in three layers:

- Low-Level Terminal Interface
- Buffered Screen
- Text GUI (also called GUI Toolkit)

This allows development in various layers depending on the requirements.

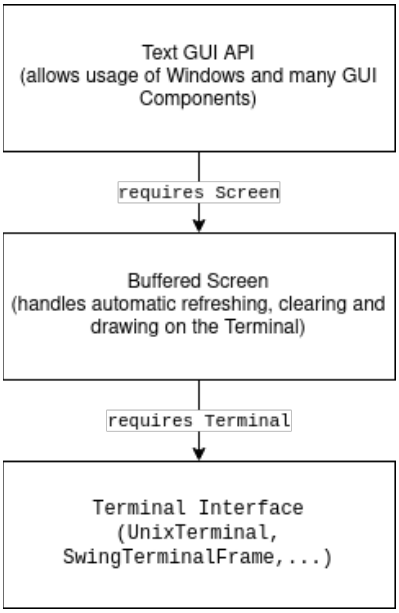


Figure 3.3: *Layers of the Lanterna API*

Requirement	**Weight (*w*)**	**Factor (*m*)**	**Score (*s*)**
Actively maintained	5	**3**	**15**
Well documented	5	**5**	**25**
Independent of native library	3	**5**	**15**
Has Gradle dependency or plug-in	4	**5**	**20**
		Total Score	**75**

3.5 User Interface Frameworks – Graphical User Interface

The sections deals with the evaluation of considered and used user interface frameworks for the GUI using a simple scoring system as described in Technical Requirements – User Interface Framework.

3.5.1 Considered Frameworks

Abstract Window Toolkit (AWT)

The oldest Java GUI framework is also the most mature, tested and documented one in this case. AWT provides an abstraction layer over the native operating system user interface. This implies, that the designs and behaviours (*look and feel*) of the native (OS) GUI are used, which some developers and users prefer. Current Java UI developers discourage the usage of AWT, not only because a lot of tools used to develop modern user interfaces are not implemented in AWT, but also because the old code base has a complicated structure while being intermingled with other low-level parts of JDK such as 2D, i18n, and Input Methods [8]. It has gotten the status of being a legacy framework [9].

Requirement	Weight (*w*)	Factor (*m*)	Score (*s*)
Actively maintained	5	**0**	**0**
Well documented	5	**5**	**25**
Independent of native library	3	**0**	**0**
Has Gradle dependency or plug-in	4	**0**	**0**
		Total Score	**25**

Swing

Swing is based on AWT and provides additional functionalities used by many developers (e.g. tabbed panels, scroll panes, list views, etc. [10]). Swing has its own look and feel, which is homogenous on all supported platforms. All components are entirely written in Java (unlike AWT, which implements platform-specific code), which makes it platform independent and very flexible. The toolkit renders controls using Java 2D APIs instead of using the API of the native GUI.

Requirement	Weight (*w*)	Factor (*m*)	Score (*s*)
Actively maintained	5	**0**	**0**
Well documented	5	**5**	**25**
Independent of native library	3	**5**	**15**
Has Gradle dependency or plug-in	4	**2**[2]	**8**
		Total Score	**40**

2 *m* = 2, there are third-party libraries which allow Swing to be used as a Gradle dependency.

3.5.2 Used Framework - JavaFX

Swing and AWT were intended to be replaced by JavaFX as the newer standard GUI library for Java, but JavaFX didn't took a significant rise in the market share of used Java UI frameworks. One of the reasons for this is likely since JavaFX was removed from the Java SE (Standard Edition) with the inclusion of modules while Swing and AWT remained included. Another reason was the shift of user interest towards web first and mobile first applications [11]. Nevertheless, since JavaFX is based on Swing (high flexibility) and has several extensions (such as WebView, JavaFX Mobile, FXML, CSS, et cetera), but also because of the extensive documentation, we have decided to use this framework.

Requirement	**Weight (*w*)**	**Factor (*m*)**	**Score (*s*)**
Actively maintained	5	**5**	**25**
Well documented	5	**5**	**25**
Independent of native library	3	**5**	**15**
Has Gradle dependency or plug-in	4	**5**	**20**
		Total Score	**85**

3.6 Approaches to Solutions – Terminal User Interface

In this section we will go through the possible approaches into developing an application with the used framework Lanterna and explain why certain approaches are less suitable and others are more for our project.

3.6.1 Considered Approaches

Direct Terminal Access

The lowest layer of the Lanterna API is the Terminal interface. Using the `DefaultTerminalFactory` it is possible to create an auto-detected Terminal implementation – for example:

- If the application runs on the command-line of a UNIX-based System: `UnixTerminal`
- If a running display server is detected: `SwingTerminalFrame`

Creating a Terminal using the `DefaultTerminalFactory` is as simple as

```
Terminal terminal = new DefaultTerminalFactory().createTerminal();
```

This development approach allows for highest level of granularity, it allows to

- Move the cursor
  ```
  terminal.setCursorPosition(20, 2);
  ```
- Printing characters to the Terminal
  ```
  terminal.putCharacter('H');     // one character at a time only!
  ```
- Modifying the Terminal colours (ANSI standard)
  ```
  terminal.setBackgroundColor(TextColor.ANSI.BLUE);
  ```
- Modifying the text style of the Terminal
  ```
  terminal.enableSGR(SGR.BOLD);
  ```
- Polling (non-blocking) and reading (blocking) keyboard input

Direct terminal access is inappropriate for our use-case, since it would require us to write several wrapper classes to allow for basic functionality (e.g. displaying multi-line text, text-input, password input, manual screen clearing). However, this does not exclude the usage of this approach entirely, since modifications of colours and text styles are part of this project.

Buffered Screen API

The job of the buffered screen API is to redraw only certain parts of a screen, when only certain parts changed, instead of drawing the whole screen. After creating a Terminal, using a Screen is as simple as

```
Screen screen = new DefaultTerminalFactory().createScreen();
screen.startScreen();
// TUI logic here
screen.stopScreen();
```

While being able to make use of the features of the Buffered Screen API between the invocation of the start and stop method. The Buffered Screen API allows for the following

- Drawing text (not only single characters) on the Screen
  ```
  TextGraphics textGraphics = screen.newTextGraphics();
  textGraphics.putString(10, 5, "Hello Lanterna!");
  ```
- Clearing the Screen
  ```
  screen.clear();
  ```
- Refreshing the Screen
  ```
  screen.refresh();
  ```
- Handling Terminal resize
  ```
  TerminalSize newSize = screen.doResizeIfNecessary();
  if(newSize != null) { terminalSize = newSize; }
  ```
- Polling / Reading keyboard input (requires `InputProvider` methods like `pollInput()` and `readInput()`)

While this approach has certainly more use-cases than the direct terminal access, it still does not fulfil many requirements for this project. Although there are some cases, where manually refreshing or clearing the screen becomes necessary, using this approach will also create too much redundancy, since there would still have to be wrapper classes (e.g. for error/confirmation dialogues, list views, etc.).

3.6.2 Used Approach - Text GUI Toolkit

The `TextGUI` interface allows for the usage of several GUI elements according to WIMP. The usage of this API is very similar to using AWT, Swing or JavaFX, since it's based on Swing. Instead of a `TextGUI`, a `WindowBasedTextGUI` should be created, since currently there is only one concrete implementation of the `TextGUI` (`MultiWindowTextGUI`).

```
WindowBasedTextGUI gui = new MultiWindowTextGUI(screen);
```

Note that a Screen is required to create a `TextGUI`. The `TextGUI` allows the usage of several GUI components, such as

- Windows (and Windows Hints)
  Note, this doesn't conflict with the Windows mentioned in Adherence to Limitations.
- LayoutManagers (`LinearLayout`, `GridLayout`, etc.)
- Panels
- Labels, Buttons, TextBoxes, CheckBoxes, Tables, ActionListsBoxes, etc.

Using this approach greatly reduces the complexity and redundancy of the TUI code, due to the possibilities to layout and structure the interface with LayoutManagers and Panels, but also since there is no need the create wrapper classes for basic functionality like password input or listing messages.

3.7 Results – Terminal User Interface

With the approaches taken into consideration, this section will go into the detail of certain development aspects of the TUI which are considerably different from the GUI.

3.7.1 Initialisation

Importing Lanterna into our project can be done using Gradle dependencies. It is as simple as adding the line

```
implementation 'com.googlecode.lanterna:lanterna:3.1.1'
```

to the dependencies section in the `build.gradle` configuration file.

The Briar Project uses Dagger2 to utilize dependency injection. Every project which forks Briar's source code has to use Dagger2 too, mainly to configure the forked client and starting up. In our case, we have used it to create the main UI classes too (called MainTUI and MainGUI, respectively).

In Java's main class, a new MainTUI object reference has to be created specifically when the command-line option –tui or tui has been passed. You can see in the following code that we use Dagger2 (the variable `briarJarApp` is built on compile-time) to get the instance of the MainTUI.

```
MainTUI mainTUI = briarJarApp.getMainTUI();
mainTUI.start();
```

3.7.2 Accessing the BriarJar API and Event Handling

The BriarJar API provides features and functionality from Briar. We have always used dependency injection – specifically constructor injection – to get the object references of the view models (BriarJar API classes). This made both UIs much more consistent.

```
@Inject
public AddContact(ContactViewModel cvm)
{
        this.cvm = cvm;
        init();
}
```

For event handling, the `EventListenerViewModel` from the BriarJar API has to be extended in the TUI classes, while an `EventBus` was injected into the constructor using dependency injection.

3.7.3 TUI Helper Class

In order to avoid duplicated code ("Don't Repeat Yourself"), a helper class called TUIUtils (TUI Utilities) has been created. There are three main objectives of the TUI helper class

- Some TUI code is used in many or all TUI classes, it should be in the TUIUtils.

- TUIUtils initialises all TUI classes on startup, holding a reference of all TUI classes. This makes a lot of handling much more easier, but may have some security concerns.
- TUIUtils handles switching windows (and clearing the screen properly before switching).

The helper class is instantiated using dependency injection in the MainTUI class. Since the helper class itself uses constructor injection to create all TUI classes and all other classes require a reference of the helper class, it is manually passed (using setters) in every other TUI class – instead of injecting the TUIUtils reference in every class – to avoid a cyclic dependency injection graph (which causes a compilation error).

```
@Inject
public TUIUtils(SignIn signIn,…) //  signIn is injected using constructor
                                     injection
{
        this.signIn = signIn;
        signIn.setTuiUtils(this); // signIn gets the reference with a setter
        ...
}
```

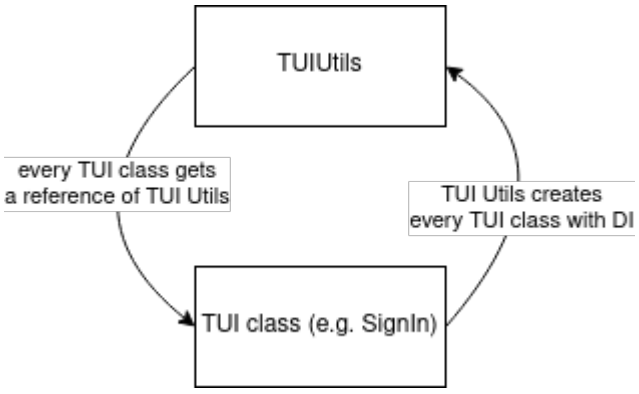


Figure 3.4: *Cyclic Dependency Injection Graph*

3.7.4 Structure of Classes

The TUI classes have the following structure:

- Initialisation: Initialisation of (most) variables (in constructor).
- Create Window: Create the window (using a render method).
- Panel Remover: Remove all panels (elements) from the window.
- Updater: Update a panel (e.g. a chat or contact list).
- Logic: UI-related logic code.
- Setters: Used to set the TUIUtils for example.
- Events: The `eventOccured(Event e)` method reacts to events from Briar.

Understanding the different segments a TUI class is composed of will be necessary highlight the difference between TUI and GUI development.

3.7.5 Clipboard – Accessing AWT in the TUI

In order to add a new contact, users have to exchange their handshake-links. Since it's inconvenient to type-out a Briar URI, the TUI allows the user to copy the link to the clipboard. Since Lanterna is based on Swing, which in return is based on AWT, it is possible to import the package "`java.awt.*`" to use the Clipboard API.

```
StringSelection selection = new StringSelection(ownLink);
Clipboard clipboard = Toolkit.getDefaultToolkit().getSystemClipboard();
clipboard.setContents(selection, selection);
```

This API requires the usage of a display server because there would not be a clipboard without a display server. Since the TUI is designed for usage without display server, possible errors have been handled here. If the user does not have a display server running, the user will be prompted to type-out the URI manually.

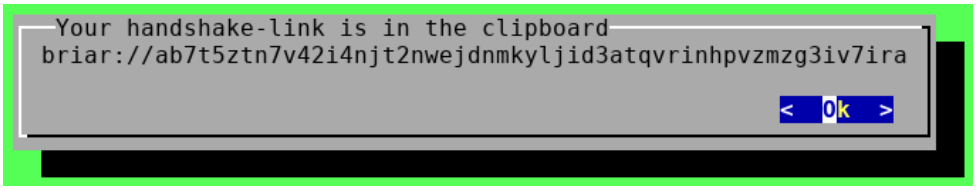


Figure 3.5: *Copying handshake links using underlying API functionality*

3.8 Approaches to Solutions – Graphical User Interface

Identical to the sections for the TUI, this section will go through the possible approaches into developing an application with the used framework JavaFX and explain why certain approaches are less suitable and others are more for our project.

3.8.1 Considered Approach - Declarative Programming

Since JavaFX Version 2, it's possible to write the GUI using FXML – an XML-based mark-up language. It allows the complete abstraction of program design from logic code. It is also an example of domain-specific declarative programming, for example:

```
<?import javafx.scene.control.Label?>
<Label text="Hello, World!"/>
```

This creates a label with the text "Hello, World!".

FMXL has gained popularity in combination with the SceneBuilder, a software which generates the inline scene graph of the UI in FXML. It allows for more efficient UI development, as well as the easier creation of layouts and structures for complex UIs. It is noteworthy that declarative and imperative programming is not mutually exclusive in the case of JavaFX.

3.8.2 Used Approaches - Imperative Programming

Calling methods and classes from the JavaFX API directly is referred to as imperative programming. It is the traditionally the more verbose way to create a UI, but is often used for long-term JavaFX projects. It allows for higher flexibility and a more dynamic UI than the declarative way.

- Flexibility: Every UI object reference can be changed with another.
- Dynamic: Instantiation of object references, which are not possible with FXML and SceneBuilder, is possible.

Despite the higher code verbosity, we have decided to use the imperative approach, because of the easier handling of events which are caused by the BriarJar API and handled within the UI code.

3.9 Results – Graphical User Interface

This sections will – just like the respective Results for the TUI section – go into the details of various development aspects which differ considerably from the TUI.

3.9.1 Initialisation

Importing Lanterna into our project can be done using Gradle dependencies. It is as simple as adding the line

```
id 'org.openjfx.javafxplugin' version '0.0.9'
```

to the plug-ins section and an additional dependency for material design

```
implementation group: 'com.jfoenix', name: 'jfoenix', version: '9.0.10'
```

to the dependencies section in the build.gradle configuration file. For the GUI, further configuration is necessary, for example

```
javafx {
    version = "17"
    modules = [ 'javafx.controls', 'javafx.fxml' ]
}
```

configures the used JavaFX modules. And the following code:

```
run {
    jvmArgs = [
            '--module-path', classpath.asPath,
            '--add-modules', 'javafx.controls',
            '--add-modules', 'javafx.fxml',
            '--add-modules', 'javafx.graphics',
            '--add-modules', 'javafx.base',
            '--add-opens', 'java.base/java.lang.reflect=ALL-UNNAMED'
       ]
 }
```

makes sure that necessary JavaFX module parameters (JVM arguments) are passed on runtime.

In the main class of the application, starting the GUI is not as easy as simply creating an object reference to the MainGUI class and rendering the user interface (like in Lanterna). JavaFX creates a GUI thread on startup which has to be initialised manually to start the GUI from another class.

```
// This is a so-called lambda expression which essentially ‘runs’ the indented
code after the arrow (->) in the JavaFX thread (simplified!).
Platform.startup(() -> {
                                MainGUI mainGUI = briarJarApp.getMainGUI();
                                mainGUI.init();

                                // a ‘Stage’ in JavaFX is a Window in WIPM
                                Stage stage = new Stage();
                                mainGUI.start(stage);

                            });
```

3.9.2 Material Design

The GUI of this project uses the Briar Styleguide [12] as a guideline, which inherits many design choices from Material Design [13]. Note, that these guidelines are not strictly enforced. To adhere to these guidelines, we have used a JavaFX extension library called JFoenix. JFoenix provides JavaFX nodes (GUI elements) but with a material look and feel. These nodes have the same naming convention as the JavaFX API, but are always prefixed with JFX, for example:

```
// for the elements in the dialogue
JFXDialogLayout dialogLayout = new JFXDialogLayout();

// create a new dialogue, add it to the background pane (root) and add a
transition
JFXDialog dialog = new JFXDialog(root, dialogLayout,
                                 JFXDialog.DialogTransition.TOP);
JFXButton button = new JFXButton("Okay");
```

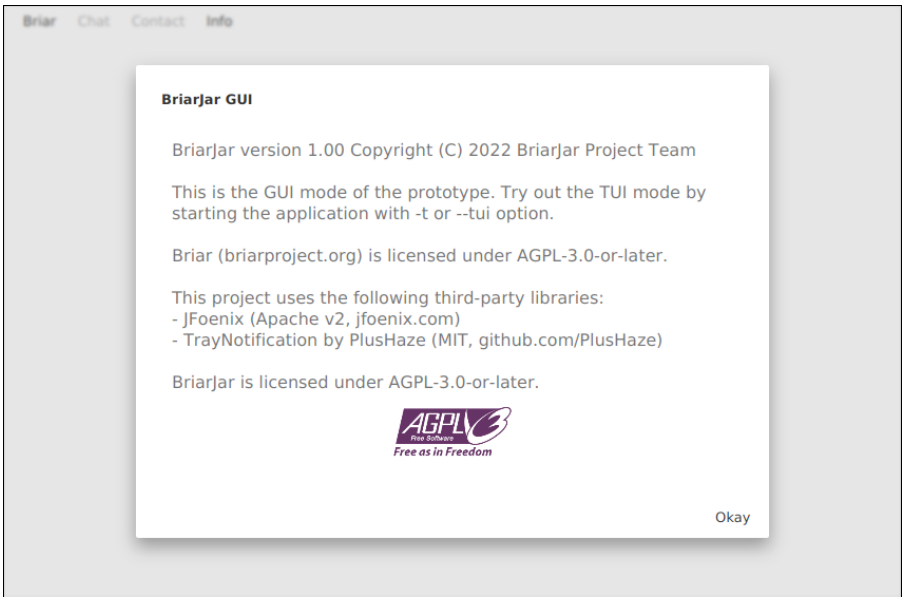


Figure 3.6: *Material Design Dialogue with JFoenix*

3.9.3 Accessing the BriarJar API and Event Handling

While the access to the BriarJar API is not much different from the TUI (constructor injection), however event handling is. Since all GUI classes are structurally different from the TUI classes (they are subclasses for other GUI nodes, e.g. RootBorderPane is a subclass of BorderPane), the EventListener interface from BriarJar has to be implemented directly. The reason for that is that Java does not support multiple inheritance, but multiple interfaces can be implemented.

3.9.4 Structure of Classes

The GUI classes have the following structure:

- Initialise Components: Initialisation of (most) variables (in constructor).
- Add Components: Add all components to the underlying pane.
- Add Handlers: Every object instance which reacts to certain user interaction gets handlers.
- Create Window: Create the window (by invoking the JavaFX show() method).
- Updater: Update a panel (e.g. a chat or contact list).
- Logic: UI-related logic code (e.g. when adding a new message to the message list).
- Setters: Used to set the TUIUtils for example.
- Events: The `eventOccured(Event e)` method reacts to events from Briar.

3.9.5 Differences in Main Screen Layout

The GUI layout shows the contact list and message pane side-by-side, which is rather easy to implement using the JavaFX BorderPane, VBox (Vertical Box) and ListView.

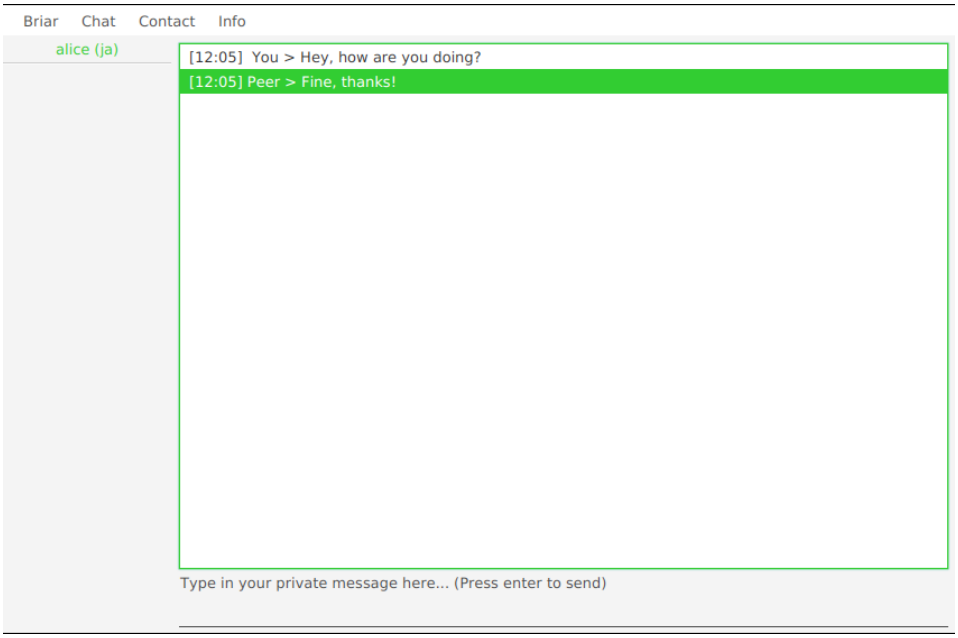


Figure 3.7: *Side-by-side Contact List and Message View in the GUI*

The TUI does technically also allow for such a constellation but because there is no BorderPane or any similar layout in the Lanterna API. This would require us to create our own layout, which was not manageable within the given time frame.

3.10 Conclusive Comparison – Usability and Development

This section will compare the differences in development of Lanterna and JavaFX, but also with the usability of the resulting UI implementations from a users perspective.

3.10.1 From a Developers Perspective

A developer, who has only written Swing or JavaFX GUI code before, can easily integrate into the TUI paradigm, provided he uses a well documented framework.

Arguably, UI creation with frameworks in Java does not have a very steep learning curve, once any Java UI framework has been studied thoroughly. Many frameworks in Java are either based on some older well-known UI framework (e.g. Lanterna is based on Swing) or aim to make the development familiar by making their API similar to another well-known in the Java world (e.g. JCurses is AWT-like).

The following table highlights all core differences of both used frameworks.

Aspect of Development	TUI Framework – Lanterna	GUI Framework – JavaFX
Importing in project	Possible by downloading the pre-compiled jar files, using Gradle or using Maven	Many possibilities with much more detailed documentation on the Gluon website
Threading	Runs on the same thread as the call to initialise the Terminal	Runs on a separate JavaFX thread
Java Modularity[3]	Not enforced	Enforced (but can be worked-around)
Layout Managers	Absolute Layout, Linear Layout, Border Layout, Grid Layout	AnchorPane, BorderPane, DialogPane, FlowPane, GridPane, Hbox, StackPane, TextFlow, TilePane, VBox
Structure	It is necessary to render components, but also to remove components when switching windows	JavaFX does not have any issues with remaining components when switching windows, since nothing is drawn on a Terminal

This table will be discussed further in the Conclusion.

3.10.2 From a Users Perspective

As already defined in the Non-Functional Requirements, our goal here is to compare the *usability* of both user interface implementations, which consists of five main pillars according to Ben Shneiderman (see the chapter Usability – Measurement for details on how the score is determined).

- Time to learn
- Speed of performance
- Rate of errors by users
- Retention over time
- Subjective satisfaction

3 A Java Module packages applications into smaller parts which can be compiled as their own jar file. Oracle recommends modularity since it's a feature in Java 9 and above.

Time to learn

"How long does it take for typical members of the user community to learn how to use the actions relevant to a set of tasks?"

The target user takes less time to learn to GUI than the TUI. This can be explained with the missing aspect of familiarity, but since the TUI implementation requires more steps to open the chat window (5 steps), while on the GUI (2 steps) the contact list and chat window are side-by-side.

Terminal User Interface Score (S)	Graphical User Interface Score (S)
6	8

Speed of Performance

"How long does it take to carry out the benchmark tasks?"

While there is no clearly defined benchmark task, we can assume a scenario where a target user has to create a Briar account and add an existing Briar user (a friend). Both user interfaces requires 11 steps to carry out this task, meaning they both fall equal in this category.

Terminal User Interface Score (S)	Graphical User Interface Score (S)
8	8

Rate of Errors by Users.

"How many and what kinds of errors do people make
in carrying out the benchmark tasks?"

The chat window in the TUI requires the usage of the arrow keys to select the send-button, while there is no send button in the GUI (but rather a small text hinting the user to press the enter key to send). It is often mistakenly assumed to press enter to send in the TUI too.

Another reoccurring error in the TUI usage is that after opening message metadata, the user can easily forget which buttons to press to get back to the message box (two times the arrow down key). This makes the TUI implementation *inconvenient*.

Terminal User Interface Score (S)	Graphical User Interface Score (S)
3	9

Retention over Time

"How well do users maintain their knowledge after an hour, a day, or a week?"

Since the functionality of the prototype is limited, it is very easy to remember the necessary steps to perform certain tasks – even after a longer period of time. Bear in mind, that in this case there is no way to determine time periods longer than two weeks.

Terminal User Interface Score (S)	Graphical User Interface Score (S)
7	9

Subjective Satisfaction

"How much did users like using various aspects of the interface?"

While this one is the most subjective criteria, the score has been based on all collected reactions to both user interfaces from our peers, but also friends and family members.

Terminal User Interface Score (S)	Graphical User Interface Score (S)
8	9

Total Scores

The following table shows all summed-up total scores. The meaning of these results will be elaborated in the Conclusion section.

Terminal User Interface Total Score	Graphical User Interface Total Score
32	**43**

3.11 Conclusion

It can be concluded that there are not many differences for a UI developer when using the user interface frameworks *JavaFX* and *Lanterna*, however the core differences – as highlighted in the Conclusive Comparison – have to be taken into consideration. The limited number of layout managers and the structure in TUI can make it more time extensive to develop convenient TUI applications, but it is neither impossible to extend the TUI framework with another layout manager nor are the existing layout managers insufficient for general usage.

From the perspective of the user, our usability evaluation results show that the graphical user interface meets the requirement *usability* much more than the terminal user interface. While the TUI has lower peripheral hardware requirements and therefore a generally higher accessibility, our TUI implementation is more inconvenient than the GUI implementation. It's possible to improve it with the usage of self-implemented layout managers, but since this was neither a project requirement nor feasible to implement and thoroughly test within the given time frame, we have omitted this improvement.

3.12 Bibliography

[1] http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.136.3720, (web.archive.org), 2022-01-01
[2] "Designing the User Interface" (2004) by Ben Shneiderman and Catherine Plaisant, Page 44
[3] https://www.iso.org/standard/16873.html (web.archive.org), 2022-02-24
[4] "Designing the User Interface" (2004) by Ben Shneiderman and Catherine Plaisant, Page 16
[5] http://www.asktog.com/papers/raskinintuit.html (web.archive.org), 2022-02-24
[6] "Designing the User Interface" (2004) by Ben Shneiderman and Catherine Plaisant, Page 16
[7] http://www-static.cc.gatech.edu/classes/cs6751_97_winter/Topics/dialog-wimp/ (web.archive.org), 2022-02-25
[8] https://openjdk.java.net/groups/awt/ (web.archive.org), 2022-02-26
[9] https://coderslegacy.com/java-gui-frameworks-guide/ (web.archive.org), 2022-02-26
[10] https://cs.nyu.edu/~yap/classes/visual/03s/lect/l7/ (web.archive.org), 2022-02-26
[11] https://www.oracle.com/technetwork/java/javase/javaclientroadmapupdatev2020may-6548840.pdf (web.archive.org), 2022-02-26
[12] https://briar-styleguide.netlify.app (web.archive.org), 2022-03-08
[13] https://material.io (web.archive.org), 2022-03-08

3.13 Table of Figures

Figure 3.1: The efficiency of command-line applications depends on the understanding of their usage....48
Figure 3.2: Midnight Commander (mc) is a TUI-based shell for UNIX-like systems....48
Figure 3.3: Layers of the Lanterna API....55
Figure 3.4: Cyclic Dependency Injection Graph....61
Figure 3.5: Copying handshake links using underlying API functionality....62
Figure 3.6: Material Design Dialogue with JFoenix....65
Figure 3.7: Side-by-side Contact List and Message View in the GUI....66

4 Contents of Compact Disk (CD)

Content:

- This diploma thesis document (file type: pdf)
- BriarJar application
  - Version 1.0 (inclusive most recent commits up to 2022-03-16)
  - Executable BriarJar client (file type: jar, 36 MB)*
- BriarJar application UML diagrams (file types: svg, puml)
  - Activity
  - Use-case
- BriarJar API documentation (generated html pages)
  - Compressed archive (file type: 7z, 5.7M)
- Source code of BriarJar with Briar submodule
  - Compressed archive (file type: 7z, 9.6 MB)

*** Data Requirements**

- OpenJDK 17
- Debian GNU/Linux or a similar OS
- x86_64 Bit Architecture
- CPU: Pentium 4, 1 GHz (minimum)
- RAM: 1 Gigabyte (minimum)
- Free Disk Space: 100 MB (minimum)

5 Attachments

- Project Handbook (includes all project development related documents)

SPENGERGASSE
ausbildung mit zukunft

BRIAR

Project

BriarJar

Project Handbook (PHB)

Version	1.01
Date	2022-03-30

Confidentiality	**[ ] Internal [x] External**
File Name	BriarJar_PHB_20220330_002

Project Partner	Briar Project
Project Supervisor	Andreas Chwatal

Project Team	Farman Khan	UI/UX Architect & Engineer
	Alexander Zeidler	Project Leader, Software Architect & Engineer

Document Revision History

Version	Date	Name(s)	Change Description
1.00	2022-03-30	FK, AZ	Release version 1.00
1.01	2022-06-27	AZ	Add "Thesis Defence Presentation" to document "Status Presentations"
Farman Khan (FK), Alexander Zeidler (AZ)			

Project Application Data

Initial Position

Briar is a free open-source messenger that is not only accessible to everyone, but can also be extended as desired. Currently, there is neither a programming interface designed for Java developers nor a full-featured desktop user interface. Scope of this work is the development of a terminal and a graphical user interface for the Briar project.

Project Objectives

The aim of the BriarJar project is to provide the client with a prototype of a simple messenger that can be run both with a graphical user interface and within a terminal. The messenger functionalities consist of the registration of an account, a contact management as well as a text communication with a desired conversation partner.

Individual Topics

Farman Khan	From Terminal to Graphical: Comparison of two different User Interface Frameworks in Java
Alexander Zeidler	Software Architecture and Implementation of Peer-to-Peer Applications

Research interests of the individual topics

Farman Khan	• Comparison of the development of graphical and terminal user interfaces in Java • User interface implementation (GUI/TUI)
Alexander Zeidler	• Review and comparison of architectural approaches for peer-to-peer networking applications • Development of an interface layer for connecting the core code to a user interface

Supervisor	DI Dr. Andreas Chwatal
Project Partner	Briar Project (i.V. Nico Alt)
Status as of	Sept. 2021

Table of Contents

The project handbook includes all project management documents created within the BriarJar diploma project.

Abbreviation	Document (EN)	Dokument (DE)	No. of Pages
PA	Project Acceptance	Projekt-Abnahme	1
PRD	Product Requirements Document	Lastenheft	12
SRS	Software Requirements Specification	Pflichtenheft	17
TC	Test Cases	Testfälle	10
SR	Status Reports	Status-Berichte	15
SP	Status Presentations	Status-Präsentationen	39
MP	Meeting Protocols	Kommunikations-Protokolle	6
TL	Time Log	Zeiterfassung	8

Projektabnahme

SPENGERGASSE
ausbildung mit zukunft

Projektname:	Projekt Nr./Kurztitel:
BriarJar - A GNU/Linux Desktop Client for Briar	BriarJar

Auftraggeber (vertreten durch):	Auftragnehmer (vertreten durch):
Briar Project (Nico Alt)	BriarJar (Farman Khan, Alexander Zeidler (PL))
Tag der Abnahme:	**Abgenommene Version (optional):**
16. März 2022	1.00 (2022-03-13)

Bei der heute erfolgten gemeinsamen Abnahme wurde festgestellt, dass sich die erbrachte Leistung bis auf die nachfolgend bezeichneten Mängel (lt. Fehlerklassenbeschreibung siehe Anhang), im vertragsgemäß zu erstellenden Zustand befand.

Mängel / Fehler (Zusammenfassung)	Behebungsfrist	Klasse

Bemerkungen und Vorbehalte:

Es war ein Vergnügen, mit Khan und Alex zusammen zu arbeiten, die mit ihrer Professionalität und eigenständiger Arbeitsweise das gesamte Briar-Team beeindruckt haben. Wir wünschen für die Zukunft alles Gute!

Gesamtprodukt abgenommen (zutreffendes ankreuzen):

JA	EINGESCHRÄNKT	NEIN

16.03.2022 [Unterschrift entfernt]
Datum & Unterschrift (Auftraggeber)

[Unterschrift entfernt]
Datum & Unterschrift (Auftragnehmer)

Erläuterung Fehlerklassen: Im Zuge der Abnahme werden eventuell bestehende Mängel in Fehlerklassen eingeteilt.
Fehlerklasse 1: Die zweckmäßige (wirtschaftlich sinnvolle) Nutzung ist durch solche Fehler nicht möglich oder unzumutbar eingeschränkt.
Fehlerklasse 2: Die zweckmäßige Nutzung ist nicht so weit beeinträchtigt, dass das System nicht dennoch verwendet werden könnte, allenfalls unter Einbeziehung zwischenzeitlichen Fehlerumgehungsroutinen. Diese Fehler werden, so weit wie möglich, während des Abnahmetests behoben.
Fehlerklasse 3: Die zweckmäßige Nutzung ist durch diese Fehler nicht oder nur unwesentlich eingeschränkt.

Die Zuordnung der Fehler in eine der obigen Fehlerklassen erfolgt einvernehmlich zwischen den Vertragspartnern. Bei Fehlern der Fehlerklasse 1 handelt es sich um "erhebliche Abweichungen", bei Fehlern der Fehlerklasse 2 und 3 um "unerhebliche Abweichungen". Bei erheblichen Abweichungen wird der Test unmittelbar nach Behebung der Abweichung durchgeführt. Eine unerhebliche Abweichung berechtigt den Auftraggeber nicht zur Verweigerung der Abnahme. Nach der Abnahme verbleibende Fehler aller Fehlerklassen werden im Rahmen der Gewährleistung gemäß einem gemeinsam zu erstellenden Zeitplan behoben. Treten im Rahmen des Funktionstests keine wesentlichen Fehler (Fehlerklasse 1) auf, gilt das abzunehmende Projekt als abgenommen. Bei Auftreten von wesentlichen Mängeln (Fehlerklasse 1) ist nach Korrektur der Funktionstest zu wiederholen.

SPENGERGASSE
ausbildung mit zukunft

BRIAR

Project

BriarJar

Product Requirements Document

(Lastenheft)

Version	1.00
Date	2021-10-10

Confidentiality	**[ ] Internal [x] External**
File Name	BriarJar_PRD_20211010_008

Project Partner	Briar Project
Project Supervisor	Andreas Chwatal

Project Team	Farman Khan	UI/UX Architect & Engineer
	Alexander Zeidler	Project Leader, Software Architect & Engineer

Document Revision History

Version	Date	Name(s)	Change Description
0.01	2021-10-10	FK, AZ	Initialise document
0.02	2021-11-24	FK, AZ	Add/Update Preface, From the Current to the Desired State, Non-/Functional Requirements, Milestones, Software Data, System Architecture, User Interfaces, Scope of Delivery
0.03	2022-01-23	FK, AZ	Add Risk Assessment & Criteria of Approval; Update Glossary
0.04	2022-01-28	FK, AZ	Add Use-Case Diagram, Use-Case Descriptions
0.05	2022-02-19	AZ	Mainly update Use-Case Diagram
0.06	2022-02-24	FK, AZ	Update System Architecture Diagram
0.07	2022-03-16	FK, AZ	Clean up this document
1.00	2022-03-30	AZ	Release version 1.00
Farman Khan (FK), Alexander Zeidler (AZ)			

Table of Contents

1 Preface 4

2 From the Current to the Desired State 5

2.1 Current State 5

2.2 Desired State 5

2.3 Methods 5

3 Risk Assessment 6

4 Functional Requirements 6

4.1 Use-Case Diagram 7

4.2 Use-Case Descriptions 8

5 Non-Functional Requirements 10

6 Milestones 10

7 Software Data 11

8 System Architecture 11

9 User Interfaces 12

10 Scope of Delivery 12

11 Criteria of Approval 12

12 Glossary 12

1 Preface

According to the DIN 69901-5, the Product Requirements Document is defined as

> "the totality of the requirements for the deliveries and work performed by a contractor within an order, as determined by the client".

The Product Requirements Document thus usually describes *what* is to be done and *for what purpose*.

2 From the Current to the Desired State

2.1 Current State

Briar is a free open-source messenger which was initially released for the Android platform in 2018.

Some of its targets are that messages should be delivered reliable via peer-to-peer without getting the messages integrity being affected.

Although most of the existing Briar code is written in Java, currently there exists no graphical or terminal messenger client for desktop machines.

Further and up-to-date information can be found on Briar Project's official website.

2.2 Desired State

The aim of the BriarJar project is to provide the client with a prototype messenger written in Java as most of the Briar code is already. This gives our client feedback on how well such an implementation is currently working.

This should be a simple messenger that can be run on GNU/Linux desktop machines offering both a graphical user interface and terminal user interface which communicate with a new API for Java developers. The messenger functionalities are listed under "Functional Requirements".

The objective is to make the Briar messenger available to more people by developing the desktop prototype BriarJar instead of relying only on the Android platform. Nevertheless, the objectives within this diploma project are not to build a final product with reliable security for public nor should it be marketed as such.

2.3 Methods

Initially, it is required to analyse Briar's source code and fork/extend the relevant parts. This results in an API for Java developers which has to be designed, developed and documented.

With the built API it is possible to design and develop the desired GUI and TUI.

Finally, all relevant parts have to be combined and delivered as quoted under "Non-Functional Requirements".

3 Risk Assessment

The priority indicates approximately how frequent a risk might occur or how much negative impact it has on our project. It is sorted ascending, which means that the first priority has the highest impact and the last has the lowest.

Priority	Risk	Mitigations / Strategies
1	Lacking Time Resources	Plan ahead and use the available time on holidays.
2	Too High Standards (Over-Perfectionism)	Lower the standards until the requirements of the minimal viable prototype are met.
3	Little Knowledge or Experience on Subject Matter	Ask for support by the official Briar Project (Nico), our Supervisor or subject-related teachers.
4	Unknown/Unclear Tasks	Create a To-Do-List and keep regular project-internal meetings to check what has been/has to be done.

4 Functional Requirements

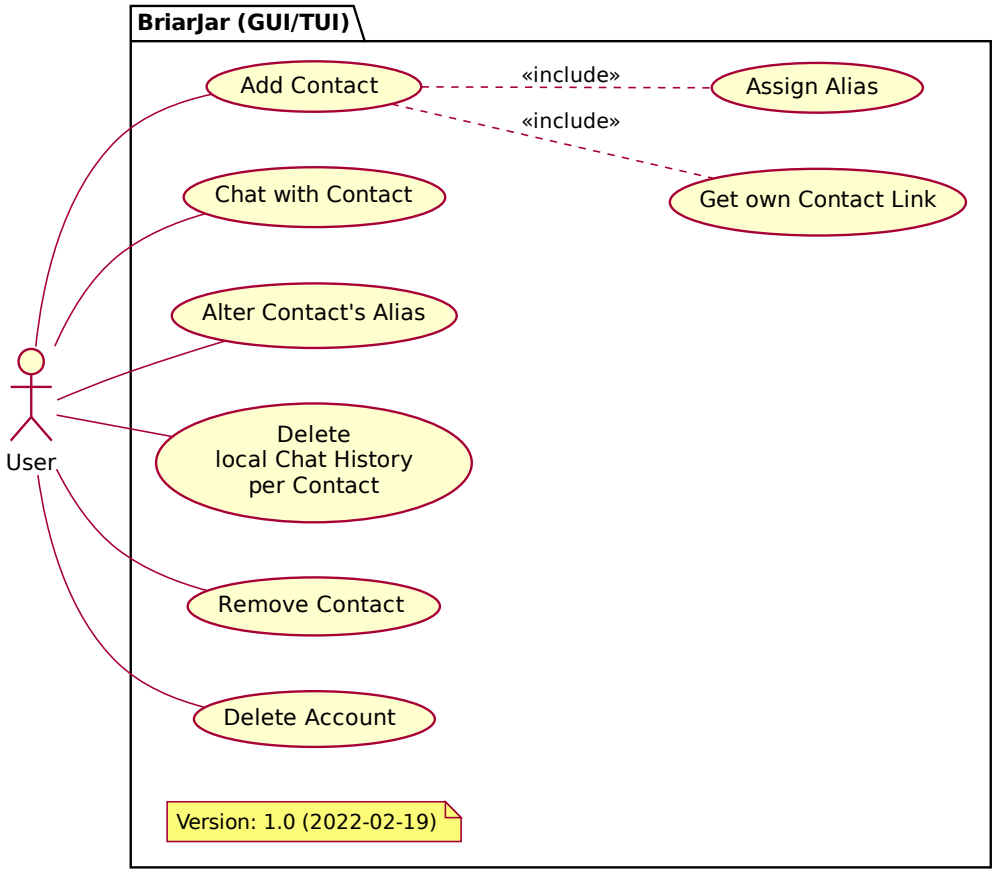
Many functions are implemented in the existing Briar code which will be made accessible with a new API in a way that targets UI developers.

The following functions have to be provided to the user:

- Register a user account
- Login with the registered user account
- Delete the registered user account
- Add a contact
- Delete a contact
- Send a text message to a contact
- Receive a text message by a contact

4.1 Use-Case Diagram

The use-case diagram represents the principle use-cases but differs slightly depending on the UI implementation. Out of diploma project scope functionalities are not included since their implementation is optional.



4.2 Use-Case Descriptions

1, Use-Case Name: Sign-Up

0. Condition: BriarJar has not been started before or account has been deleted.
1. The user starts BriarJar.
2. The user inputs a desired username.
3. The user inputs a strong passphrase.
4. The user selects "Sign-Up".

4.1 If the passphrase is too weak, an exception is thrown.

5. The user is redirected to Main Screen

2, Use-Case Name: Delete Account

0. Condition: The user has an existing account.
1. The user starts BriarJar and optionally signs in.
2. The user selects "Delete Account" and confirm the deletion.
3. BriarJar shows the Sign-Up mask.

3, Use-Case Name: Sign-In

0. Condition: The user has already created an account.
1. The user starts BriarJar.
2. The user inserts their account passphrase.

2.1 If the passphrase is invalid, an error dialogue is shown.

3. The user is being signed in.
4. The user is redirected to the Main Screen.

4, Use-Case Name: Add Contact

0. Condition: The user is signed in and has the handshake link of a peer.
1. The user selects "Add Contact".
2. The user copies and shares their own contact link to the peer.
3. The user pastes the contact link of the peer into the Add Contact mask.
4. The user selects "Start Handshake" to start the handshaking process.
5. If the peer has also done steps 0 - 4 respectively, the handshake process begins.
6. As soon as the process finalises the new contact is shown up in the contact list.

5, Use-Case Name: Get own Handshake Link

0. Condition: The user is signed in.

1. The user selects "Add Contact".

2. The user copies their own Handshake Link.

6, Use-Case Name: Remove Contact

0. Condition: The user is signed in and has added at least one contact.

1. The user selects a Contact.

2. The user selects "Remove Contact".

3. The contact is removed from the contact list.

4. The user will not receive any further messages from the removed contact.

7, Use-Case Name: Choose Contact

0. Condition: The user is signed in and has added at least one contact.

1. The user selects a contact.

2. The user can now either exchange messages with this contact or remove them from the contact list.

8, Use-Case Name: Write Message

0. Condition: The user is signed in and has added at least one contact.

1. The user selects a contact from the contact list.

2. The Conversation mask is now open.

3. The user types a message in the message field.

4. The user selects "Send".

4.1 As long as the selected contact is offline, the message cannot be sent.

5. The sent message is being added to the bottom of the Conversation history.

9, Use-Case Name: Read Message

0. Condition: The user is signed in and has added at least one contact.

1. The user selects a contact from the contact list.

2. The Conversation mask is now open.

3. Possible new received messages are added at the bottom.

4. The user can read the new messages.

5 Non-Functional Requirements

Security Since it is a prototype within a diploma project, it cannot be put too much effort on security.

Usability The use of the software should be self-explanatory for the most part.

Environment The software should run on GNU/Linux operating systems (at least Debian) and should not require any other software except for Java.

Maintainability The software should be an open source prototype and offer the possibility of maintenance.

Scalability The software should create one local account within the "briar" configuration folder. Depending on possible restrictions of Briar-Core, it does not have to be necessarily possible to use multiple messenger instances at the same time from different logged in operating system user accounts. There should be no synchronisation of any kind between different user accounts or devices.

Localisation The software should be available in English.

Architecture The software (jar file) should consist of the GUI and TUI, besides the API which interacts with Briar.

6 Milestones

Planned Date	Milestone Name	Description
2021-10-16	Setup Phase	Create PM-related documents/templates and technical environment
2021-11-04	Design Phase	Make major software architecture related decisions & research Briar's source code with its dependencies
2022-01-20	Minimum Viable Prototype	Implement basic account & contact management
2022-02-12	Development Phase	Complete implementation of basic chat functionality
2022-02-26	Testing Phase	Create simple test cases
2022-03-13	Clean-Up Phase	Clean-up source code and finish project documentation
2022-04-05	Project Submission	~

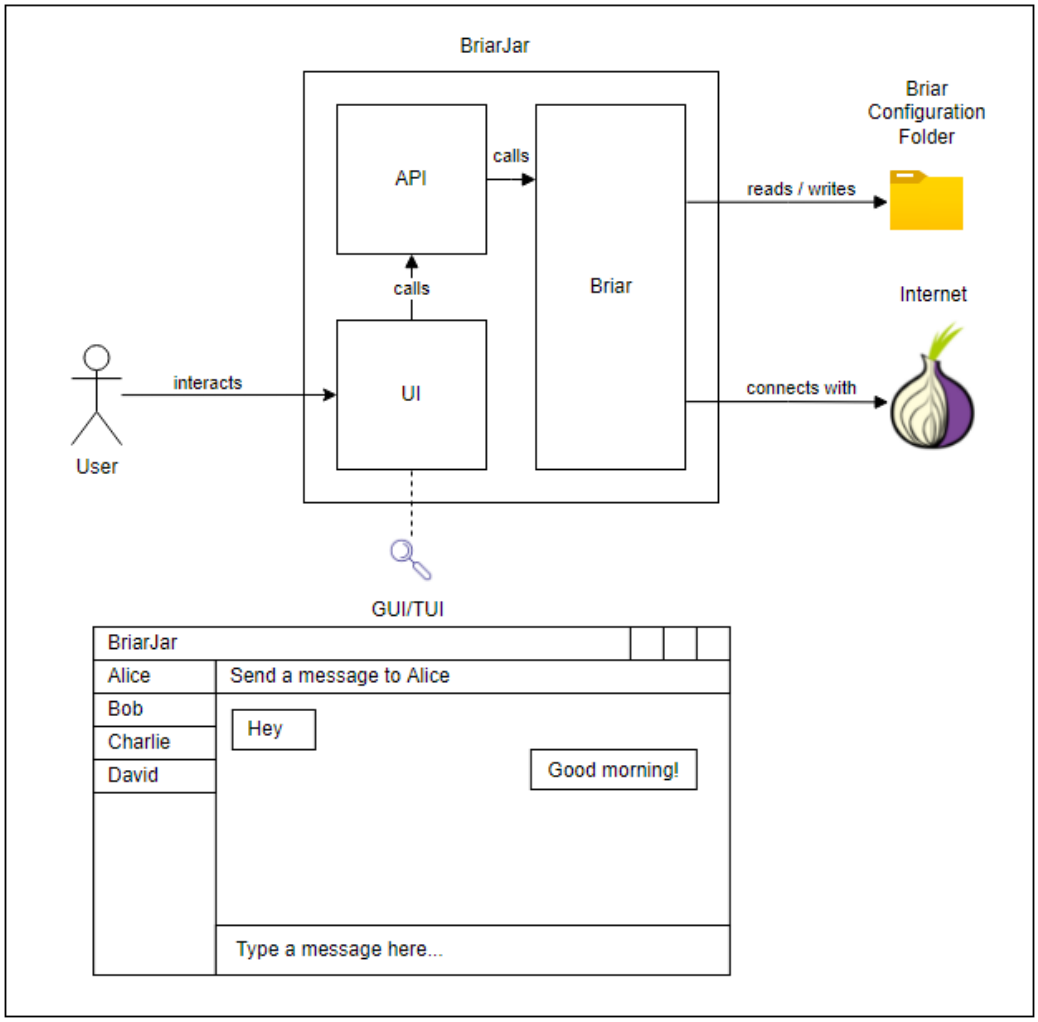
7 Software Data

The user should get an executable jar file (includes the graphical and terminal user interface) with no extra installation procedure needed. But the software should use the configuration folder which is created by the application.

The database handling, key management and similar should stay under Briar's control.

If the configuration folder gets corrupted there is no self-healing process. In such a case, all data will be likely inaccessible if no backup exists. Although, security specialists could be able to restore data depending on many different factors.

8 System Architecture



9 User Interfaces

The software will be presented in two modes:

- Graphical User Interface (GUI)
- Terminal User Interface (TUI)

Both editions require a computer with a GNU/Linux operating system and Java installed as well as a working Internet connection, a keyboard and a monitor, but the GUI extends the requirements to a computer mouse and a running display server.

10 Scope of Delivery

The client will receive a working prototype of BriarJar. This prototype consists of one executable Java Archive (jar file) which allows for the invocation of the user interface (terminal and graphical) by passing a command-line argument (for example, --tui starts the TUI).

11 Criteria of Approval

Objective (for both GUI and TUI)	**Project completion in %**
Project dependencies are satisfied on Debian GNU/Linux with Java 17 installed	10
BriarJar allows for account management (sign up, sign in, delete account)	30
Both User Interfaces (TUI, GUI) are fully prototyped and can be invoked	50
BriarJar allows for contact management (generate handshake link, add/remove contact)	70
BriarJar allows displaying, sending and receiving messages	90
Tests are implemented / done for functional requirements	100

12 Glossary

API	Application Programming Interface
Briar	Messenger developed by Briar Project
Free and open-source software (FOSS)	Software licensed with freedom in mind and source code available to public
GUI	Graphical User Interface
Interface	Layer between different components
jar	Run-able Java archive application
TUI	Terminal User Interface

SPENGERGASSE
ausbildung mit zukunft

BRIAR

Project

BriarJar

Software Requirements Specification

(Pflichtenheft)

Version	1.00
Date	2021-10-10

Confidentiality	**[x] Internal [ ] External**
File Name	BriarJar_SRS_20211010_007

Project Partner	Briar Project
Project Supervisor	Andreas Chwatal

Project Team	Farman Khan	UI/UX Architect & Engineer
	Alexander Zeidler	Project Leader, Software Architect & Engineer

Document Revision History

Version	Date	Name(s)	Change Description
0.01	2021-10-10	FK, AZ	Initialise document
0.02	2021-11-24	FK, AZ	Add/Update Preface, GUI/TUI Screen Layouts, Dependency Injection, Gradle, User Interface Implementation, Development Setup
0.03	2022-01-23	FK, AZ	Add Software Design Pattern, Class Diagram (first Revision), Data Requirements; Update Dependency Injection
0.04	2022-01-28	AZ	Add Activity Diagram
0.05	2022-02-19	AZ	Mainly update Activity Diagram
0.06	2022-03-17	FK, AZ	Update this document
1.00	2022-03-30	AZ	Release version 1.00
Farman Khan (FK), Alexander Zeidler (AZ)			

Table of Contents

1 Preface 4

2 Overview 5
2.1 System Description 5
2.1.1 Activity Diagram 5
2.2 GUI Screen Layouts 7
2.2.1 Sign-up Screen 7
2.2.2 Sign-in Screen 7
2.2.3 Main Chat Screen 8
2.3 TUI Screen Layouts 9
2.3.1 Sign-up Screen 9
2.3.2 Sign-in Screen 9
2.3.3 Contact Selection Screen 9

3 Technical Description 10
3.1 Software Design Pattern (MVC) 10
3.2 Dependency Injection (Dagger2) 10
3.2.1 Good Practices with Dagger2 10
3.2.2 Basic Principles 10
3.2.3 Implementation 11
3.3 Gradle 12
3.3.1 Gradle Tasks 12
3.3.2 Gradle Dependencies and Plug-ins 12
3.4 User Interface Implementation 13
3.4.1 User Interface Invocation 13
3.4.2 Passing Java Object References to the UI 13
3.4.3 TUI-specific Limitations 13
3.5 Structure of Classes 13
3.6 Data Requirements 14
3.6.1 Operating System 14
3.6.2 BriarJar General 14
3.6.3 BriarJar Interaction Limits 14

4 Development Setup 15
4.1 Code Style and Formatting 15
4.2 Git Setup 15
4.2.1 Repository 15
4.2.2 Cloning 15
4.2.3 Commit Rules 15
4.2.4 Briar as a Git Submodule 16
4.3 Environment Setup 16
4.3.1 Research (Briar Headless) 16
4.3.2 Development (BriarJar) 16
4.4 Software Versions 17

1 Preface

According to DIN 69901-5, the Software Requirements Specification comprises the

> "realisation specifications drawn up by the contractor based on the implementation of the Product Requirements Document specified by the client".

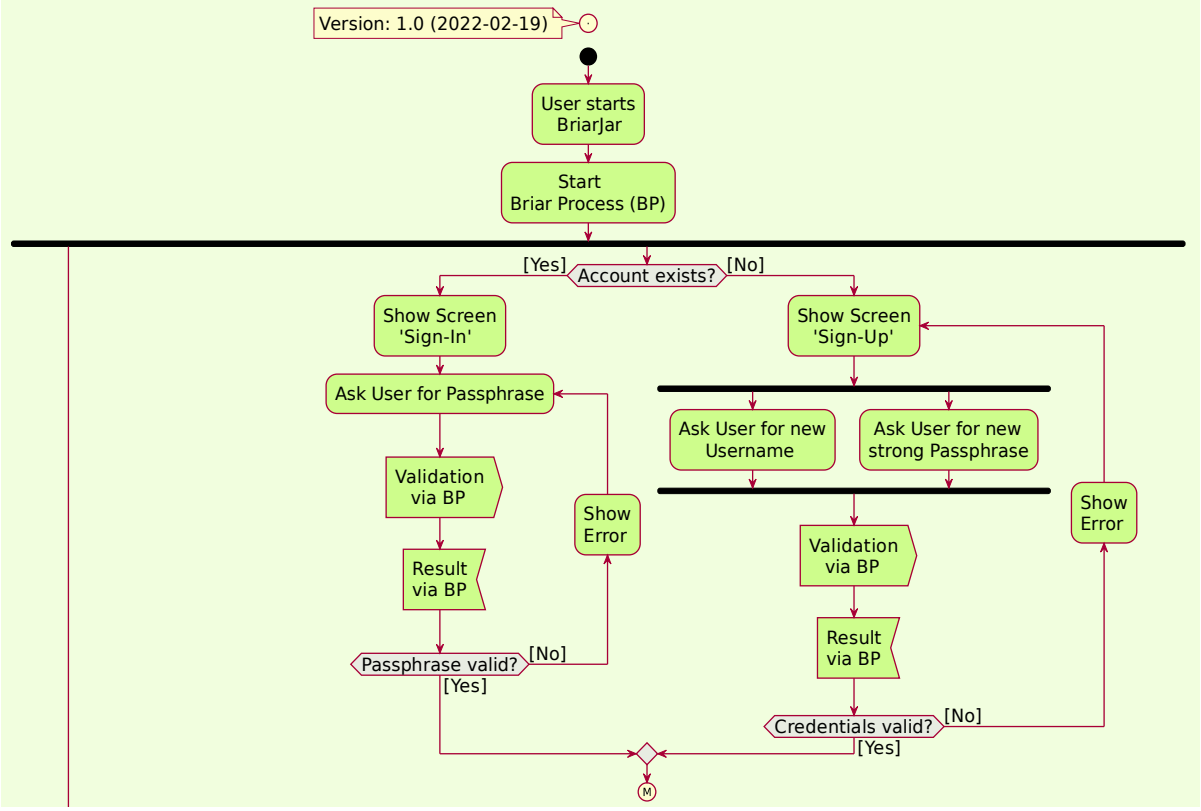
The requirements of the previously elaborated specifications are now linked to technical specifications of the operating and maintenance environment.

2 Overview

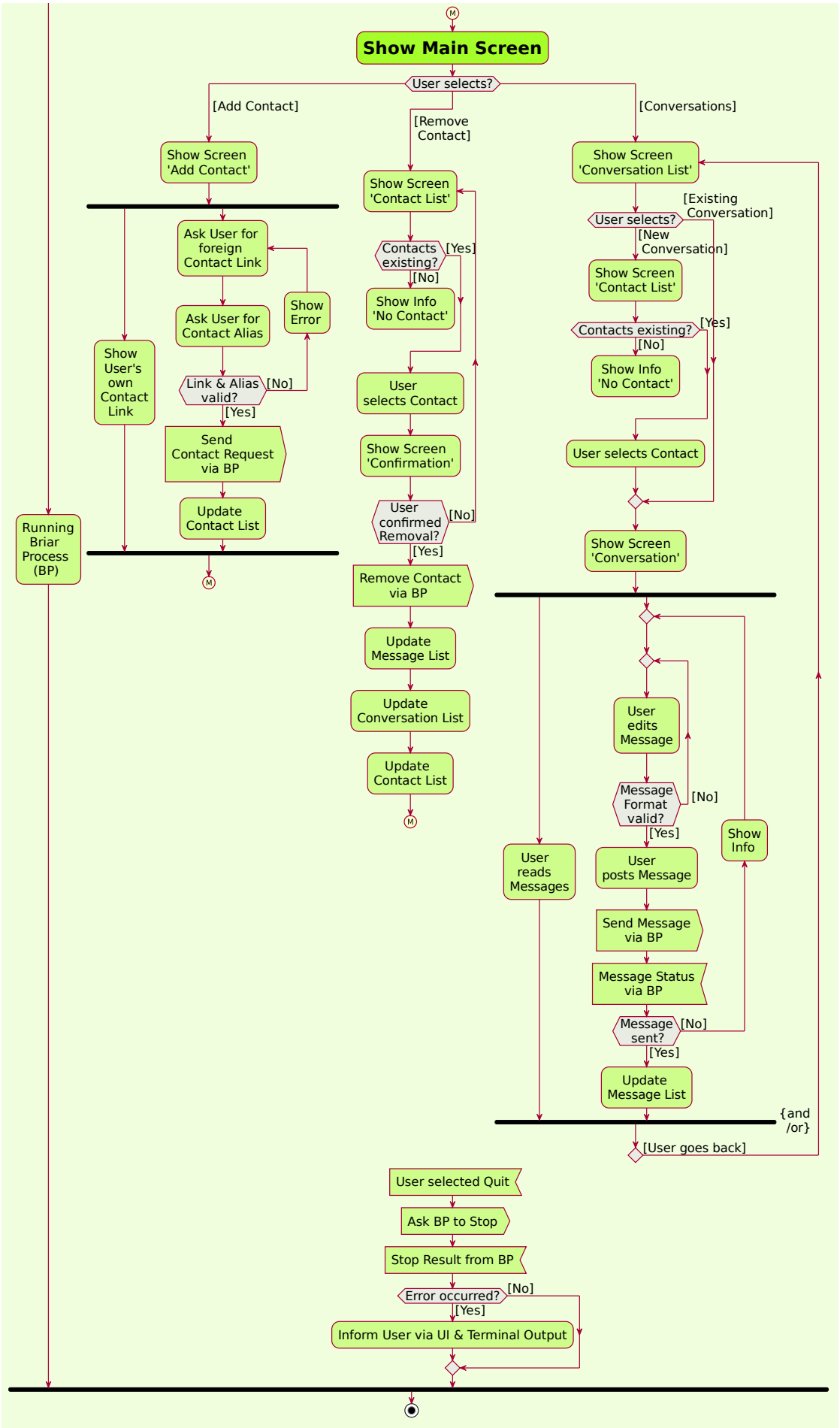
2.1 System Description

2.1.1 Activity Diagram

The activity diagram represents the principle flow but differs slightly depending on the UI implementation. Out of diploma project scope functionalities are not included since their implementations are optional.

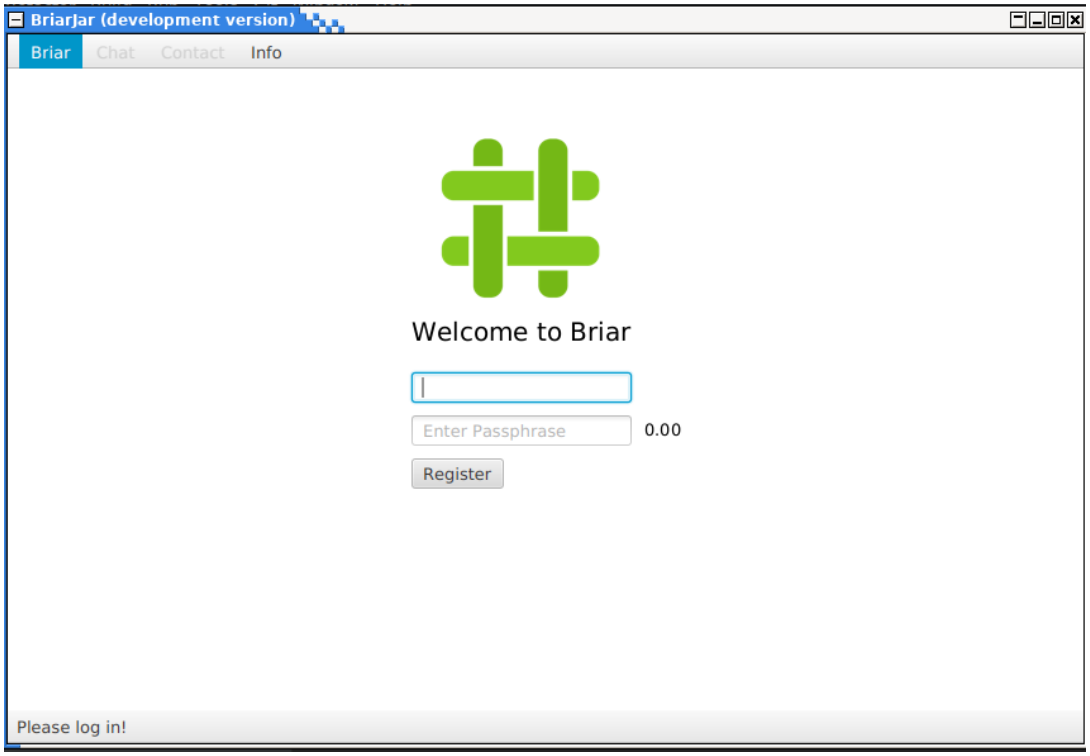


Continuous on next page...

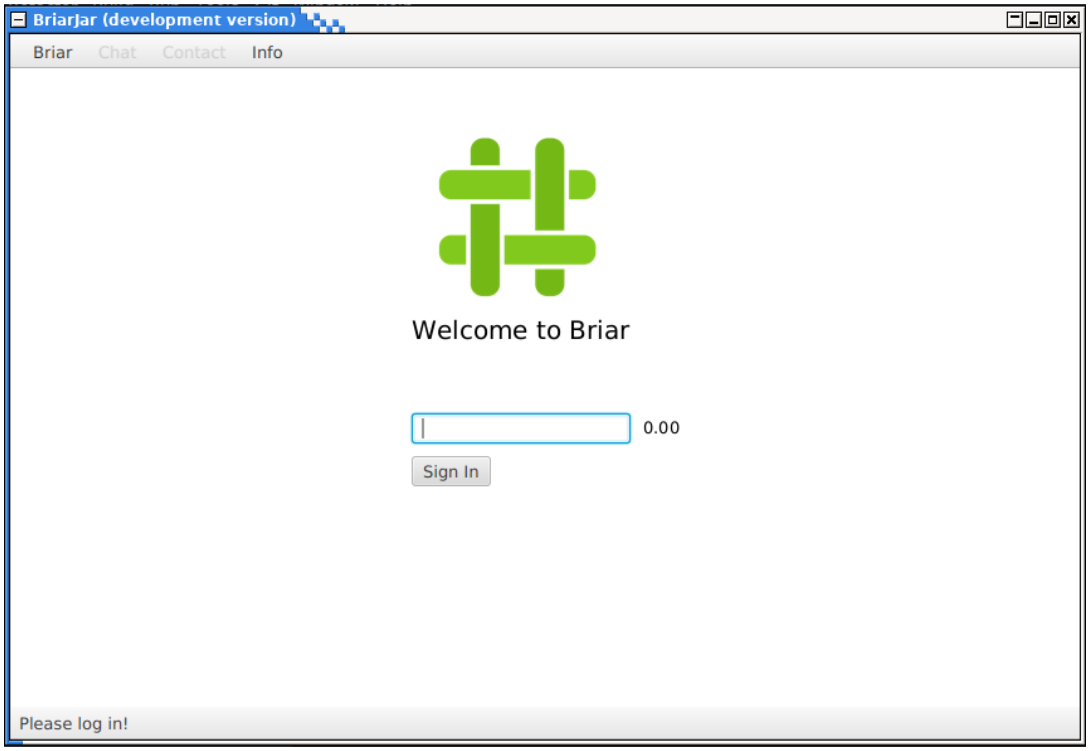

Show Main Screen
User selects?
[Add Contact]
[Remove Contact]
[Conversations]
Show Screen 'Add Contact'
Ask User for foreign Contact Link
Ask User for Contact Alias
Link & Alias valid?
[No]
[Yes]
Show Error
Send Contact Request via BP
Update Contact List
Show User's own Contact Link
Running Briar Process (BP)
Show Screen 'Contact List'
Contacts existing?
[Yes]
[No]
Show Info 'No Contact'
User selects Contact
Show Screen 'Confirmation'
User confirmed Removal?
[No]
[Yes]
Remove Contact via BP
Update Message List
Update Conversation List
Update Contact List
Show Screen 'Conversation List'
[Existing Conversation]
User selects?
[New Conversation]
Show Screen 'Contact List'
Contacts existing?
[Yes]
[No]
Show Info 'No Contact'
User selects Contact
Show Screen 'Conversation'
User edits Message
Message Format valid?
[No]
[Yes]
Show Info
User reads Messages
User posts Message
Send Message via BP
Message Status via BP
Message sent?
[No]
[Yes]
Update Message List
{and /or}
[User goes back]
User selected Quit
Ask BP to Stop
Stop Result from BP
Error occurred?
[No]
[Yes]
Inform User via UI & Terminal Output

2.2 GUI Screen Layouts

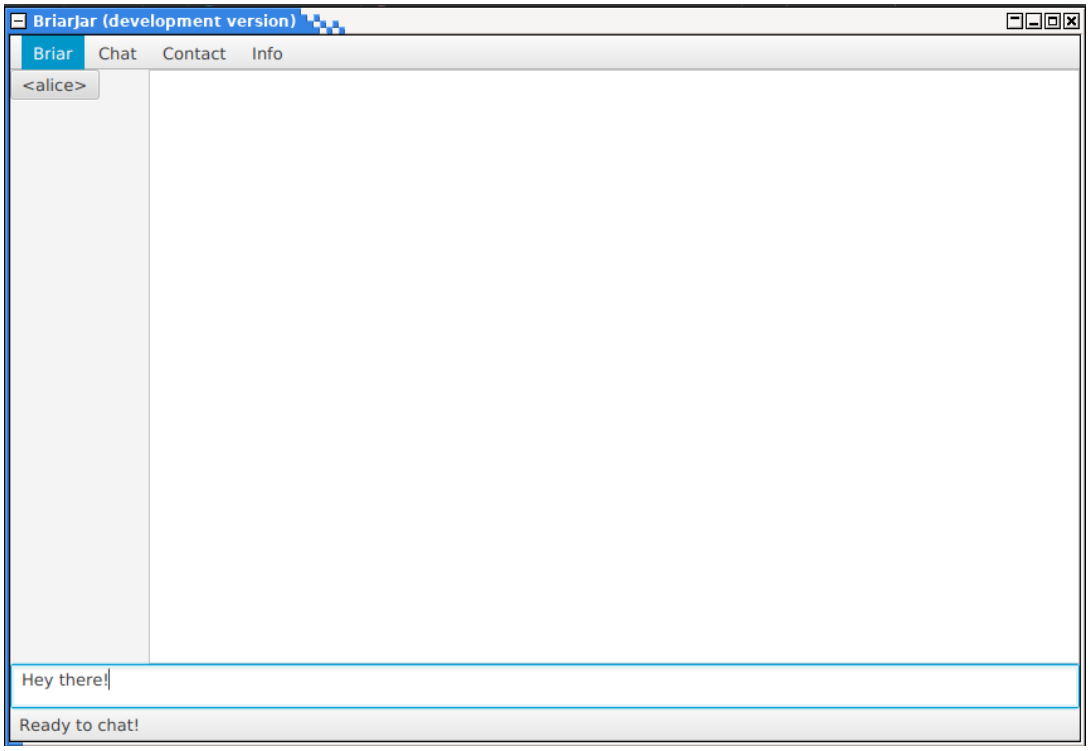
2.2.1 Sign-up Screen



2.2.2 Sign-in Screen

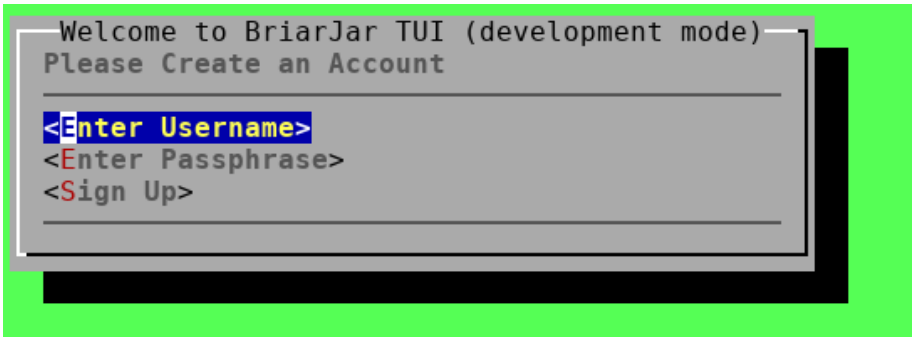


2.2.3 Main Chat Screen

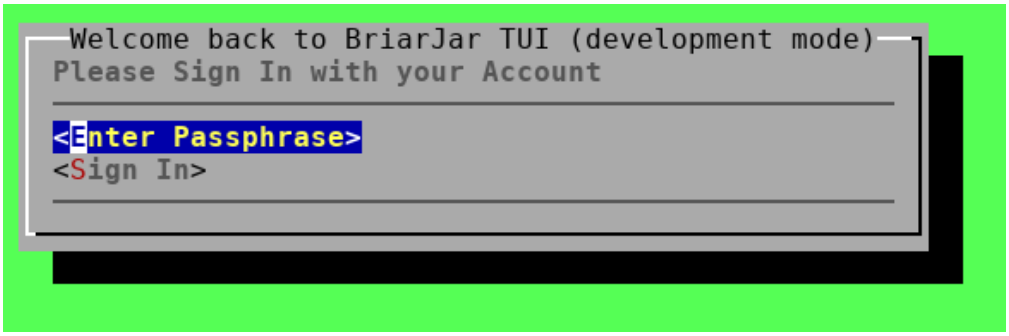


2.3 TUI Screen Layouts

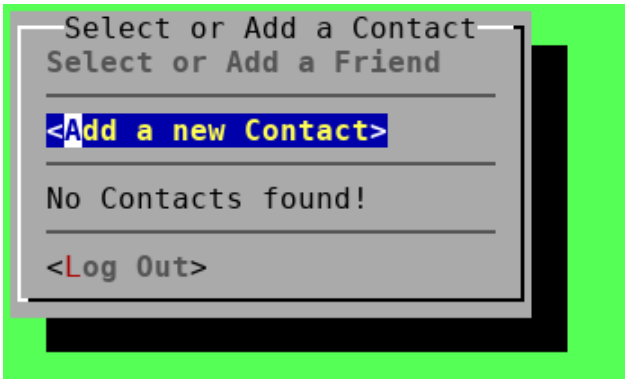
2.3.1 Sign-up Screen



2.3.2 Sign-in Screen



2.3.3 Contact Selection Screen



3 Technical Description

3.1 Software Design Pattern (MVC)

Component	Implementation	Info
M odel	Briar	Provided by the client (Briar Project)
V iew	BriarJar GUI/TUI	User input / output
C ontroller	BriarJar API	Internally named "ViewModels"; Interface for View

3.2 Dependency Injection (Dagger2)

"Giving an object its instance variables." - James Shore

Dependency Injection (DI) is a pattern on how to satisfy classes' dependencies on other objects by annotating them in different ways. DI can reduce the written code by programmers a lot but brings also new complexity in the project, especially for beginners.

Briar decided in 2016 to switch to another DI named Dagger2 which is the successor of Dagger and maintained by Google. Therefore, BriarJar will also make use of Dagger2.

3.2.1 Good Practices with Dagger2

- use constructor injection with @Inject annotation whenever possible
- it is easy to program against interfaces when you can not or do not want to create an instance
- when constructor injection is not possible (e.g. due to a read-only 3rd party library) or using Mocks or similar is desired, use @Module s instead with:
  - @Binds when no configuration of the dependency is needed (less expensive)
  - @Provides when configuration of the dependency is needed (more expensive)

3.2.2 Basic Principles

Classes which should be injected need an @Inject annotated (no-args) constructor.

Dagger2 can inject into constructors, fields and less wanted into methods, too.

@Inject annotated fields will be injected but without an @Inject annotated (no-args) constructor not instantiated.

3.2.3 Implementation

Following an example on how BriarJar mainly uses Dagger2 in its classes. Dagger2 is more than just the example below like, for instance, Briar(Jar) uses also components and modules, but that is described in detail on Dagger2's website (https://dagger.dev/dev-guide/) or can be viewed in Briar(Jar)'s source code.

Constructor Injection - Desired Class

```
@Singleton
public class ContactViewModel {

      private final ContactManager contactManager;

      @Inject
      public ContactViewModel( ContactManager contactManager )
      {
             this.contactManager = contactManager;
      }
(...)
```

Constructor Injection - Receiver

```
public class ContactList extends EventListenerViewModel {

      private final ContactViewModel cvm;

      @Inject
      public ContactList( EventBus           eventBus,
                          ContactViewModel   cvm )
      {
             (...)
             this.cvm = cvm;
      }
(...)
```

3.3 Gradle

There are three main objectives for the Gradle setup in this project:

- Running the client (within the IDE).
- Creating a runnable jar file.
- Optional: Creating Tests.

3.3.1 Gradle Tasks

- The Application Task: Builds the dependencies and invokes the main method of the application.
- The ShadowJar Task: Creates a so-called “FatJar” to allow users to execute the application without installing dependencies (stand-alone).

3.3.2 Gradle Dependencies and Plug-ins

The following Gradle plug-ins will be used:

- Java
- Application
- Idea
- JavaFX (version 0.0.9)
- Shadow (version 7.1.1)

The following Gradle dependencies will be used:

- Briar-as-subproject
- Dagger (version 2.40.5)
- Lanterna (version 3.1.1)
- JFoenix (version 9.0.10)

3.4 User Interface Implementation

3.4.1 User Interface Invocation

The default user interface is the GUI. The terminal user interface can simply be invoked by passing the argument --tui .

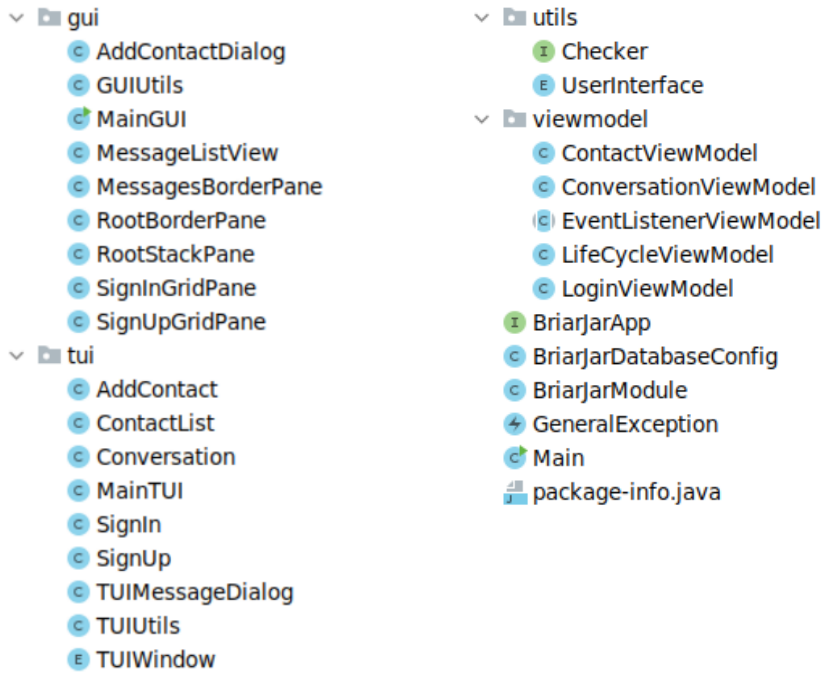
3.4.2 Passing Java Object References to the UI

The instances of the view models (BriarJar API) will be passed to the UI classes by using Dagger2's constructor injection. The view models provide access to the features and functions of Briar.

3.4.3 TUI-specific Limitations

The terminal user interface adheres the limitations of the implementation of the "DefaultTerminalFactory". For example, the background colour can not be in the same shade of green as the colours provided by Briar Project. Therefore, the colour should be as close as possible, but within the limits of the Lanterna API (in this specific case, we will use ANSI.GREEN_BRIGHT).

3.5 Structure of Classes



3.6 Data Requirements

3.6.1 Operating System

- Debian GNU/Linux or a similar OS
- x86_64 Bit Architecture
- CPU: Pentium 4, 1 GHz (minimum)
- RAM: 1 Gigabyte (minimum)
- Free Disk Space: 100 MB (minimum)

3.6.2 BriarJar General

- OpenJDK 17

3.6.3 BriarJar Interaction Limits

	Minimum	Maximum	Character-Encoding
Message-Text	1	30720 - 1 (Briar 30.720)	UTF-8
Username	1	50	UTF-8
Alias	1	50	UTF-8
Passphrase	15	200	UTF-8

4 Development Setup

4.1 Code Style and Formatting

Derived from Briar Project but not strictly enforced at this time (see chapter "General Design Concept" in diploma thesis).

- Lines should be no longer than 80 characters.
- Make tabs for indentation with a tab width of 4.
- Egyptian brackets (else/catch/finally) continue the same line as the preceding bracket.
- if/else/for/while may omit brackets if the body fits on a single line.
- If the if uses brackets, the else should too (and vice versa).
- Use foo rather than mFoo, _foo, etc for field names.
- Use foo rather than this.foo unless there is also a local foo.
- Create CONSTANTS_LIKE_THIS and variablesLikeThis.
- Use AbcCamelCase rather than ABCCamelCase.
- Avoid staircase indentation.
- Use static imports.

4.2 Git Setup

4.2.1 Repository

We have decided to request and create a git repository using the official Briar Project's GitLab instance which is available at:

https://code.briarproject.org/briar/briarjar (only accessible by PTM currently)

4.2.2 Cloning

Cloning the repository requires the –recurse-submodules parameter:

```
git clone --recurse-submodules git@code.briarproject.org/briar/briarjar
```

4.2.3 Commit Rules

1. Commit frequently and avoid conjugations (instead of “Add x and y” split into “Add x” and “Add y”).
2. Keep the messages concise and precise.
3. Begin all messages in the imperative form (“Add …”, “Fix …”, “Delete …”).

4.2.4 Briar as a Git Submodule

In order to make use of Briar's internals and source code, we have to obtain it and amalgamate it with our code base. This is either doable by using a Gradle dependency or a git submodule. We have used a git submodule, since our project partner recommended us that way.

We have added the git submodule with the following command:

```
git submodule add https://code.briarproject.org/briar/briar
```

4.3 Environment Setup

4.3.1 Research (Briar Headless)

IntelliJ provides several features for documentation look-up. Therefore, Briar's source code has to be obtained and successfully compiled.

1. The Briar source code has to be cloned.

   ```
   git clone https://code.briarproject.org/briar/briar
   ```

2. The cloned directory has to be opened with IntelliJ and then "trusted".

3. Following lines from the "settings.gradle" have to be deleted:

   ```
   include 'bramble-android'
   include 'briar-android'
   ```

4. The following line has to be appended to the gradle.properties file:

   ```
   org.gradle.configureondemand=true
   ```

5. The "build.gradle" file from the desired Briar project to compile (eg. "briar-headless") has to be opened.

6. Inside the "build.gradle" file, there should be lines beginning with "task…" which have a green play button on the left side.

7. The desired build can be compiled by clicking on the desired compilation task.

8. After compilation, the desired jar files should be inside a build/libs directory.

4.3.2 Development (BriarJar)

Before cloning, SSH access to the repository is required.

1. The BriarJar source code has to be cloned.

   ```
   git clone git@code.briarproject.org:briar/briarjar
   ```

2. The cloned directory has to be opened with IntelliJ and then "trusted".

3. Optional: IntelliJ might require to change the project SDK to a lower version (for dependency compatibility reasons). This can be done via the menu “File” > “Project Settings”

4. Now, the Gradle Tasks for cleaning and building should be executed.

5. In the Main.java class, a green play icon should be visible. By clicking it, the Main class is executed.

4.4 Software Versions

Software / Dependency	Version
Java	17 (JDK)
IntelliJ	IDEA 2021.2.1
JavaFX (Gradle)	0.0.9
Lanterna (Gradle)	3.1.1
Dagger (Gradle)	2.40.5
Shadow (Gradle)	7.1.1

SPENGERGASSE
ausbildung mit zukunft

BRIAR

Project

BriarJar

Test Cases (TC)

Version	1.00
Date	2022-02-19

Confidentiality	**[ ] Internal [x] External**
File Name	BriarJar_TC_20220119_003

Project Partner	Briar Project
Project Supervisor	Andreas Chwatal

Project Team	Farman Khan	UI/UX Architect & Engineer
	Alexander Zeidler	Project Leader, Software Architect & Engineer

Document Revision History

Version	Date	Name(s)	Change Description
0.01	2022-02-19	FK, AZ	Initialise document and run tests
0.02	2022-03-13	FK, AZ	Update test case: p-c-1
1.00	2022-03-30	AZ	Release version 1.00
Farman Khan (FK), Alexander Zeidler (AZ)			

Table of Contents

1 Preface 4

1.1 Scope 4

1.2 Testing Environments 4

1.2.1 Tester FK 4

1.2.2 Tester AZ 4

1.3 Glossary of Test Tables 4

2 Tests – Proper Use 5

3 Tests – Improper Use 7

1 Preface

1.1 Scope

This document contains documented test cases about the GUI implementation if they are within the project scope. The TUI implementation and additional functionalities like change contact alias and similar are not subject of this documentation due to time resource limitations.

1.2 Testing Environments

1.2.1 Tester FK

Operating System (OS)	Fedora GNU/Linux 34
Kernel	5.16.9-100.fc34.x86_64
xorg-server	1.20.11
Java	OpenJDK Runtime Environment 21.9 (build 17.0.2+8)
	OpenJDK 64-Bit Server VM 21.9 (build 17.0.2+8, mixed mode, sharing)
JavaFX Runtime Version	17+18

1.2.2 Tester AZ

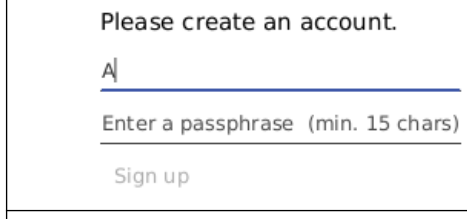
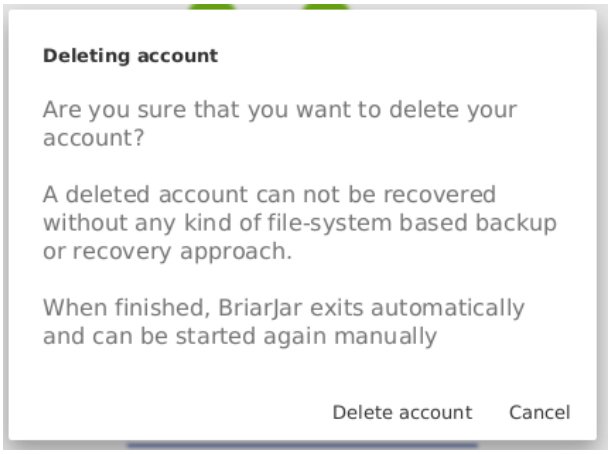
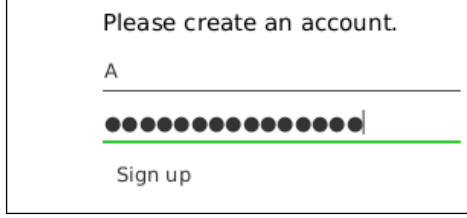
Operating System (OS)	Debain GNU/Linux 11 (bullseye)
Kernel	5.15.15-2~bpo11+1 (2022-02-03)
xorg-server	2:1.20.11-1+deb11u1
Java	OpenJDK Runtime Environment (build 17.0.2+8-Debian-1deb11u1)
	OpenJDK 64-Bit Server VM (build 17.0.2+8-Debian-1deb11u1, mixed mode, sharing)
JavaFX Runtime Version	17

1.3 Glossary of Test Tables

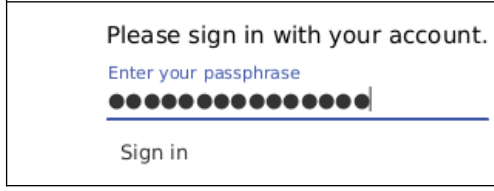
ID	An ongoing enumeration. First digit p (proper use) or i (improper use), second digit a to z (table of test) third and last digit 0 ... n (objective of test).
Scope	The tested scope (sign-up, sign-in, contacts, messages).
Tester	Initials of the tester(s). Full names are quoted in the document revision history.
Objective	The objective (goal) of the test case.
Input	Used input parameters (where available / meaningful).
Version	The tested software version (date of last project Git commit).
Success Date	The date of the successful test (if there exists an issue, the date is put in parentheses).
Comments	Comments about anything special to be considered.
Documentation	Screenshots or other resources to substantiate the successful results.

2 Tests – Proper Use

ID	Scope	Sign-up / Delete	Tester	AZ
p-a-	Objective	Input	Version	Success Date
1	valid user name	1 character, A	2022-02-19	2022-02-19
2	valid passphrase	15 characters, A ... O	2022-02-19	2022-02-19
3	delete account (both signed-out and signed-in)		2022-02-19	2022-02-19
Comments				
-				

Documentation	
1	3
	
2	
	

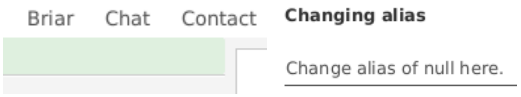
ID	Scope	Sign-in	Tester	AZ
p-b-	Objective	Input	Version	Success Date
1	valid passphrase	15 characters, A ... O	2022-02-19	2022-02-19
Comments				
-				

Documentation	
1	
	

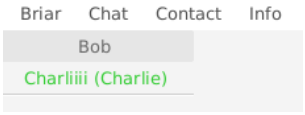
ID	Scope	Contacts	Tester	AZ
p-c-	Objective	Input	Version	Success Date
1	add contact mutually	Alice, Bob	2022-03-13	2022-03-13
2	remove contact	Alice	2022-02-19	2022-02-19

Comments

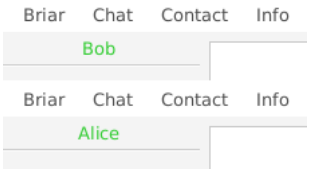
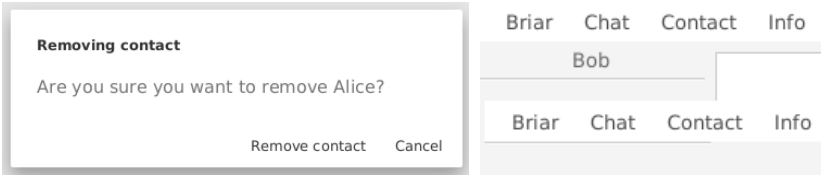
p-c-1: ~~There is a bug of not saving the specified alias sometimes. It seems to occur when both are not online at the same time when adding each other. The result is an alias named "null" which is blank in the GUI.~~
~~Workaround: click `Contact > Change Contact Alias` and rename it.~~



Solved. Briar does not store an alias which is equal to the user name, but BriarJar just displayed the - sometimes not stored - alias. The recent version of BriarJar displays both alias and user name or otherwise just the user name:



Documentation

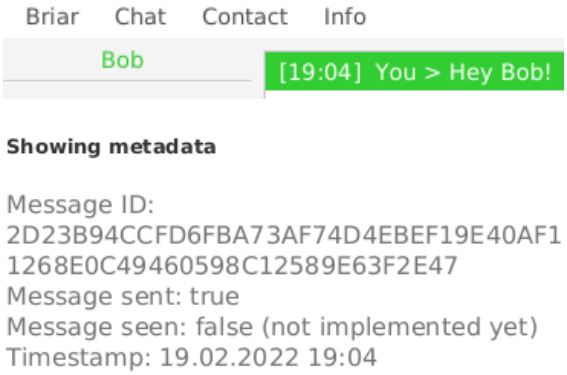
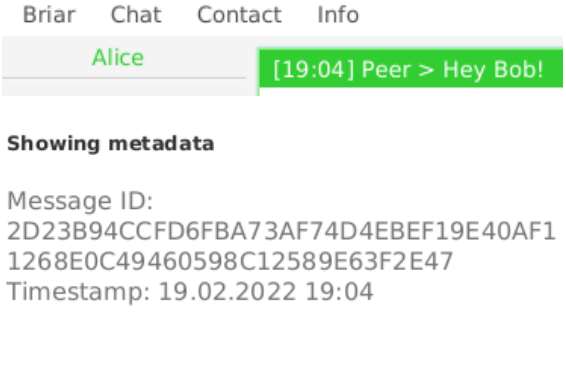
1	2
	

ID	Scope	Messages	Tester	AZ
p-d-	Objective	Input	Version	Success Date
1	send message	Hey Bob!	2022-02-19	2022-02-19
2	receive sent message		2022-02-19	2022-02-19

Comments

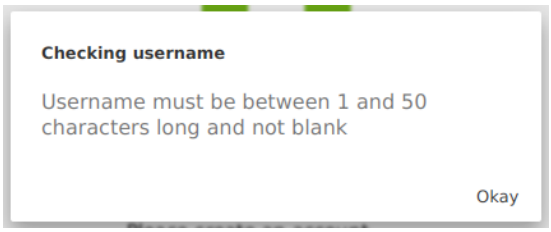
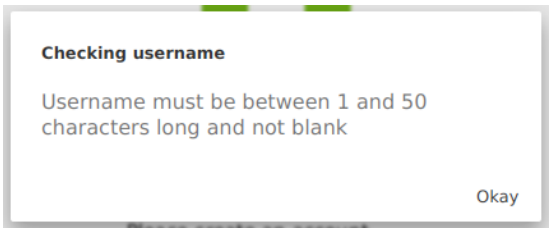
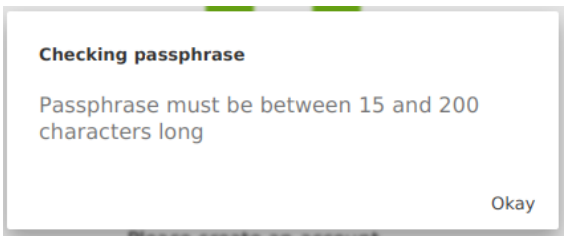
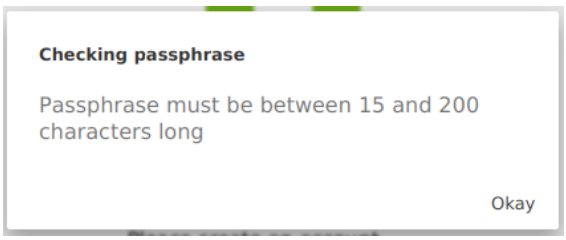
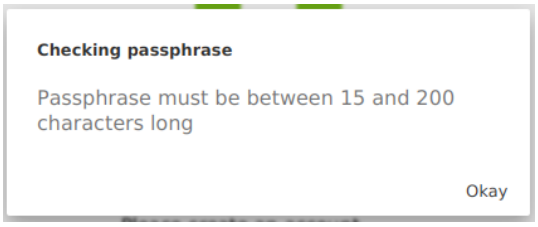
-

Documentation

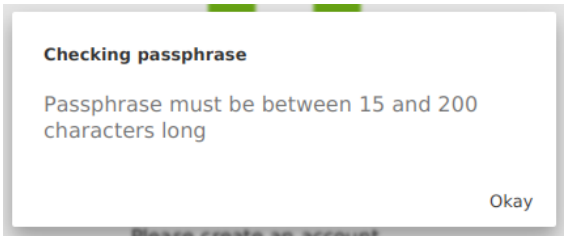
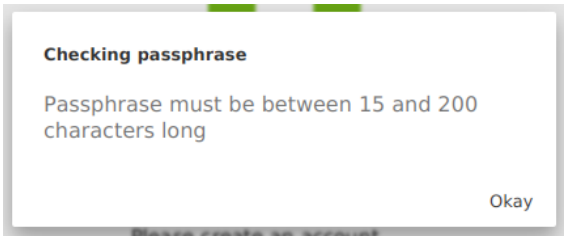
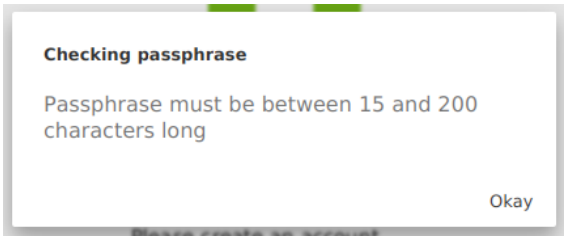
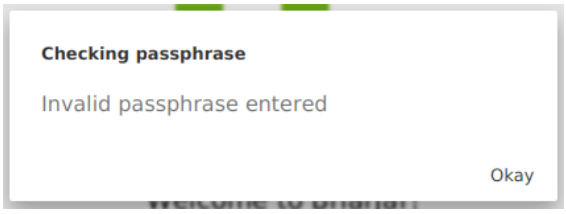
1	2
	

3 Tests – Improper Use

ID	Scope	Sign-up (error messages)	Tester	FK
i-a-	Objective	Input	Version	Success Date
1	no (equals too short) user name	0 characters	2022-02-19	2022-02-19
2	too long user name	50 + 1 characters	2022-02-19	2022-02-19
3	no passphrase	0 characters	2022-02-19	2022-02-19
4	too short passphrase	12 - 1 characters	2022-02-19	2022-02-19
5	too long passphrase	200 + 1 characters	2022-02-19	2022-02-19
Comments				
-				

Documentation	
1	2
	
3	4
	
5	
	

ID	Scope	Sign-in (error messages)	Tester	FK
i-b-	Objective	Input	Version	Success Date
1	no passphrase	0 characters	2022-02-19	2022-02-19
2	too short passphrase	12 - 1 characters	2022-02-19	2022-02-19
3	too long passphrase	200 +1 characters	2022-02-19	2022-02-19
4	invalid passphrase	12 characters, AA…	2022-02-19	2022-02-19
Comments				
-				

Documentation	
1	2
	
3	4
	

ID	Scope	Contacts (error messages)	Tester	FK, AZ
i-c-	Objective	Input	Version	Success Date
1	empty link	0 characters	2022-02-19	2022-02-19
2	own link		2022-02-19	2022-02-19
3	too long link (link + 1 character)		2022-02-19	(2022-02-19)
4	already added link		2022-02-19	2022-02-19

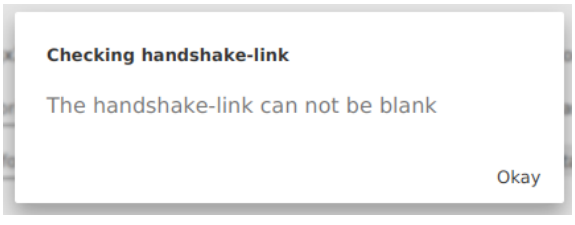
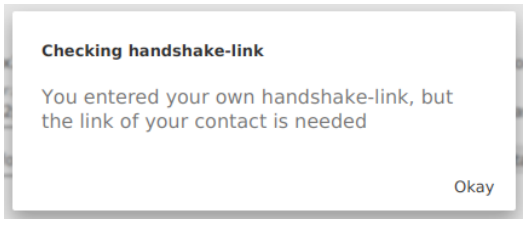
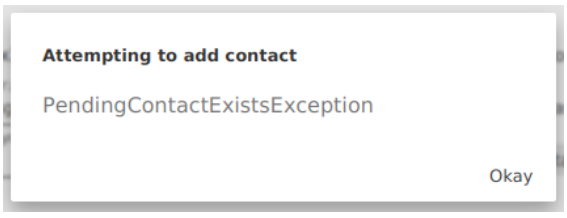
Comments

i-c-3: It is fine that there is no error message, since Briar filters just the needed information and discards the rest.

```java
package org.briarproject.bramble.api.contact;

import java.util.regex.Pattern;

public interface HandshakeLinkConstants {

    The current version of the handshake link format.
    int FORMAT_VERSION = 0;

    The length of a base32-encoded handshake link in bytes, excluding the
    'briar://' prefix.
    int BASE32_LINK_BYTES = 53;

    The length of a raw handshake link in bytes, before base32 encoding.
    int RAW_LINK_BYTES = 33;

    Regular expression for matching handshake links, including or excluding the
    'briar://' prefix.
    Pattern LINK_REGEX =
            Pattern.compile("(briar://)?([a-z2-7]{" + BASE32_LINK_BYTES + "})");

    Label for hashing handshake public keys to calculate their identifiers.
    String ID_LABEL = "org.briarproject.bramble/HANDSHAKE_KEY_ID";
}
```

```java
Matcher matcher = LINK_REGEX.matcher(link);
if (!matcher.find()) throw new FormatException();
// Discard 'briar://' and anything before or after the link
link = matcher.group(2);
```

Documentation

1	2
	
3	4
no error message needed, see comment	

ID	Scope	Messages (error messages)		Tester	FK, AZ
i-d-	Objective		Input	Version	Success Date
1	empty message		0 characters	2022-02-19	2022-02-19
2	message text too long		30.719 + 1 characters (UI implementation limit)	2022-02-19	2022-02-19
Comments					
-					

Documentation	
1	2
	

SPENGERGASSE
ausbildung mit zukunft

BRIAR

Project

BriarJar

Status Reports
(SR)

Version	1.00
Date	2021-10-17

Confidentiality	**[ ] Internal [x] External**
File Name	BriarJar_SR_20211017_013

Project Partner	Briar Project
Project Supervisor	Andreas Chwatal

Project Team	Farman Khan	UI/UX Architect & Engineer
	Alexander Zeidler	Project Leader, Software Architect & Engineer

Document Revision History

Version	Date	Name(s)	Change Description
0.01	2021-10-17	FK, AZ	Initialise document and add regular status report
0.02	2021-10-31	FK, AZ	Update regular status report
0.03	2021-11-14	FK, AZ	Update regular status report
0.04	2021-11-28	FK, AZ	Update regular status report
0.05	2021-12-12	FK, AZ	Update regular status report
0.06	2021-12-26	FK, AZ	Update regular status report
0.07	2022-01-09	FK, AZ	Update regular status report
0.08	2022-01-23	FK, AZ	Update regular status report
0.09	2022-02-06	FK, AZ	Update regular status report
0.10	2022-02-20	FK, AZ	Update regular status report
0.11	2022-03-06	AZ	Update regular status report
0.12	2022-03-20	FK, AZ	Update regular status report
1.00	2022-03-30	AZ	Update regular status report, Release version 1.00
Farman Khan (FK), Alexander Zeidler (AZ)			

Table of Contents

1 Introduction 4

2 Lessons Learned 4

3 Milestone Trend Analysis 5

4 Status Report 6

5 Recent & Upcoming Work 7

6 History of Status Reports 8

6.1 2021-12-12 8

6.2 2021-12-26 9

6.3 2022-01-09 10

6.4 2022-01-23 11

6.5 2022-02-06 12

6.6 2022-02-20 13

6.7 2022-03-06 14

6.8 2022-03-20 15

1 Introduction

This document provides status report information for our project development course.

It intends to inform our professors about the project progress since the last report issued using traffic light colours and a milestone trend analysis.

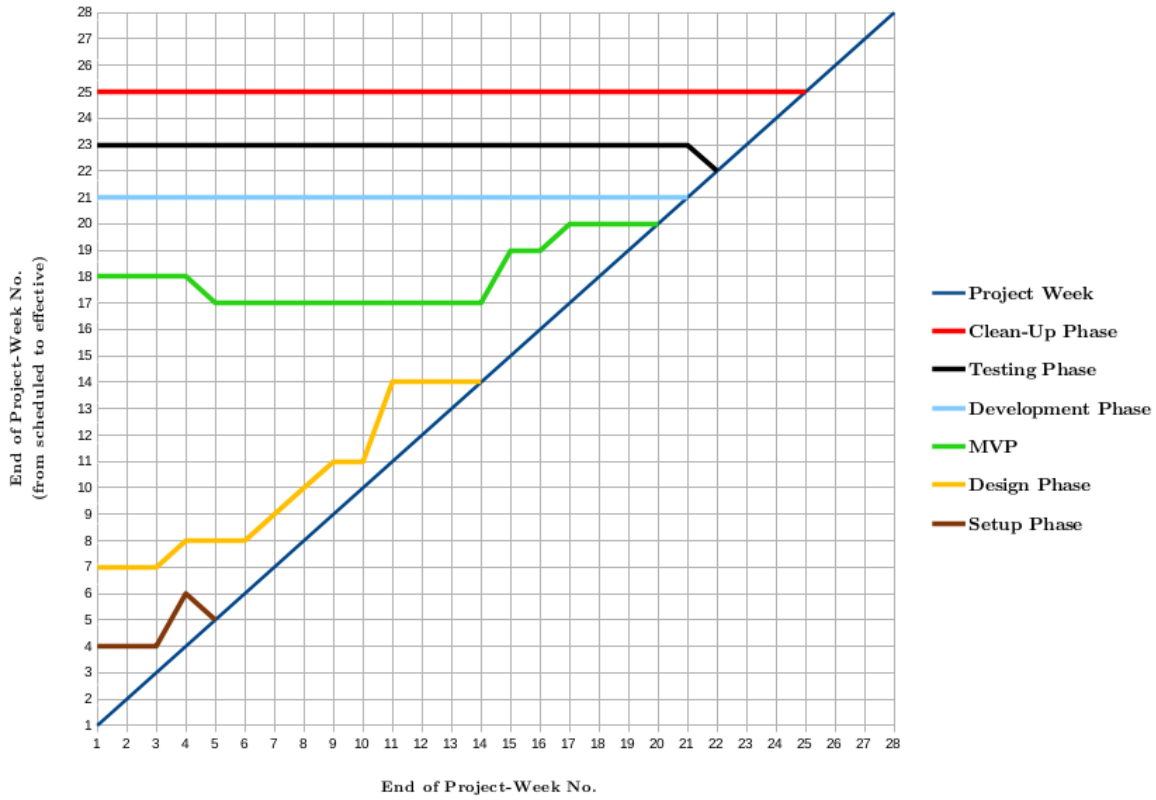
2 Lessons Learned

Looking back, there are some points with potential for improvement.

In terms of education, it would have been desirable to have already completed all the programming lessons, because topics from the last two semesters would have saved a lot of self-study time. In addition, it was no longer possible to integrate newly learned subject areas into the project, as decisions had already been made at that point. The implementation of the diploma project parallel to the last two semesters increased the stress level significantly and also contributes to the fact that students either have to repeat semesters or achieve an unavoidable lower grade, especially if they are still working at the same time, as this form of education actually provides for. While the duration of the last semester was halved, the examination load, on the other side, was only partially reduced and also compressed due to the included holiday period.

The project progress is described in more detail at the end of each individual thesis topic.

3 Milestone Trend Analysis



Project Week	Calendar Week	Sunday
1	38	21-09-26
2	39	21-10-03
3	40	21-10-10
4	41	21-10-17
5	42	21-10-24
6	43	21-10-31
7	44	21-11-07
8	45	21-11-14
9	46	21-11-21
10	47	21-11-28
11	48	21-12-05
12	49	21-12-12
13	50	21-12-19
14	51	21-12-26

Project Week	Calendar Week	Sunday
15	52	22-01-02
16	1	22-01-09
17	2	22-01-16
18	3	22-01-23
19	4	22-01-30
20	5	22-02-06
21	6	22-02-13
22	7	22-02-20
23	8	22-02-27
24	9	22-03-06
25	10	22-03-13
26	11	22-03-20
27	12	22-03-27
28	13	22-04-03

Current Version

4 Status Report

	2021						2022						
	October		November		December		January		February		March		
	17	31	14	28	12	26	9	23	6	20	6	20	30
Report No.	1	2	3	4	5	6	7	8	9	10	11	12	13

Hours Total	**Khan**	61.5	91.5	129.5	146.5	145.5	168	196	203	225.5	281.5	308.5	331	390.5
	Zeidler	68	107.5	160	167	169.5	202	252	258.5	336	458	551.5	577	660

Overall Progress													

Context *All players*													
Culture *Team members*													
Due Dates *Milestones etc.*													
Recent Work *Research etc.*													
Resources *Available time*													

No.	Open Issues	Counter Measurement	Solved on
1	The available time is very limited.	The limited time is well known in our education environment but it has to be somewhat accepted.	never

5 Recent & Upcoming Work

Date	No.	Recent Work	Upcoming Work
2021-10-17	1	Prepare documents & development environment	Finish Setup Phase & start with Design Phase
2021-10-31	2	Continue work & do research on Briar's code	Continue Design Phase
2021-11-14	3	Work on project development tasks & experiment with Briar's code	Finish Design Phase
2021-11-28	4	Update / Edit status report, milestone trend analysis, product requirements document and software requirements specification	Finish Design Phase
2021-12-12	5	Experiment with BriarJar (load contact list, remove logout functionality)	Finish Design Phase
2021-12-26	6	Finish Design Phase, keep up-to-date with project's code, work on UI, work on Dagger2, major code optimisation	Finish MVP
2022-01-09	7	Improve project structure, implement event handling, continue work on TUI & GUI, write thesis	Finish MVP
2022-01-23	8	Update product requirements document and software requirements specification	Finish MVP
2022-02-06	9	Include UML use-case & activity diagram, get first messages transmitted, work on UI & exception handling	Finish Development Phase & Testing Phase
2022-02-20	10	Finish Development Phase & Testing Phase	Finish Clean-Up Phase & deliver prototype to client
2022-03-06	11	Prepare presentation, update docs, write thesis	Finish Clean-Up Phase & deliver prototype to client
2022-03-20	12	Finish project	Finish project
2022-03-30	13	Finish project	-

6 History of Status Reports

6.1 2021-12-12

	2021						2022					
	October		November		December		January		February		March	
	17	31	14	28	12	26	9	23	6	20	6	20
Report No.	1	2	3	4	5							

Hours Total	**Khan**	61.5	91.5	129.5	146.5	145.5							
	Zeidler	68	107.5	160	167	169.5							

Overall Progress													

Context *All players*													
Culture *Team members*													
Due Dates *Milestones etc.*													
Recent Work *Research etc.*													
Resources *Available time*													

No.	Open Issues	Counter Measurement	Solved on
1	The available time is very limited.	The limited time is well known in our education environment but it has to be somewhat accepted.	DP end?
2	The Design Phase milestone has some delays, but that is okay due to the generous planned buffer to the next milestone (MVP).	The quality of the design has to be reduced.	PW 16 / CW 01
3	The upcoming time will be very difficult due to a too high workload in the HTL combined with exams. Therefore the work on the DP has to be paused regularly which is quite sad/bad.	There are hardly any possibilities to get around this issue.	
4	Same as No. 3. There was no DP related progress beside project management tasks.	Same as before.	
5	Currently no time available to work on DP.	A lot of work will be done on this during the upcoming winter holidays.	

6.2 2021-12-26

	2021						2022					
	October		November		December		January		February		March	
	17	31	14	28	12	26	9	23	6	20	6	20
Report No.	1	2	3	4	5	6						

Hours Total	**Khan**	61.5	91.5	129.5	146.5	145.5	168						
	Zeidler	68	107.5	160	167	169.5	202						

Overall Progress													

Context *All players*													
Culture *Team members*													
Due Dates *Milestones etc.*													
Recent Work *Research etc.*													
Resources *Available time*													

No.	Open Issues	Counter Measurement	Solved on
1	The available time is very limited.	The limited time is well known in our education environment but it has to be somewhat accepted.	DP end?

6.3 2022-01-09

	2021						2022					
	October		November		December		January		February		March	
	17	31	14	28	12	26	9	23	6	20	6	20
Report No.	1	2	3	4	5	6	7					

Hours Total	**Khan**	61.5	91.5	129.5	146.5	145.5	168	196					
	Zeidler	68	107.5	160	167	169.5	202	252					

Overall Progress													

Context *All players*													
Culture *Team members*													
Due Dates *Milestones etc.*													
Recent Work *Research etc.*													
Resources *Available time*													

No.	Open Issues	Counter Measurement	Solved on
1	The available time is very limited.	The limited time is well known in our education environment but it has to be somewhat accepted.	DP end?

6.4 2022-01-23

	2021						2022					
	October		November		December		January		February		March	
	17	31	14	28	12	26	9	23	6	20	6	20
Report No.	1	2	3	4	5	6	7	8				

Hours Total	**Khan**	61.5	91.5	129.5	146.5	145.5	168	196	203				
	Zeidler	68	107.5	160	167	169.5	202	252	258.5				

Overall Progress												

Context *All players*												
Culture *Team members*												
Due Dates *Milestones etc.*												
Recent Work *Research etc.*												
Resources *Available time*												

No.	Open Issues	Counter Measurement	Solved on
1	The available time is very limited.	The limited time is well known in our education environment but it has to be somewhat accepted.	DP end?

6.5 2022-02-06

	2021						2022					
	October		November		December		January		February		March	
	17	31	14	28	12	26	9	23	6	20	6	20
Report No.	1	2	3	4	5	6	7	8	9			

Hours Total	**Khan**	61.5	91.5	129.5	146.5	145.5	168	196	203	225.5			
	Zeidler	68	107.5	160	167	169.5	202	252	258.5	336			

Overall Progress												

Context *All players*												
Culture *Team members*												
Due Dates *Milestones etc.*												
Recent Work *Research etc.*												
Resources *Available time*												

No.	Open Issues	Counter Measurement	Solved on
1	The available time is very limited.	The limited time is well known in our education environment but it has to be somewhat accepted.	DP end?

6.6 2022-02-20

	2021						2022					
	October		November		December		January		February		March	
	17	31	14	28	12	26	9	23	6	20	6	20
Report No.	1	2	3	4	5	6	7	8	9	10		

Hours Total	**Khan**	61.5	91.5	129.5	146.5	145.5	168	196	203	225.5	281.5		
	Zeidler	68	107.5	160	167	169.5	202	252	258.5	336	458		

Overall Progress												

Context *All players*												
Culture *Team members*												
Due Dates *Milestones etc.*												
Recent Work *Research etc.*												
Resources *Available time*												

No.	Open Issues	Counter Measurement	Solved on
1	The available time is very limited.	The limited time is well known in our education environment but it has to be somewhat accepted.	DP end?

6.7 2022-03-06

	2021						2022					
	October		November		December		January		February		March	
	17	31	14	28	12	26	9	23	6	20	6	20
Report No.	1	2	3	4	5	6	7	8	9	10	11	

Hours Total	**Khan**	61.5	91.5	129.5	146.5	145.5	168	196	203	225.5	281.5	308.5	
	Zeidler	68	107.5	160	167	169.5	202	252	258.5	336	458	551.5	

Overall Progress												

Context *All players*												
Culture *Team members*												
Due Dates *Milestones etc.*												
Recent Work *Research etc.*												
Resources *Available time*												

No.	Open Issues	Counter Measurement	Solved on
1	The available time is very limited.	The limited time is well known in our education environment but it has to be somewhat accepted.	DP end?

6.8 2022-03-20

	2021						2022						
	October		November		December		January		February		March		
	17	31	14	28	12	26	9	23	6	20	6	20	30
Report No.	1	2	3	4	5	6	7	8	9	10	11	12	

Hours Total	**Khan**	61.5	91.5	129.5	146.5	145.5	168	196	203	225.5	281.5	308.5	331	
	Zeidler	68	107.5	160	167	169.5	202	252	258.5	336	458	551.5	577	

Overall Progress													

Context *All players*													
Culture *Team members*													
Due Dates *Milestones etc.*													
Recent Work *Research etc.*													
Resources *Available time*													

No.	Open Issues	Counter Measurement	Solved on
1	The available time is very limited.	The limited time is well known in our education environment but it has to be somewhat accepted.	DP end?

SPENGERGASSE
ausbildung mit zukunft

BRIAR

Project

BriarJar

Status Presentations (SP)

Version	1.01
Date	2021-11-03

Confidentiality	**[ ] Internal [x] External**
File Name	BriarJar_SP_20211103_004

Project Partner	Briar Project
Project Supervisor	Andreas Chwatal

Project Team	Farman Khan	UI/UX Architect & Engineer
	Alexander Zeidler	Project Leader, Software Architect & Engineer

Document Revision History

Version	Date	Name(s)	Change Description
0.01	2021-11-03	FK, AZ	Initialise document, add today's presentation "Diploma Project Introduction"
0.02	2022-02-23	FK, AZ	Add today's presentation "Advanced Status"
1.00	2022-03-30	FK, AZ	Add today's presentation "Final Presentation"; Release version 1.00
1.01	2022-06-27	AZ	Add today's presentation "Thesis Defence Presentation"
Farman Khan (FK), Alexander Zeidler (AZ)			

Table of Contents

1 2021-11-03 - Diploma Project Introduction 4

1.1 Preface 4

1.2 Slides 4

2 2022-02-23 - Advanced Status 10

2.1 Preface 10

2.2 Slides 10

3 2022-03-30 - Final Presentation 15

3.1 Preface 15

3.2 Slides 15

4 2022-06-27 - Thesis Defence Presentation 26

4.1 Preface 26

4.2 Slides 26

1 2021-11-03 - Diploma Project Introduction

1.1 Preface

This presentation was given by the BriarJar project team and informs about the diploma project's objectives and introduces the Briar messenger respectively the Briar Project. Furthermore, it enables feedback from collogues and professors.

1.2 Slides

CLIENT

Briar Project [1]

- "Secure messaging, anywhere"

BRIAR

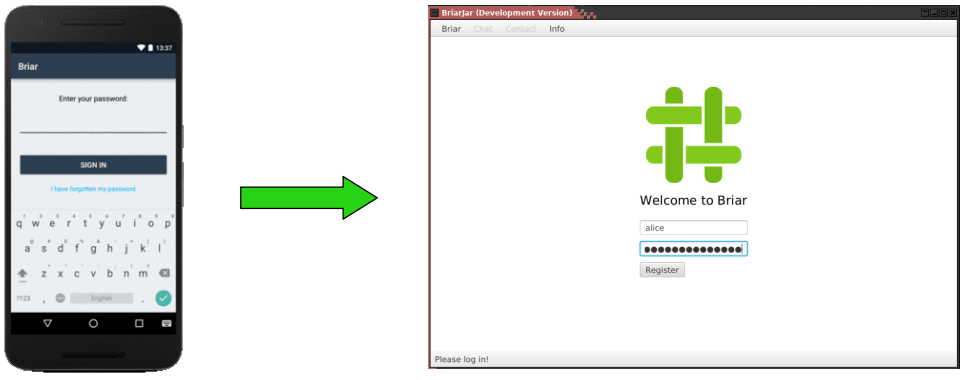
- Free and Open Source Software
- Android Messenger (Stable Release in 2018)

[1] https://briarproject.org/

3/12

PROJECT OUTPUT

Porting existing Briar Messenger to GNU/Linux Desktop



4/12

PROJECT OUTPUT

- **Prototype called “BriarJar”**
- **Mainly written in Java**
- **Both Graphical & Terminal User Interface (GUI/TUI)**
- **Platform GNU/Linux:**
  - Debian
  - Fedora
- **Build up on Briar’s Open Source code**

5/12

EXISTING CODE BASE

- **Well known Protocols**
- **Some self designed & developed Protocols**
- **Focused on using Peer-to-Peer (P2P) Communication**

- **Security Audit from Cure53 in 2017 (private beta version)**
  - discovered 1 high, 4 middle and 3 low vulnerabilities
  - “the quality and readability of the app’s source code was rather exceptional”
  - “the Briar secure messenger can be recommended for use”

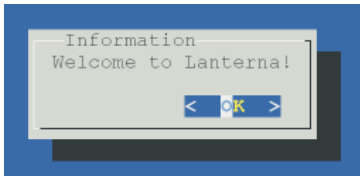
6/12

API DEVELOPMENT

- **Logical Connection to Briar's Source Code**
- **Analyse their existing Work**
  - Gradle (build automation tool - similar to Maven)
  - Dagger2 (DI framework comparable to Spring)
  - mostly Java, a bit of Kotlin
- **Design & Implement the Code Architecture**
  - regarding @Module, @Component, etc.

7/12

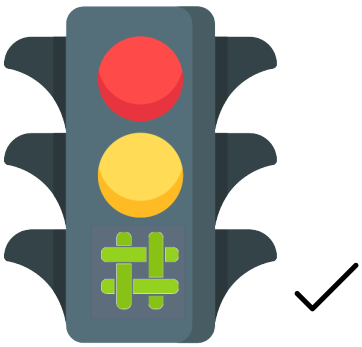
UI (GUI/TUI) DEVELOPMENT



- **Graphical User Interface**
  - Written in JavaFX
- **Terminal User Interface**
  - Written in Lanterna
- **Both will make use of "BriarJar API"**
- **Both will be evaluated:**
  - Differences?
  - Limitations?

8/12

TRAFFIC LIGHT STATUS



- **Generally good progress**
- **Some delays due to limited time**
- **Exemplary client communication**

9/12

PROGRESS

- **Setup Phase (completed)**
  - GitLab Setup
  - IntelliJ Setup
  - Research on Gradle
  - Project Initialisation
- **Design Phase (progressing)**
  - Research on Dagger2
  - System Architecture & Design Decisions

10/12

MILESTONE-TREND-ANALYSIS (MTA)

11/12

THANK YOU FOR YOUR ATTENTION

Want to learn more?

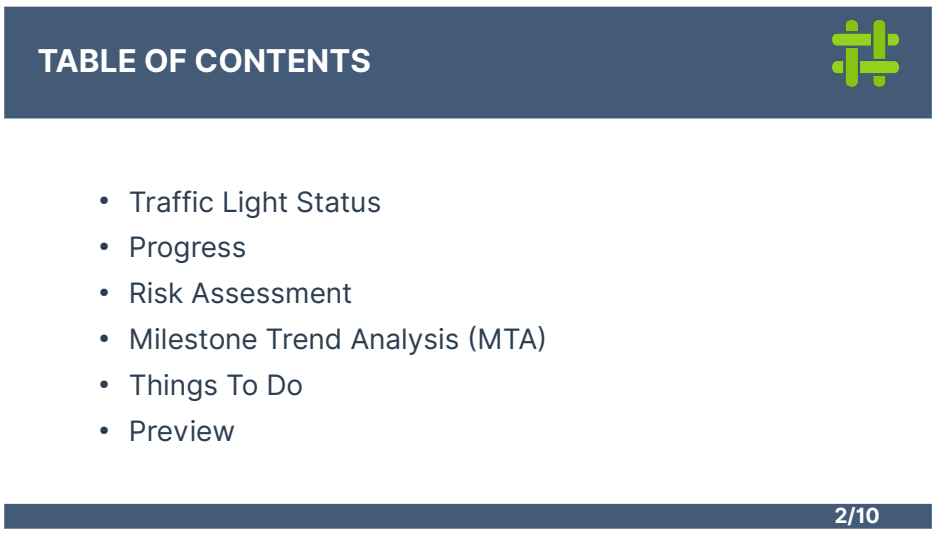
→ briarproject.org

2 2022-02-23 - Advanced Status

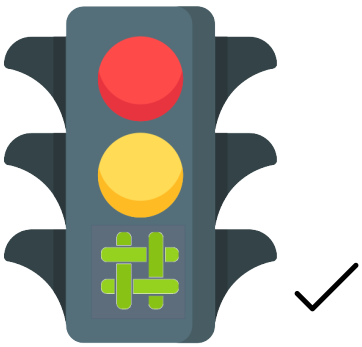
2.1 Preface

This presentation was given by the BriarJar project team and informs classmates about the diploma project's advanced status. Most of the work was finished at that time, since 5 weeks to finalise the project were left and the last shortened semester did not allow for anything else than writing the thesis.

2.2 Slides



TRAFFIC LIGHT STATUS



- Generally good progress
- Some delays due to limited time
- Exemplary client communication

3/10

RISK ASSESSMENT

Priority	Risk	Mitigation / Strategies
1	**Lacking Time Resources**	Plan ahead and use the available time on holidays.
2	**Too High Standards (Over-Perfectionism)**	Lower the standards until the requirements of the minimal viable prototype are met.
3	**Little Knowledge or Experience on Subject Matter**	Ask for support by the official Briar Project, our Supervisor or subject-related teachers.
4	**Unknown/Unclear Tasks**	Create a To-Do-List and keep regular project-internal meetings to check what has been/has to be done.

4/10

PROGRESS 1/2

- **Design Phase (completed)**
  - Major design and architectural decisions
  - Research Briar's dependencies (eg. Dagger)
  - Research on UI frameworks
- **MVP (completed)**
  - Implement basic account management
  - Implement basic contact management
  - Present one of both user interfaces

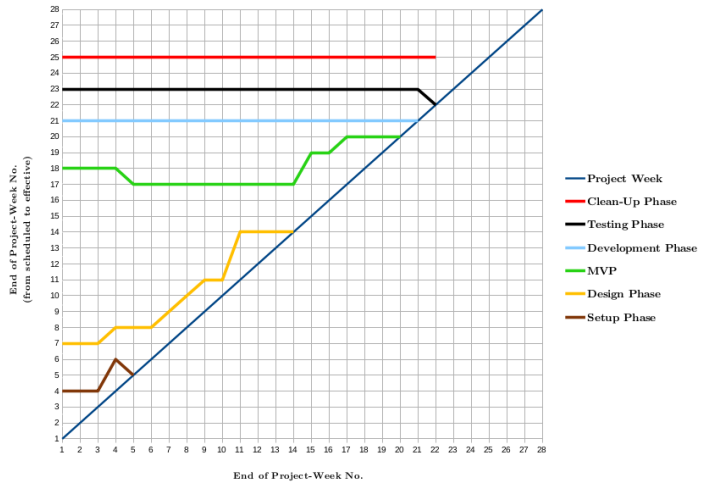
5/10

PROGRESS 2/2

- **Development Phase (completed)**
  - Complete implementation of basic chat functionality
  - Complete documentation
  - Present both user interfaces
- **Testing Phase (completed)**
  - Implement / create simple test cases
- **Clean-Up Phase (progressing)**
  - Fix bugs and clean up code

6/10

MILESTONE-TREND-ANALYSIS (MTA)



7/10

THINGS TO DO

- Clean up code & documentation (progressing)
- Deliver prototype to client
- Complete individual thesis topics
- Submit project officially

8/10

PREVIEW

We would like to show you...

- How to create an account
- How to add contacts
- How to exchange messages

9/10

THANK YOU FOR YOUR ATTENTION

Want to learn more?

→ briarproject.org

3 2022-03-30 - Final Presentation

3.1 Preface

This final presentation was given by the BriarJar project team and informs classmates about the individual topics of the diploma thesis in more detail. It begins with an overview of the project, followed by the objectives, challenges, the individual results and closes with a "Lessons Learned".

3.2 Slides

OVERVIEW 1/3 - CLIENT

Briar Project [1]

- "Secure messaging, anywhere"

BRIAR

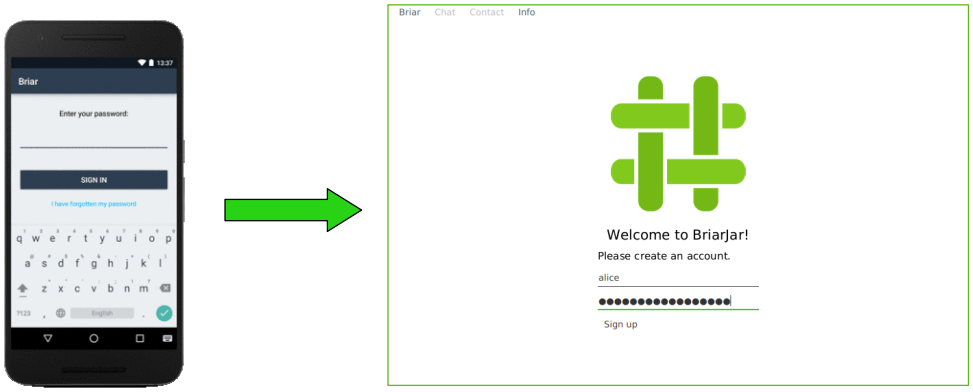
- Free and Open Source Software
- Android Messenger (Stable Release in 2018)

[1] https://briarproject.org/

3/21

OVERVIEW 2/3 – PROJECT OUTPUT

Port Existing Briar Messenger to GNU/Linux Desktop



4/21

OVERVIEW 3/3 – PROJECT OBJECTIVES

- **Desktop Messenger Prototype Called "BriarJar"**
  - Graphical and Terminal User Interface (GUI/TUI)
  - Implementation and Documentation of an API for UIs
- **Target Platform GNU/Linux**
- **Provides Briar's Basic Chat Functionalities**
  - Account Management
  - Contact Management
  - Private Conversations

5/21

Software Architecture and Implementation of Peer-to-Peer Applications

Alexander Zeidler

OBJECTIVES

- **Theoretical**
  - Comparison of Architectural Approaches for P2P Networking Applications
  - Research on Overlay Network Protocols
- **Practical**
  - API Development
  - Error Handling
  - API Documentation

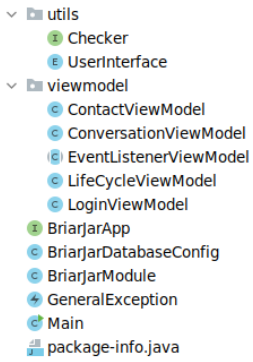
7/21

CHALLENGES

- **Dependency Injection Using Dagger2**
- **Delayed Class Design**
- **Consistency in API and Documentation**

8/21

RESULTS 1/3



9/21

RESULTS 2/3

Gets a collection of private message headers exchanged with the passed `ContactId`, useful for calling `getMessageText(MessageId)`.

Params: c – contact's `ContactId` , not null

Returns: a `Collection` of `ConversationMessageHeaders`

Throws: `GeneralException` – if compliance is not met or returning is not possible for another reason

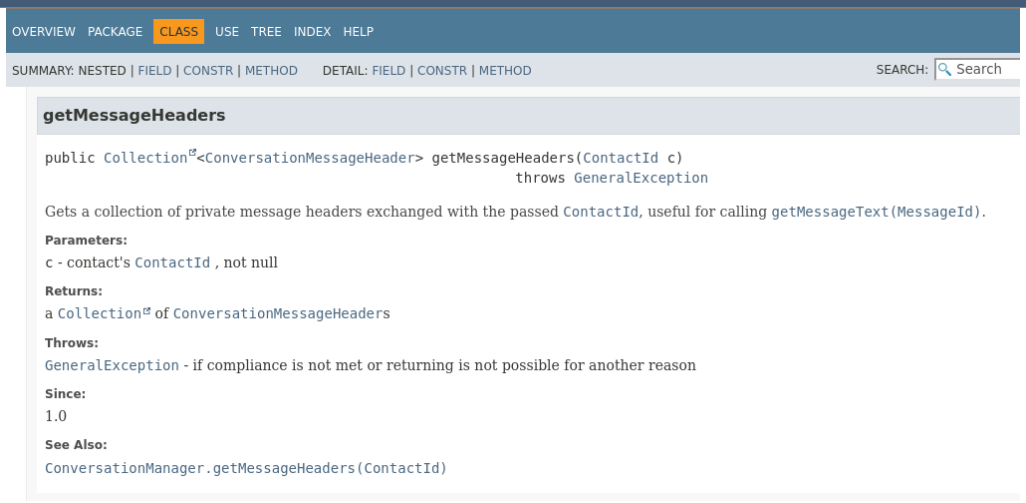
Since: 1.0

See Also: `ConversationManager.getMessageHeaders(ContactId)`

```
public Collection< ConversationMessageHeader >
      getMessageHeaders( ContactId c )
throws GeneralException
{
```

10/21

RESULTS 3/3



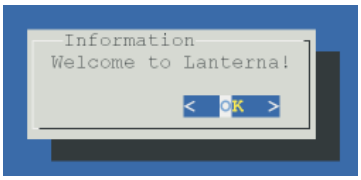
11/21

From Terminal to Graphical: Comparison of Two Different User Interface Frameworks in Java

Farman Khan

OBJECTIVES

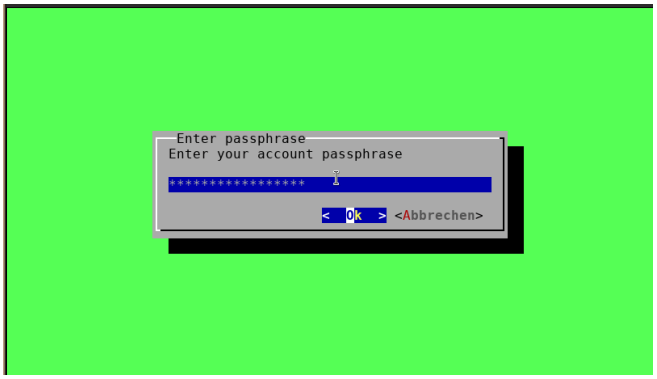
- **Theoretical**
  - Determine UI Frameworks to Use for this Project
  - Structure and Layout Both User Interfaces
  - Evaluate Differences for Developers and Users
- **Practical**
  - Implement Both User Interfaces



13/21

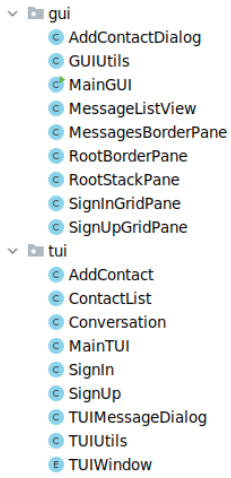
CHALLENGES

- **Structure UI Classes and Components Properly**
- **Import UI Dependencies and Java Modularity**
- **Remove TUI Artefacts**



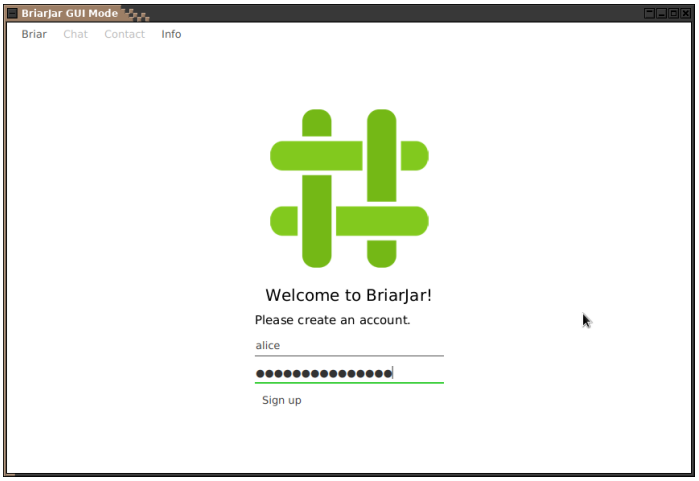
14/21

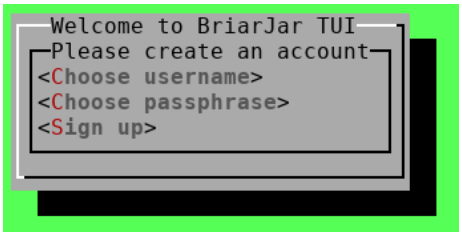
RESULTS 1/4



15/21

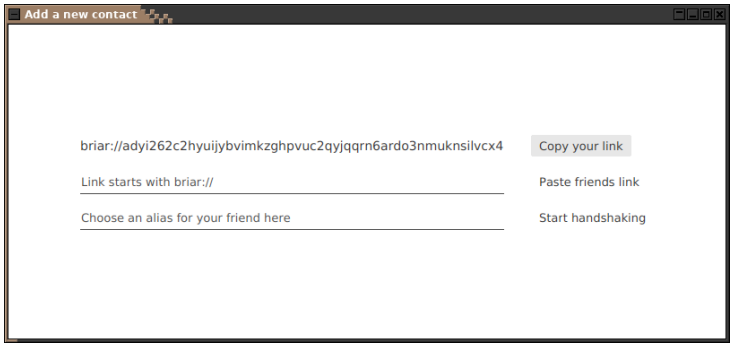
RESULTS 2/4

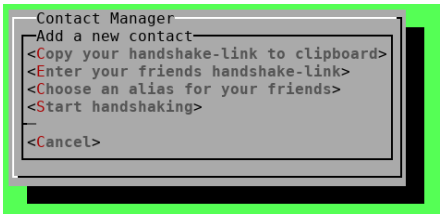




16/21

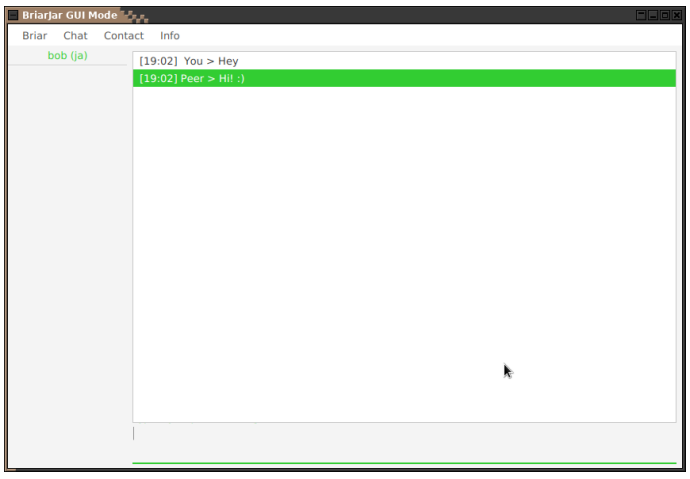
RESULTS 3/4

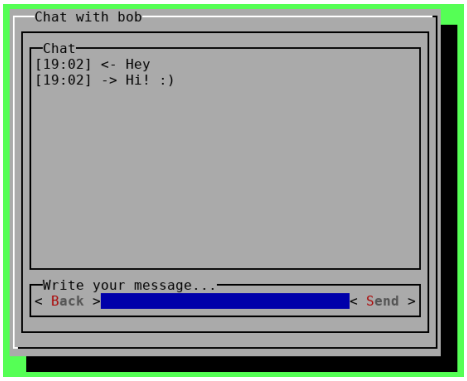




17/21

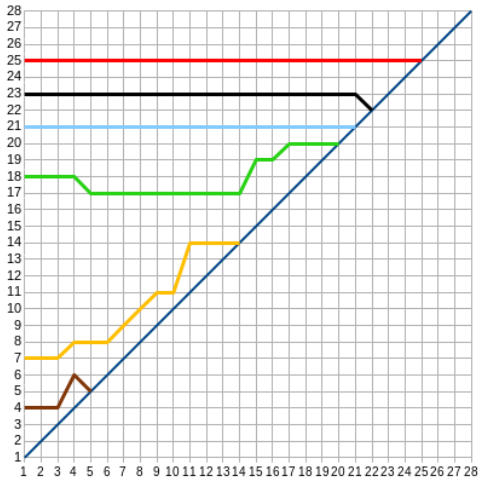
RESULTS 4/4





18/21

MILESTONE TREND ANALYSIS (MTA)



19/21

LESSONS LEARNED

- **Comfortable Atmosphere**
- **Regular Feedback and Consultations**
- **Realistic Estimation of Reachable Objectives**

- **More Communication beyond Team Members**
- **Find Balance between Reasonable Perfectionism and Available Time**

20/21


THANK YOU FOR YOUR ATTENTION
Want to learn more?
→ briarproject.org

4 2022-06-27 - Thesis Defence Presentation

4.1 Preface

This presentation was given by the BriarJar project team for the thesis defence and is attached here retrospectively for the sake of completeness. The presentation was therefore not part of the official diploma thesis submission in April. The presentation summarises the most important facts of the complete diploma project and was given in front of an audience to whom the project and the topic were partly completely new.

4.2 Slides

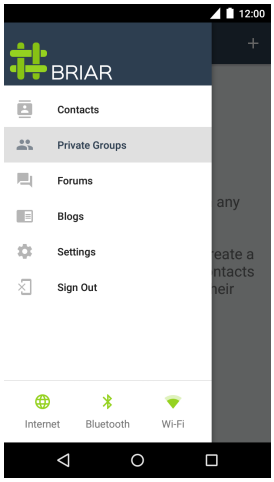
BRIAR

Briar [1]

- "Secure messaging, anywhere"

- Free and Open Source Software
- Peer-to-Peer (P2P)

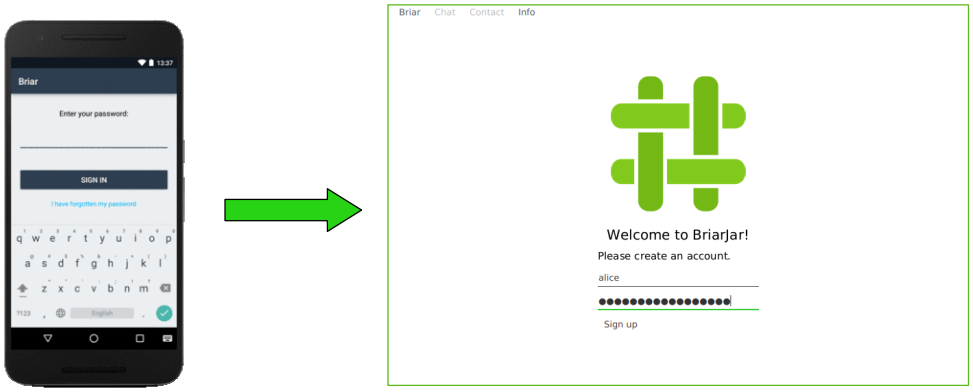
[1] https://briarproject.org/



3/28

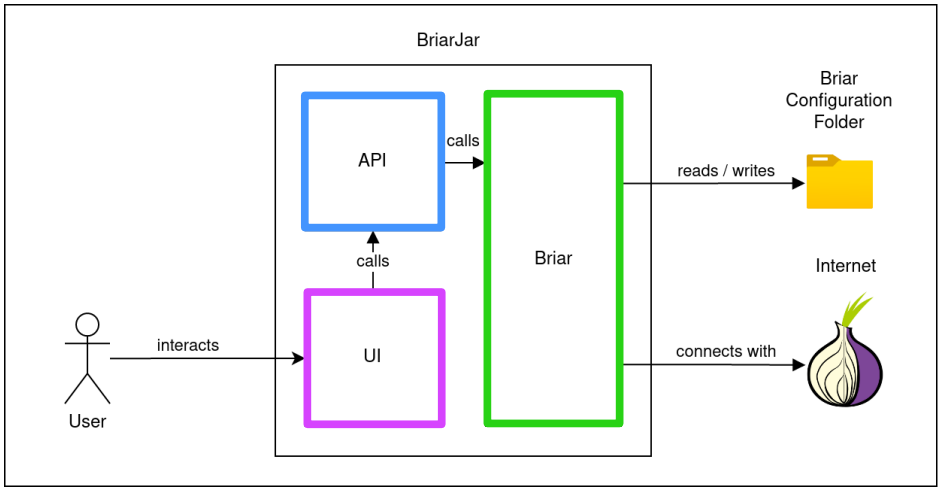
PROJECT OBJECTIVE

Port Existing Briar Messenger to GNU/Linux Desktop



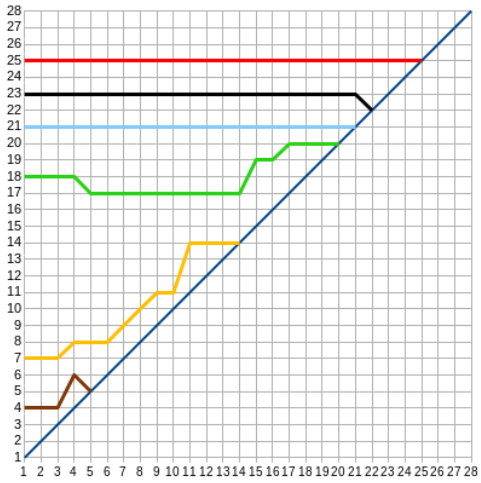
4/28

SYSTEM ARCHITECTURE



5/28

MILESTONE TREND ANALYSIS (MTA)



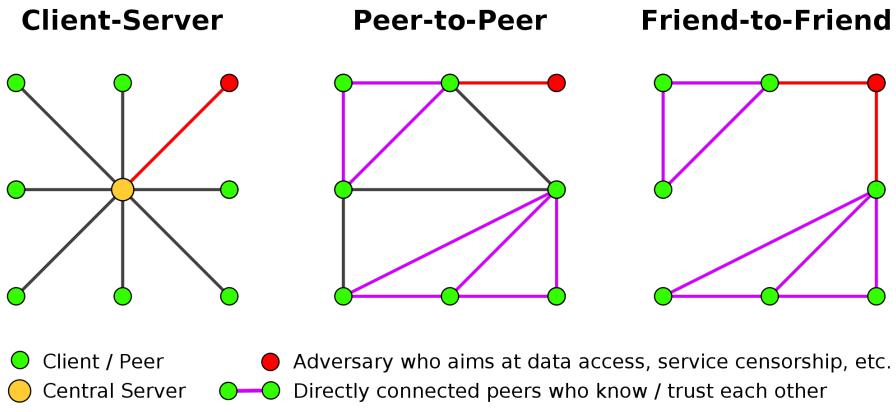
Axes... End of project-week number, from scheduled to effective

6/28

Software Architecture and Implementation of Peer-to-Peer Applications

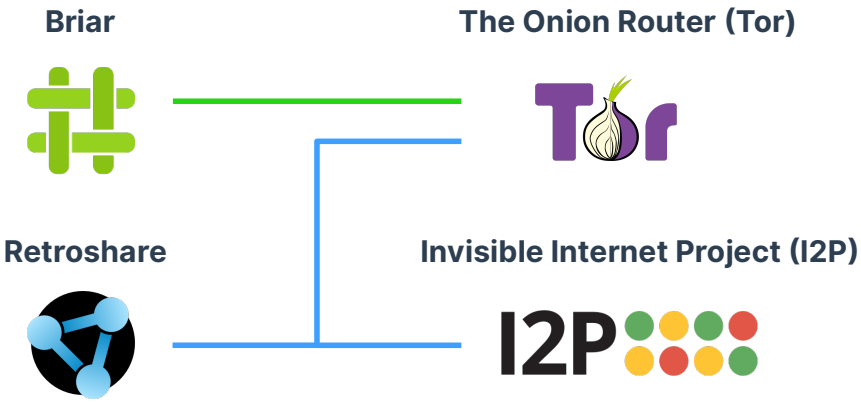
Alexander Zeidler

NETWORK TOPOLOGIES



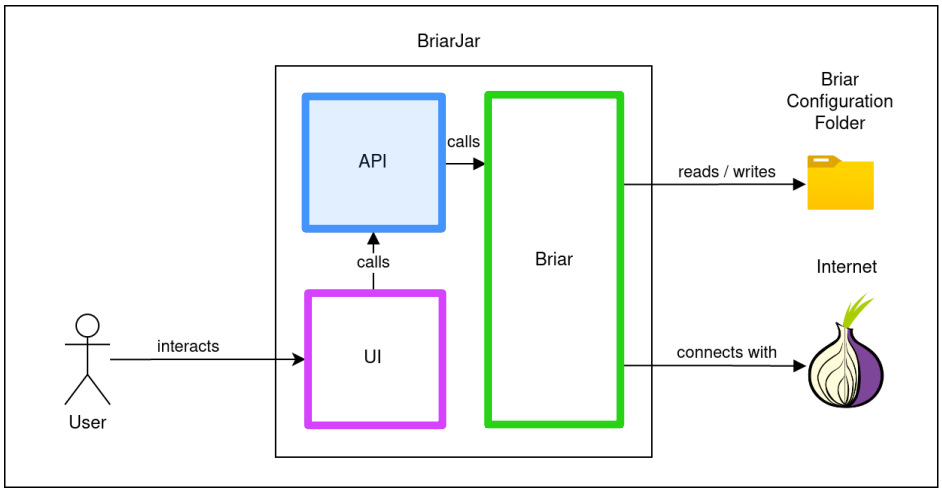
8/28

OVERLAY NETWORKS



9/28

SYSTEM ARCHITECTURE



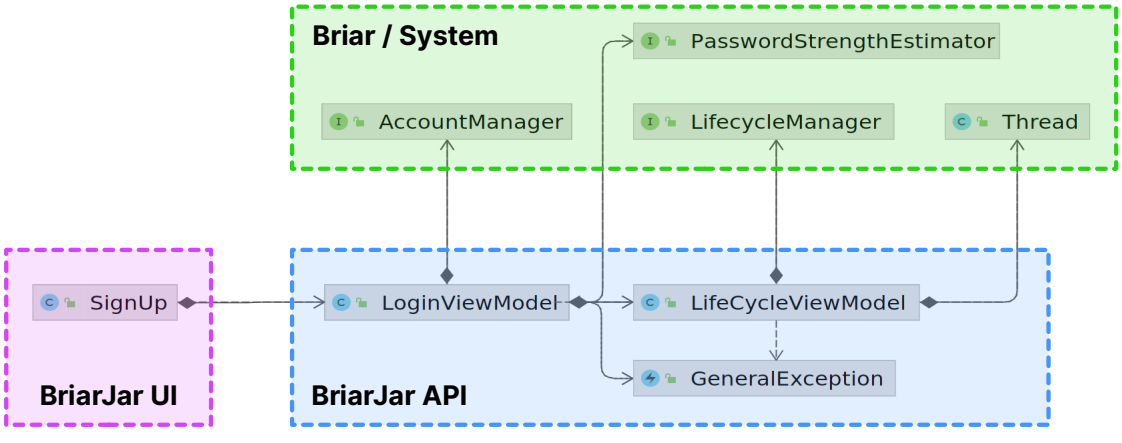
10/28

CLASS OFFER FROM BRIARJAR API

- Lifecycle
- Login
- Contact
- Conversation
- Event Listener

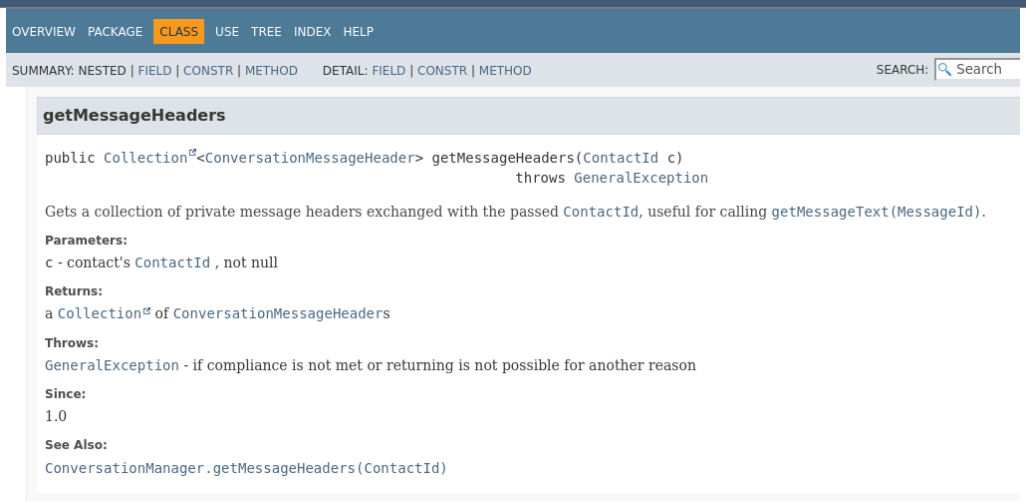
11/28

INTERACTIONS DURING SIGN-UP



12/28

DOCUMENTATION (HTML VERSION)



13/28

RESULT

- Functional
- Easy to Use
- Well Documented
- Optimised Error Handling

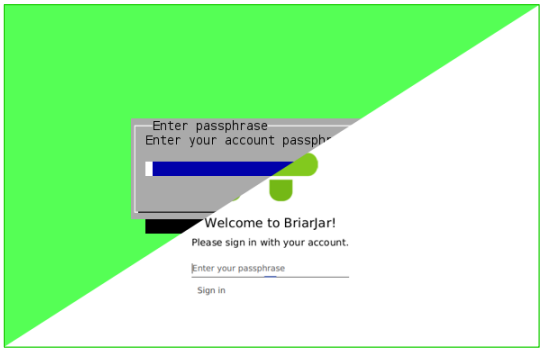
14/28

From Terminal to Graphical: Comparison of Two Different User Interface Frameworks in Java

Farman Khan

MOTIVATION & OBJECTIVES

- **Theoretical**
  - Determine **U**ser **I**nterface (UI) Frameworks
  - Determine Programming Approach
- **Practical**
  - Implement Both User Interfaces
  - Evaluate Differences
    - For Developers
    - For Users



17/28

UI FRAMEWORKS 1/2 - REQUIREMENTS

Requirement	Details	Weight
Actively maintained	Last update after February 2019	5
Well documented	Preferably JavaDoc or similar	5
Independent of native library	Pure Java, no system calls to low-level UI	3
Has Gradle dependency	Increases maintainability of project	4

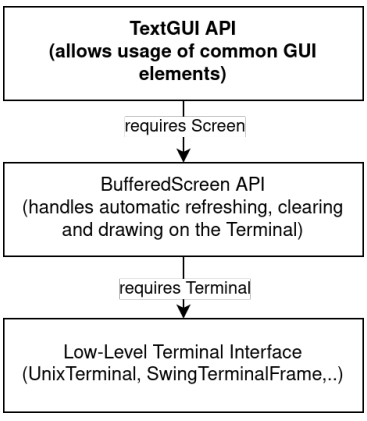
18/28

UI FRAMEWORKS 2/2 – CONSIDERATIONS

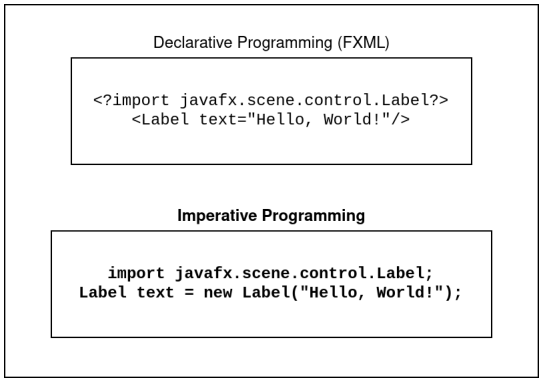
Terminal User Interface	Graphical User Interface
Java Curses (JCurses)	Abstract Window Toolkit (AWT)
CHARVA	Swing
Lanterna	**JavaFX**

19/28

APPROACH TO REALISATION



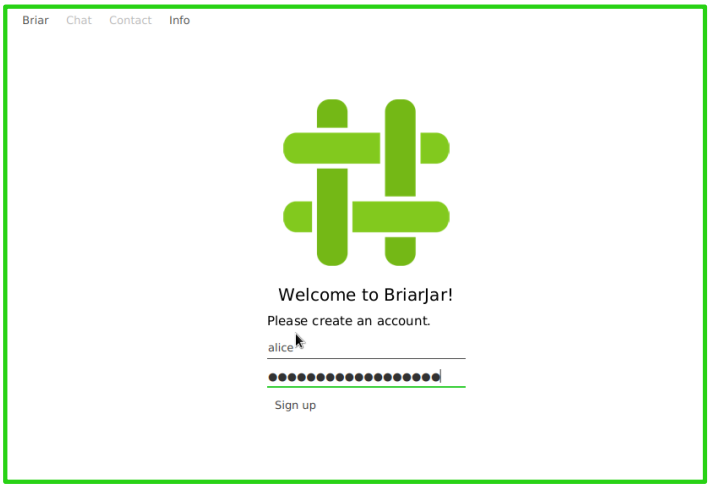
Lanterna

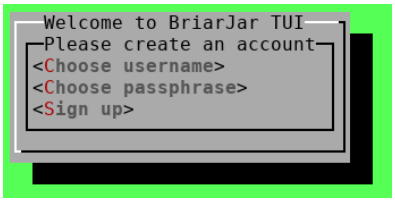


JavaFX

20/28

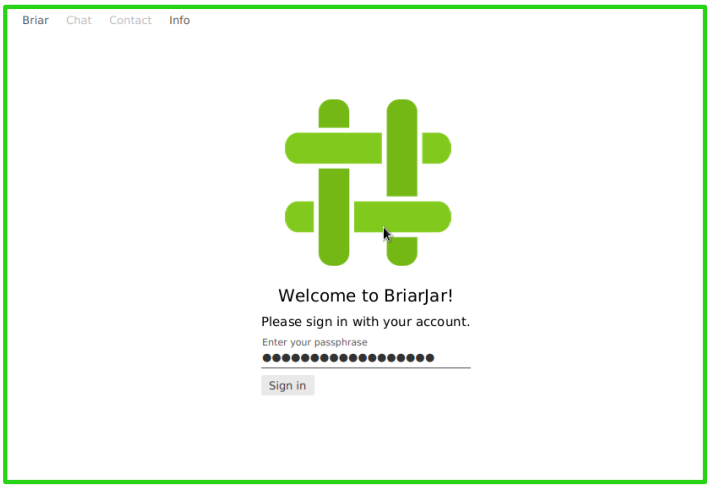
RESULTS 1/4 - SIGN UP

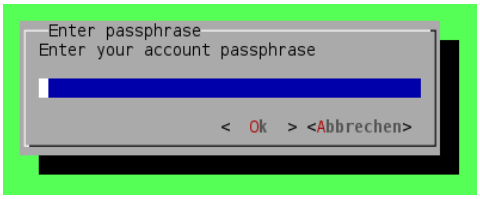




21/28

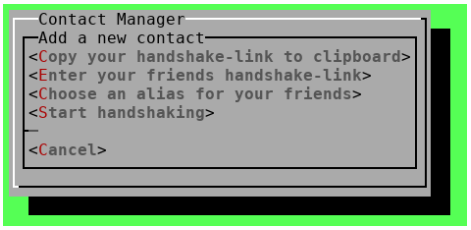
RESULTS 2/4 - SIGN IN





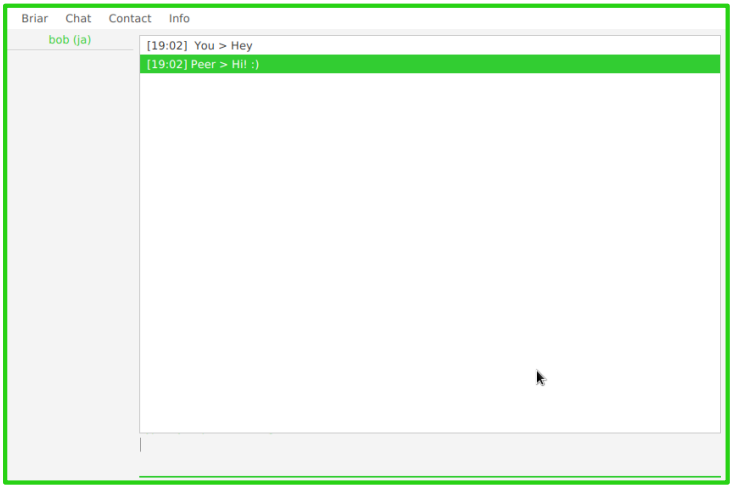
22/28

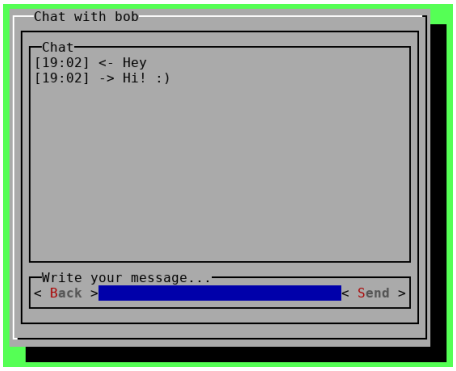
RESULTS 3/4 - ADD CONTACT



23/28

RESULTS 4/4 - MAIN SCREEN



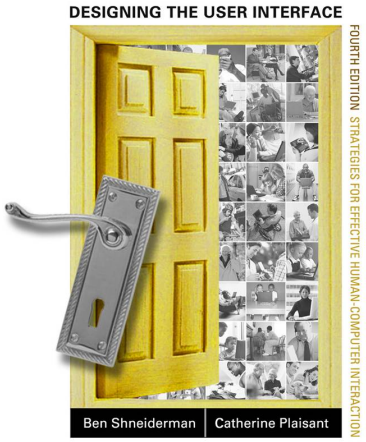


24/28

USABILITY EVALUATION 1/2 - DEFINITION

The Five Pillars Usability[2] are...

- **Time to Learn**
- **Speed of Performance**
- **Rate of Errors by Users**
- **Retention over Time**
- **Subjective Satisfaction**



[2] "Designing the User Interface" (2004) by Ben Shneiderman, Catherine Plaisant, Page 16

25/28

USABILITY EVALUATION 2/2 - RESULTS

	Terminal User Interface	Graphical User Interface
Time to Learn	6	8
Speed of Performance	8	8
Rate of Errors by Users	3	9
Retention over Time	7	9
Subjective Satisfaction	8	9
Total	**32**	**43**

26/28

CONCLUSION

- **Implemented Both User Interfaces**
- **Integrated with BriarJar API**
- **Evaluated for Usability**
- **Added Extra Features (GUI)**

THANK YOU
FOR
YOUR ATTENTION

SPENGERGASSE
ausbildung mit zukunft

BRIAR

Project

BriarJar

Meeting Protocols (MP)

Version	1.00
Date	2021-08-10

Confidentiality	**[ ] Internal [x] External**
File Name	BriarJar_MP_20210810_011

Project Partner	Briar Project
Project Supervisor	Andreas Chwatal

Project Team	Farman Khan	UI/UX Architect & Engineer
	Alexander Zeidler	Project Leader, Software Architect & Engineer

Document Revision History

Version	Date	Name(s)	Change Description
0.01	2021-08-10	FK, AZ	Initialise document and add today's meeting protocol
0.02	2021-09-15	FK, AZ	Add today's meeting protocol
0.03	2021-09-18	FK, AZ	Add today's meeting protocol
0.04	2021-09-28	FK	Add today's meeting protocol
0.05	2022-10-01	FK	Add today's meeting protocol
0.06	2022-11-05	FK	Add today's meeting protocol
0.07	2022-12-28	FK	Add today's meeting protocol
0.08	2022-02-01	FK	Add today's meeting protocol
0.09	2022-02-08	FK	Add today's meeting protocol
0.10	2022-03-15	FK	Add today's meeting protocol
1.00	2022-03-30	AZ	Release version 1.00
Farman Khan (FK), Alexander Zeidler (AZ)			

Table of Contents

1 Preface 4

2 Protocols 4

2.1 2021-08-10 (Voice) 4

2.1.1 Introductory Meeting with Briar Project Team 4

2.2 2021-09-15 (Chat) 4

2.2.1 Collaboration as Client 4

2.3 2021-09-18 (Voice) 4

2.3.1 Brief Status Report 4

2.3.2 Project Scope 4

2.4 2021-09-28 (Chat) 5

2.4.1 Brief Status Report 5

2.4.2 GitLab Repository 5

2.4.3 Technical Questions 5

2.5 2021-10-01 (Voice) 5

2.5.1 Brief Status Report 5

2.5.2 GitLab for Project Management 5

2.5.3 Gradle & Git Setup 5

2.5.4 Resolve IntelliJ Issues 5

2.6 2021-11-05 (Chat) 6

2.6.1 ViewModel Injection 6

2.7 2021-12-28 (Chat) 6

2.7.1 Internet Connectivity Issues 6

2.8 2022-02-01 (Chat) 6

2.8.1 Internet Connectivity Issues (done) 6

2.9 2022-02-08 (Chat) 6

2.9.1 Proposal to Fill the Acceptance Documents 6

2.9.2 Brief Status Videos 6

2.10 2022-03-15 (Chat) 6

2.10.1 Testing Prototype & Acceptance Document signed 6

1 Preface

This document summarises all Voice-over-IP meetings and discussions on the chat platform Mattermost between the BriarJar project team and the client Briar Project.

2 Protocols

2.1 2021-08-10 (Voice)

2.1.1 Introductory Meeting with Briar Project Team

Briar Project would gladly collaborate as our client. We have learned about Briar Project's current projects and discussed where there is room for collaboration for us. Currently, no decision has been made, since we decided to ask our supervisor first.

2.2 2021-09-15 (Chat)

2.2.1 Collaboration as Client

We have decided to collaborate with Briar Project while working semi-independently on our own desktop client "BriarJar". Nico, one of the members of the Briar Project Team, decided to sign the cooperative agreement in representation of Briar.

2.3 2021-09-18 (Voice)

2.3.1 Brief Status Report

We decided to not use the briar-headless API since it offers a REST API and WebSocket and is actually intended for use with other programming languages.

Nico also recommended us to call the functions/methods of the core code directly instead of using briar-headless.

2.3.2 Project Scope

We informed Nico about several changes. To comply with our professors instructions and diploma proposal acceptance conditions we adapted the project scope including the individual parts multiple times.

Finally, Khan will compare and deliver two user interfaces (UI), one is graphical (GUI), the other text-based (TUI).

Alex will do research on peer-to-peer applications and build an interface for Java developers on top of Briar's source code.

2.4 2021-09-28 (Chat)

2.4.1 Brief Status Report

Our project proposal is done and accepted.

2.4.2 GitLab Repository

We have asked for a private GitLab repository for our project.

2.4.3 Technical Questions

We asked questions about event handling, Dagger2 and Java-Kotlin interoperability. We received the answers in great detail by Michael very quickly.

2.5 2021-10-01 (Voice)

2.5.1 Brief Status Report

The diploma project proposal has been approved by the school's principal.

Due to the busy school schedule we may have to postpone the current milestone "Setup Phase Completed".

2.5.2 GitLab for Project Management

Nico told us mainly about GitLab's time tracking possibilities (using quick actions: /estimate & /spend with issues).

He recommended us to break down our project management related tasks using GitLab's features (Epics, Roadmaps, Milestones, Issues).

2.5.3 Gradle & Git Setup

Instead of using only Gradle for the initialisation of BriarJar on IntelliJ, Nico recommended us to use the "git submodule" command to fetch the branch "briar-as-subproject".

2.5.4 Resolve IntelliJ Issues

To research the Briar Source Code IntelliJ required the installation of Android SDK. Nico showed us a configuration command to circumvent this requirement since we don't want to build the briar-android component anyway.

2.6 2021-11-05 (Chat)

2.6.1 ViewModel Injection

Nico, Micheal and Mikolai explained how Dagger2 was utilized by the Briar Android App to inject the ViewModel classes into the UI classes. Since this type of injection may be too Android specific, it was unclear how we could utilize it.

2.7 2021-12-28 (Chat)

2.7.1 Internet Connectivity Issues

We faced an issue of BriarJar not connecting to the internet and we asked for help. Sebastian immediately replicated our issue successfully by (un)commenting some lines of code, but we still couldn't determine how to solve our issue.

2.8 2022-02-01 (Chat)

2.8.1 Internet Connectivity Issues (done)

Alex has found the source of the issue and solved it. BriarJar's internet connectivity was successful.

2.9 2022-02-08 (Chat)

2.9.1 Proposal to Fill the Acceptance Documents

Upon proposing to fill-out acceptance documents, Nico was glad to cooperate.

2.9.2 Brief Status Videos

We have sent the Briar Project videos of the progress of our current prototype and received some positive responses by the Briar Project Team.

2.10 2022-03-15 (Chat)

2.10.1 Testing Prototype & Acceptance Document signed

We talked with Nico about the acceptance document and to what extend the original template had to be filled. Nico decided to compile and run BriarJar on his computer. He was very glad with the UI/UX and experienced no errors – besides for a small TUI bug which we fixed immediately. He gave us feedback and created GitLab issues for minor things to improve. When working on those, the TUI navigation should be improved, too.

Nico contently signed our acceptance document.

SPENGERGASSE
ausbildung mit zukunft

BRIAR

Project

BriarJar

Time Log
(TL)

Version	Final
Date	2022-03-30

Confidentiality	**[x] Internal [ ] External**
File Name	BriarJar_TL

Project Partner	Briar Project
Project Supervisor	Andreas Chwatal

Project Team	Farman Khan	UI/UX Architect & Engineer
	Alexander Zeidler	Project Leader, Software Architect & Engineer

Table of Contents

1 Farman Khan_____3

2 Alexander Zeidler_____5

2.1 Implementation & Project Management.....5

2.2 Conversations (Project Partner, Supervisor).....7

2.3 Thesis (Research, Creation).....8

1 Farman Khan

Total Hours	390,5

Date	Hours	Comment
2021-03-30	12	finalise docs and prepare for printing, send for printing
2021-03-29	9	clean-up documents and cont. finalising docs
2021-03-28	4	finalise diploma thesis documents
2021-03-26	10	merge diploma thesis' and fix several formatting issues
2021-03-25	5	fix several formatting issues in diploma thesis
2021-03-24	6	simulate and discuss diploma binding to print
2021-03-23	6	write acknowledgements, discuss diploma license and briarjar email
2021-03-22	2,5	chat with client about publishing diploma thesis
2022-03-17	5,5	improve several project management documents
2022-03-15	5	fix tui bug, communication with client regarding acceptance documents
2022-03-13	6	fix some gui and tui bugs
2022-03-10	6	clean-up thesis
2022-02-26	5	cont. working on thesis
2022-02-25	12	cont. working on thesis
2022-02-24	14	cont. working on thesis
2022-02-19	3	add test cases
2022-02-18	8	merge meeting protocol document, cont. ind. thesis topic
2022-02-14	6	prepare presentation
2022-02-11	3	research on thesis topic, minor work on gui
2022-02-10	6	cont. working on gui, lot's of bugs fixed
2022-02-09	12	major redesign of GUI, bugs fixed, GUI features implemented
2022-02-08	6	work on message loading algorithms (UI)
2022-02-07	8	implment notifications on GUI, fixing shutdown bugs, env. testing
2022-02-06	5	implement GUI messageview
2022-02-04	7	re-structure tui classes and fix bug (contact list not updating)
2022-02-03	8	fix serveral exceptions, re-layouting of tui, further research
2022-02-02	3	research and implement chatbox for tui
2022-01-23	3	update project management docs
2022-01-22	3,5	cont writing on PH and LH
2022-01-02	6	begin writing thesis
2022-01-01	3	research on thesis topic, documentaries
2021-12-30	2	prepare webserver and video presentation
2021-12-29	5,5	implement for contactAdding process, merge with alex' event handling code
2021-12-28	6	start implementation of addContact handshake process
2021-12-27	5,5	fix several TUI and GUI (resource) bugs, merging code with current dagger implmentation
2021-12-26	2	meeting on Dagger troubleshooting, testing, code clean-up and status report
2021-12-26	10	cont. TUI development, successful building of fatJar with shadowJar
2021-12-24	3	clean-up & optimize much code, restructure model (ViewModelProvider), dagger optimizations
2021-12-24	6	get up-to-date with current briar code, draft (new) class diagram, discussion about further design decisions
2021-12-23	4	more research on TUI framework, experiments and first 'successful' connection using gradle
2021-12-07	6	research on gradle and using mavenCentral() to get TUI dependencies
2021-12-02	3	research on TUI framework (lanterna)
2021-11-24	2,5	progress project development documents: SRS, PRD
2021-11-22	3	edit status report and milestone trend analysis docs to fulfil project development professors' requirements
2021-11-09	1	synchronise and clean up time log
2021-11-08	5	update status report, its layout and add invested time information, prepare docs for submission (PRE course)

2021-11-07	5	fix some GUI bugs, add registration, login and password strength estimation to draft-GUI functionality
2021-11-06	7	drafting architecture diagrams, successful GUI instance connection and code-cleaning
2021-11-05	3	continue work, prepare for meeting with project partner
2021-11-03	6	prepare introduction presentation
2021-11-02	3	update GitLab issues
2021-11-01	8,5	study & implement draft-classes of BriarJar's start-procedure to combine TUI, GUI, Dagger2 & Briar libs, update PM docs, git issues & repo
2021-10-30	7	clean up code, experiment with viewmodels
2021-10-29	6,5	experiment with dagger, generate a handshake link
2021-10-28	5	get started with Dagger2 and DI
2021-10-26	5	configure IntelliJ git projects, write SRS, troubleshoot gradle
2021-10-21	4	continue product requirements doc
2021-10-13	1,5	initialise status reports document
2021-10-11	1,5	alter product requirements document
2021-10-08	6,5	initialise product requirements document, design system architecture schematic, configure BriarJar (eg. gradle)
2021-10-05	4	add git submodule to own project, init gradle file, resolve issues
2021-10-04	2,5	fix IntelliJ build errors, research git submodules.
2021-09-30	2,5	get familiar with git and GitLab
2021-09-23	4,5	finish and submit diploma application related documents
2021-09-19	1	contact supervisor and Java programming professor to talk about DP's scope
2021-09-18	12,5	work on diploma project proposal, research (a lot) on Briar's source code
2021-09-01	6	contact & join a meeting with people of Briar Project for introduce ourself and spot diploma project collaboration opportunities
2021-09-01	6	setup Nextcloud server for project docs and backups, troubleshoot NGINX.
2021-09-01	5	purchase briarjar.org domain, setup the debian server

2 Alexander Zeidler

Total Hours All Sections	660.0

2.1 Implementation & Project Management

Total Hours	444.0

Date	Hours	Comments
2022-03-30	9.0	click through an online copy shop to distribute information for Alice & Bob | finalise all documents for the Project Handbook
2022-03-29	3.0	finalise status presentation & documents
2022-03-17	2.5	update project management documents
2022-03-16	3.0	update project management documents
2022-03-13	7.0	finalise source code | add Javadoc HTML pages archive | release version 1.00 | prepare for project acceptance
2022-03-12	7.5	troubleshoot missing alias bug | read about copyright & licenses | finalise source code
2022-03-06	0.5	update status report
2022-02-24	0.5	update software architecture diagram
2022-02-23	4.5	prepare presentation
2022-02-21	7.0	synchronise document styles | create Time Log document | prepare docs | create PHBs
2022-02-20	11.0	continue work | update diagrams | update status report | prepare presentation | synchronise document styles
2022-02-19	12.0	write test documentation | test software | fix minor code issues | update SRS & PRD
2022-02-18	4.0	discuss documentation | write test documentation
2022-02-16	7.5	improve code
2022-02-15	3.0	mainly add Javadoc to GeneralException
2022-02-13	2.0	improve code
2022-02-11	10.0	write Javadoc | improve design of UI-ViewModels relationship
2022-02-10	12.5	write Javadoc | add further null checks to ViewModels
2022-02-09	10.5	start writing Javadoc
2022-02-08	13.0	read about Javadoc for APIs | try to write test classes
2022-02-07	11.0	implement shutdown functionality | read about JavaDoc | troubleshooting
2022-02-06	11.5	mainly extend & implement exception handling
2022-02-05	12.5	extend exception handling's functionality further
2022-02-04	10.5	implement proper exception handling & therby improve TUI's method to sign in
2022-02-03	8.5	implement online status visibility | improve TUI contactlist
2022-02-02	10.5	get first message transmitted | alter design of ViewModels | collect available Briar events
2022-02-01	2.0	solve a problem and get first network connection established
2022-01-31	2.0	get feedback from a project manager professor and create new use-case diagram

2022-01-28	1.0	correct UML diagrams and move them to git
2022-01-27	11.5	complete activity diagram
2022-01-26	6.5	study plantuml as IntelliJ plugin | create use-case & activity diagram
2022-01-23	6.5	continue work on SRS and PRD | try new code style | update status report
2022-01-09	1.0	update status report

2021-12-31	1.0	experiment with code style
2021-12-30	2.5	experiment with running the FatJAR output file | provide status report to supervisor
2021-12-29	9.0	implement event handling & adapt existing code
2021-12-28	10.0	analyse startup procedure of Briar code & event handling
2021-12-27	5.0	improve project structure (source code) | alter code style | troubleshoot execution errors of builded FatJAR | start with event handling
2021-12-26	10.5	finish Dagger2 implementation so far | improve mainly funtional interactions
2021-12-25	12.0	implement some ViewModels | implement & troubleshoot Dagger2 code further
2021-12-24	9.0	read through current BriarJar implementation | alter sign in/up code | alter wording | discuss about class-design | start implementing contact & messages handling
2021-12-23	1.0	update repos and check current state
2021-12-12	1.0	update status report

2021-11-29	1.5	go through updated documents | experiment with BriarJar (load contact list, remove logout functionality)
2021-11-28	0.5	update status report
2021-11-24	2.5	continue work on SRS and PRD
2021-11-22	1.0	continue work on PRD
2021-11-22	3.0	edit status report and milestone trend analysis docs to fulfil project development professors' requirements
2021-11-09	1.0	synchronise and clean up time log
2021-11-08	5.0	update status report, its layout and add invested time information | prepare docs for submission (PRE course)
2021-11-07	11.0	finish Dagger2 research for now, update simple Dagger2 example & create an additional one (field injection still missing)
2021-11-06	7.5	draft classes design (how they work together), experiment how to switch GUI screens (register, login, main)
2021-11-05	3.0	continue work | prepare for meeting with project partner
2021-11-04	2.5	Dagger2 related stuff like creating show case classes
2021-11-03	5.0	finish DP introduction presentation | cleaned up cloud
2021-11-02	8.0	prepare DP introduction presentation & do some more research for it | start writing about Dagger2 in SRS doc
2021-11-01	8.5	study & implement draft-classes of BriarJar's start-procedure to combine TUI, GUI, Dagger2 & Briar libs | update PM docs, git issues & repo

2021-10-31	10.0	study Dagger2 and walktrough Briar's implementation
2021-10-30	7.0	inspect Briar's structure and code deeper and create first structure of classes
2021-10-29	5.0	experiment with Dagger2 and call Briar's code

2021-10-28	5.0	get started with Dagger2 (dependency injection framework)
2021-10-26	5.0	configure IntelliJ git projects | write Software Requirements Specification (SRS) | troubleshoot gradle
2021-10-22	2.5	continue work
2021-10-21	3.5	rewrite certain chapters in PRD
2021-10-20	1.5	create MTA (milestone trend analysis) document, review product requirement doc
2021-10-13	1.5	initialise status reports doc
2021-10-11	1.5	alter PRD
2021-10-08	6.5	initialise Product Requirements Document (PRD), design system architecture schematic, configure BriarJar (e.g. gradle)
2021-10-06	3.0	upload existing documents to cloud storage
2021-10-05	1.0	collect & prepare documents | discuss BriarJar IntelliJ project configuration
2021-10-04	1.0	clean up cloud storage

2021-09-30	2.5	get familiar with git and GitLab
2021-09-29	3.5	configure git repository and work on build problems in IntelliJ
2021-09-23	4.5	finish and submit diploma application related documents
2021-09-19	1.0	contact supervisor and Java programming professor to talk about DP's scope
2021-09-18	12.5	work on diploma project proposal, research of Briar's source code
2021-09-00	20.0	research, design and create DP related template documents

2.2 Conversations (Project Partner, Supervisor)

Total Hours	**14.5**

Date	Hours	Comments
2022-03-23	1.0	meeting with supervisor
2022-03-16	1.0	chat with client about project acceptance
2022-03-14	1.0	chat with client about project acceptance
2022-02-03	1.0	provide supervisor with updates
2021-11-05	1.0	talk with client about Dagger2, system architecture and general news on both sides
2021-10-10	0.5	provide supervisor with updates
2021-09-23	2.5	review diploma proposal with supervisor
2021-09-18	0.5	talk with client about TUI clients
2021-09-00	6.0	contact & join a meeting with people of Briar Project for introduce ourself and spot diploma project collaboration opportunities

2.3 Thesis (Research, Creation)

Total Hours	201.5

Date	Hours	Comments
2022-03-29	7.0	revise & complete thesis document
2022-03-28	5.5	complete thesis & common parts | merge theses to a single document
2022-03-27	14.0	revise thesis & common parts | merge theses to a single document
2022-03-26	14.0	revise thesis & common parts
2022-03-25	12.5	revise thesis & common parts
2022-03-24	7.0	revise thesis & common parts
2022-03-23	10.0	write conclusion | revise thesis & common parts | research about publication
2022-03-12	3.5	improve bibliography | update thesis to reflect current source code
2022-03-04	14.0	research | finish thesis & wait for feedback
2022-03-03	8.0	research | write thesis
2022-03-02	12.0	research | write thesis
2022-03-01	10.0	research | write thesis
2022-02-28	8.0	research | write thesis
2022-02-27	10.0	revise thesis | research
2022-02-26	8.5	revise thesis | write German abstract & Introduction
2022-02-25	9.0	revise thesis
2022-02-24	1.5	prepare collective thesis document
2022-02-15	6.0	write thesis
2022-02-14	5.5	write thesis
2022-02-13	8.0	write thesis
2022-02-12	6.0	write thesis
2022-01-01	9.5	research | write thesis
2021-12-31	9.0	read "Beautiful Code" | do some research | write thesis
2021-12-30	3.0	update table of contents | overview the books "Peer to Peer: Harnessing the Power of Disruptive Technologies" & "Beautiful Code"